Vannarho Core Risk Engine - User Guide

Growth Mindset Pty Ltd trading as Vannarho

24 March 2026
VRE version 0.11.2

Document History

Date	Author	Comment
7 October 2016 - 19 June 2020	Quaternion	updates for releases 1- 5
30 June 2021 - 6 October 2023	Acadia	updates for release 6 - 11
21 December 2023	Vannarho	updates release 0.1.0
3 June 2024	Vannarho	updates for release 0.2.0
1 December 2024	Vannarho	updates for release 0.3.0
26 July 2025	Vannarho	updates for VRE release 0.4.0
16 March 2026	Vannarho	updates for VRE release 0.5.0 - 0.10.0
22 March 2026	Vannarho	updates for VRE release 0.11.0 - 0.11.1, including Pillar 2 banking-book execution, IMA validation/publication, and maintained reporting examples
23 March 2026	Vannarho	updates for VRE release 0.11.2, including Pillar 3 first-class coverage, forward banking-book example paths, and expanded regulatory example closure
24 March 2026	Vannarho	refreshed current-execution and example documentation, added generated maintained-toggle inventory, and aligned path references to current maintained directories

Contents

1 Introduction **14**
- 1.1 Scope . . . 14
- 1.2 VRE in Python . . . 18
- 1.3 Roadmap . . . 18
- 1.4 Further Resources . . . 19

2 VRE Data Flow **20**

3 Current Execution Surface **21**

4 Current Outputs, Reporting and Evidence **23**

5 Regulatory Compliance and Auditability **24**

6 Shared Boundary With VRE-RaaS **26**

7 Config Doctor **27**

8 Getting and Building VRE **28**
- 8.1 VRE Releases . . . 28
- 8.2 Access via Git . . . 29
- 8.3 Prerequisites for Building VRE . . . 30
- 8.4 Building VRE . . . 33

9 Examples **35**
- 9.1 Maintained Regulatory and Reporting Examples . . . 37
  - 9.1.1 Maintained reporting true-flow estate . . . 37
  - 9.1.2 Current E2E lanes . . . 38
  - 9.1.3 How to read the examples . . . 38
  - 9.1.4 Pillar 2 and IMA examples . . . 38
- 9.2 Getting Started . . . 44
- 9.3 Minimal Setup . . . 44
- 9.4 Products . . . 44
  - 9.4.1 Scripted Trade . . . 46
- 9.5 Curve Building . . . 47
  - 9.5.1 Bootstrap Consistency . . . 48
  - 9.5.2 Discount Ratio Curves . . . 48
  - 9.5.3 Fixed vs Float Cross Currency Helpers . . . 49
  - 9.5.4 USD-Prime Curve Building . . . 49
  - 9.5.5 Bond Yield Shifted Curves . . . 49
  - 9.5.6 Central Bank Meeting Dates . . . 50
  - 9.5.7 SABR Volatility Surfaces . . . 50
- 9.6 Reporting . . . 50
- 9.7 Market Risk . . . 54
  - 9.7.1 Sensitivity Analysis . . . 54
  - 9.7.2 Sensitivity Analysis and Pricing with Smile . . . 57
  - 9.7.3 Parametric VaR . . . 57
  - 9.7.4 Stress Testing . . . 57

9.7.5 Stress Testing in the Par Domain . . . 58
9.7.6 Stressed Sensitivity Analysis . . . 58
9.7.7 Historical Simulation VaR . . . 58
9.7.8 SA-MR Simplified Market Risk Capital . . . 59
9.7.9 SA-MR Sensitivities-Based Method (SBM) . . . 60
9.7.10 IMA Pure . . . 60
9.7.11 IMA and SA-MR Mixed Desk Parity . . . 61
9.7.12 Multi-Entity Regulatory Scope (SA-MR) . . . 61
9.7.13 P&L and P&L Explain . . . 62
9.7.14 Stand-alone Par Conversion Utility . . . 63
9.7.15 Base Scenario Utility . . . 64
9.7.16 Zero Shift to Par Shift Conversion Utility . . . 64
9.8 CVA Risk . . . 64
9.8.1 BA-CVA Variants . . . 64
9.8.2 SA-CVA Variants . . . 65
9.8.3 Multi-Entity Regulatory Scope (SA-CCR → SA-CVA) . . . 66
9.8.4 SA-CVA Backend Parity (FD, AAD, GPU) . . . 66
9.8.5 MAR50 Suite Wrappers . . . 67
9.9 Initial Margin . . . 67
9.9.1 ISDA SIMM and IM Schedule . . . 67
9.9.2 Dynamic Initial Margin and MVA . . . 69
9.9.3 Dynamic SIMM . . . 71
9.10 Exposure Family: Non-Collateral . . . 73
9.10.1 Swap with flat yield curve . . . 75
9.10.2 Swap with normal yield curves . . . 76
9.10.3 Forward Rate Agreement . . . 76
9.10.4 Swaptions . . . 77
9.10.5 Cap/Floor . . . 80
9.10.6 FX Forward and FX Option . . . 81
9.10.7 Non-Resetting and Resetting Cross Currency Swaps . . . 82
9.10.8 Equity Derivatives . . . 83
9.10.9 Commodity Derivatives . . . 83
9.10.10 Inflation CPI and YOY Swap - TODO . . . 84
9.10.11 Credit Default Swap . . . 84
9.10.12 Capped/Floored (Digital) CMS Spread . . . 86
9.10.13 Capped (Digital) CMS Spread with Formula-based Payoff . . . 86
9.10.14 Long-term simulation . . . 87
9.10.15 Choice of Measure . . . 88
9.10.16 Simulation in the two-factor Hull-White model . . . 89
9.10.17 Wrong-Way-Risk . . . 91
9.10.18 Flip View . . . 92
9.11 Exposure Family: Collateral . . . 92
9.11.1 MPoR (Biweekly) Grid . . . 92
9.11.2 Close-out Grid . . . 96
9.11.3 First MPoR Adjustment . . . 97
9.12 Exposure Family: Methods . . . 98
9.12.1 Benchmarking AMC vs Classic Simulation . . . 99
9.12.2 Scripted Bermudan . . . 105

9.12.3 Scripted TaRF . . . 106
9.12.4 Forward Bond . . . 106
9.12.5 Overlapping Close-out Grids in AMC . . . 106
9.12.6 Extract Scenario Statistics . . . 107
9.13 XVA Risk . . . 107
9.13.1 XVA Stress Testing . . . 107
9.13.2 XVA P&L Explain . . . 107
9.13.3 XVA Sensitivities . . . 108
9.13.4 Regulatory CVA Capital Moved to CVA Risk . . . 108
9.14 Credit Risk . . . 109
9.14.1 SA-CCR . . . 109
9.14.2 IMM . . . 110
9.14.3 Credit Portfolio Model . . . 111
9.15 Performance . . . 112
9.15.1 Multi-threading . . . 112
9.15.2 NPV Sensitivities with AAD and GPUs . . . 112
9.15.3 CVA Sensitivities with AAD . . . 112
9.16 VREPython . . . 113

10 Current Parameterisation And Runtime Contracts **115**

11 Legacy XML Reference Status **116**

12 Parameterisation **117**
12.1 Analytics: `vre.toml` and resolved run bundles . . . 118
12.1.0.1 Run identity and packaging . . . 119
12.1.0.2 Market and fixing data . . . 119
12.1.0.3 Resolved runtime contracts . . . 120
12.1.0.4 Portfolio payloads and retained XML exceptions . . . 120
12.1.0.5 Calendar and currency extensions . . . 121
12.1.0.6 Operational behaviour . . . 121
12.1.1 Logging . . . 121
12.1.2 Pricing, Cashflows, Curves . . . 122
12.1.2.1 NPV . . . 123
12.1.2.2 Cashflows . . . 123
12.1.2.3 Curves and credit curves . . . 123
12.1.2.4 Portfolio details and auxiliary reports . . . 123
12.1.3 Simulation and Model Calibration . . . 123
12.1.3.1 Calibration . . . 123
12.1.3.2 Simulation . . . 123
12.1.3.3 Compiled host CE controls . . . 124
12.1.3.4 Compiled-AAD and external-device controls . . . 125
12.1.3.5 Scenario and scenario statistics . . . 126
12.1.3.6 PFE . . . 127
12.1.4 Value Adjustments . . . 127
12.1.5 Sensitivity and Stress Testing . . . 127
12.1.5.1 Sensitivity . . . 127
12.1.5.2 Stress and sensitivity stress . . . 128
12.1.5.3 Conversion analytics . . . 128

12.1.6 Value at Risk . . . 128
12.1.7 P&L, P&L Explain, ZeroToParShift, Scenario . . . 128
12.1.8 Initial Margin: ISDA SIMM, IM Schedule, CRIF Generation . . . 129
12.1.9 XVA Stress Testing . . . 129
12.1.10 XVA Sensitivity . . . 129
12.1.11 XVA Explain . . . 129
12.1.12 CCR Capital: SA-CCR and IMM . . . 130
12.1.12.1 SA-CCR . . . 130
12.1.12.2 IMM . . . 130
12.1.13 CVA Capital: SA-CVA and BA-CVA . . . 130
12.1.14 Market Risk: SA-MR and IMA . . . 131
12.2 Market View And Today's Market Publication . . . 132
12.2.1 Discounting Curves . . . 133
12.2.2 Index Curves . . . 133
12.2.3 Yield Curves . . . 134
12.2.4 Swap Index Curves . . . 134
12.2.5 FX Spot . . . 134
12.2.6 FX Volatilities . . . 134
12.2.7 Swaption Volatilities . . . 134
12.2.8 Cap/Floor Volatilities . . . 135
12.2.9 Default Curves . . . 135
12.2.10 Securities . . . 135
12.2.11 Equity Curves . . . 135
12.2.12 Equity Volatilities . . . 135
12.2.13 Inflation Index Curves . . . 135
12.2.14 Inflation Cap/Floor Volatility Surfaces . . . 135
12.2.15 CDS Volatility Structures . . . 136
12.2.16 Base Correlation Structures . . . 136
12.2.17 Correlation Structures . . . 136
12.2.18 Commodity Curves and Volatilities . . . 136
12.2.19 Market Configurations . . . 136
12.3 Simulation Runtime And Credit Simulation . . . 137
12.3.1 Parameters . . . 137
12.3.1.1 XVA runtime controls . . . 138
12.3.1.2 CPU runtime controls . . . 138
12.3.2 Model . . . 138
12.3.2.1 Cross-asset model profile . . . 138
12.3.2.2 Cross-asset correlations . . . 140
12.3.3 Market . . . 140
12.3.4 CAM extension controls and strict modes . . . 140
12.3.4.1 Credit simulation . . . 141
12.4 Sensitivity Runtime And Shift Tables . . . 142
12.4.1 FactorSpec contract and sensitivity axis canonicalization . . . 144
12.4.1.1 Discount curves . . . 145
12.4.1.2 Credit curves . . . 145
12.4.1.3 FX spots . . . 145
12.4.1.4 Swaption volatilities . . . 145
12.4.1.5 Base correlations . . . 145

12.5 Stress Packs And Scenario Inputs . . . 147
12.5.0.1 Curve-like families. . . . 150
12.5.0.2 Spot-like families. . . . 151
12.5.0.3 FX-volatility family. . . . 151
12.5.0.4 Other volatility families. . . . 152
12.6 Calendar Adjustments Publication . . . 154
12.7 Authored seed . . . 155
12.8 Calendar records . . . 155
12.9 Base-calendar derivation rules . . . 157
12.10 Compatibility XML and the CUSTOM_ prefix . . . 157
12.11 Practical examples . . . 157
12.11.0.1 Overlaying an existing calendar. . . . 157
12.11.0.2 Creating a derived calendar. . . . 157
12.12 Operational interpretation . . . 158
12.13 Curve Set And Runtime Curve Configuration . . . 159
12.13.1 Yield and Default Curve Structures . . . 160
12.13.1.1 Yield / YieldCurve . . . 160
12.13.1.2 Default / DefaultCurve . . . 160
12.13.2 Swaption Volatility Structures . . . 161
12.13.3 FX Volatility Structures . . . 161
12.13.4 Equity Curve Structures . . . 161
12.13.5 Equity Volatility Structures . . . 162
12.13.6 Inflation Curves . . . 162
12.13.7 Inflation Cap/Floor Volatility Surfaces . . . 162
12.13.8 CDS Volatilities . . . 163
12.13.9 Base Correlations . . . 163
12.13.10 Commodity Curves and Volatilities . . . 163
12.13.10.1 CommodityCurve . . . 163
12.13.10.2 CommodityVolatility . . . 163
12.13.11 Securities . . . 164
12.13.12 Correlations . . . 164
12.13.13 Bootstrap Configuration . . . 164
12.13.14 One Dimensional Solver Configuration . . . 164
12.14 Reference Data Manifest And Registries . . . 165
12.15 Manifest entries . . . 166
12.16 Domain registry records . . . 167
12.17 Bond reference data . . . 167
12.18 Resolved runtime view . . . 169
12.19 Compatibility XML . . . 170
12.20 Operational interpretation . . . 170
12.21 IBOR Fallback Policy . . . 171
12.22 Authored policy seed . . . 172
12.23 Global settings . . . 172
12.24 Fallback rules . . . 173
12.25 Built-in default configuration . . . 175
12.26 Testing utilities and switch-date overrides . . . 175
12.27 Compatibility XML . . . 175
12.28 Operational interpretation . . . 176

12.29 SIMM Registry And Calibration . . . 177
12.29.1 SIMM Calibration . . . 177
12.29.2 General Structure . . . 178
12.29.3 Interest Rate . . . 178
12.29.4 Credit Qualifying . . . 179
12.29.5 Credit Non-Qualifying . . . 179
12.29.6 Equity . . . 179
12.29.7 Commodity . . . 180
12.29.8 FX . . . 180
12.29.9 Risk-class correlations . . . 180
12.29.10 Operational rule . . . 181
12.30 Convention Manifest And Registry . . . 182
12.30.1 Zero Conventions . . . 182
12.30.2 Deposit Conventions . . . 183
12.30.3 Future Conventions . . . 183
12.30.4 FRA Conventions . . . 183
12.30.5 OIS Conventions . . . 183
12.30.6 Swap Conventions . . . 184
12.30.7 Average OIS Conventions . . . 184
12.30.8 Tenor Basis Swap Conventions . . . 184
12.30.9 Tenor Basis Two Swap Conventions . . . 184
12.30.10 BMA / SIFMA Basis Swap Conventions . . . 185
12.30.11 FX Conventions . . . 185
12.30.12 Cross Currency Basis Swap Conventions . . . 185
12.30.13 Cross Currency Fix-Float Conventions . . . 185
12.30.14 Inflation Swap Conventions . . . 186
12.30.15 CMS Spread Option Conventions . . . 186
12.30.16 Ibor Index Conventions . . . 186
12.30.17 Overnight Index Conventions . . . 187
12.30.18 Inflation Index Conventions . . . 187
12.30.19 Swap Index Conventions . . . 187
12.30.20 FX Option Conventions . . . 187
12.30.21 FX Option Time Weighting Scheme . . . 188
12.30.22 Commodity Forward Conventions . . . 188
12.30.23 Commodity Future Conventions . . . 188
12.30.24 Credit Default Swap Conventions . . . 188
12.30.25 Bond Yield Conventions . . . 189
12.31 Historical Return Policy . . . 190
12.32 Authored seed . . . 191
12.33 Return entries . . . 191
12.34 Return types . . . 192
12.35 Displacement . . . 193
12.36 Built-in default configuration . . . 193
12.37 Admissibility caps and floors . . . 194
12.38 Compatibility XML . . . 194
12.39 Operational interpretation . . . 194

13 Collateral Balances **195**
13.1 Current contract role . . . 195

13.2 Top-level structure . . . 195
13.3 Required row fields . . . 196
13.4 Balance amount fields . . . 196
13.5 Relationship to independent amount . . . 196
13.6 Optional row qualifiers . . . 197
13.7 Extended netting-set identity . . . 197
13.8 Operational interpretation . . . 198

14 Counterparty Information **199**
14.1 Current contract role . . . 199
14.2 Authored sources . . . 200
14.2.0.1 Counterparty master. . . . 200
14.2.0.2 Regulatory profile. . . . 200
14.2.0.3 Correlation set. . . . 201
14.3 Top-level resolved structure . . . 201
14.4 Counterparty records . . . 201
14.5 Default and omission rules . . . 204
14.6 Correlation set . . . 204
14.7 Compatibility XML . . . 205
14.8 Operational interpretation . . . 205
14.9 Allowable Values . . . 206

15 Netting Set Information **217**
15.1 Top-level publication structure . . . 218
15.2 Uncollateralised netting sets . . . 218
15.3 Collateralised netting sets . . . 219
15.4 Collateral-balance linkage . . . 220
15.5 Current operational rule . . . 221

16 Market Data **222**
16.1 Zero Rate . . . 223
16.2 Discount Factor . . . 224
16.3 FX Spot Rate . . . 224
16.4 FX Forward Rate . . . 225
16.5 Deposit Rate . . . 226
16.6 FRA Rate . . . 226
16.7 Money Market Futures Price . . . 227
16.8 Overnight Index Futures Price . . . 228
16.9 Swap Rate . . . 228
16.10 Basis Swap Spread . . . 229
16.11 Cross Currency Basis Swap Spread . . . 230
16.12 CDS Spread . . . 230
16.13 CDS Upfront Price . . . 231
16.14 CDS Recovery Rate . . . 232
16.15 CDS Option Implied Volatility . . . 232
16.16 Security Recovery Rate . . . 232
16.17 Hazard Rate (Instantaneous Probability of Default) . . . 233
16.18 FX Option Implied Volatility . . . 233
16.19 Cap Floor Implied Volatility . . . 234

16.20 Swaption Implied Volatility . . . 235
16.21 Equity Spot Price . . . 236
16.22 Equity Forward Price . . . 236
16.23 Equity Dividend Yield . . . 237
16.24 Equity Option Implied Volatility . . . 237
16.25 Equity Option Premium . . . 238
16.26 Commodity Spot Price . . . 238
16.27 Commodity Forward Price . . . 239
16.28 Commodity Option Implied Volatility . . . 239
16.29 Zero Coupon Inflation Swap Rate . . . 241
16.30 Year on Year Inflation Swap Rate . . . 242
16.31 Zero Coupon Inflation Cap Floor Price . . . 242
16.32 Inflation Seasonality Correction Factors . . . 243
16.33 Bond Yield Spreads . . . 243
16.34 Bond Prices . . . 243
16.35 Bond Conversion Factor . . . 244
16.36 Base Correlations . . . 244
16.37 Correlations . . . 244
16.38 Conditional Prepayment Rates . . . 245

17 Fixing History **246**

18 Dividends History **249**

19 Trade Data **251**
19.1 Envelope . . . 252
19.1.1 Netting Set Details . . . 253
19.2 Trade Specific Data . . . 253
19.2.1 Swap . . . 254
19.2.2 Zero Coupon Swap . . . 255
19.2.3 Cap/Floor . . . 256
19.2.4 Forward Rate Agreement . . . 258
19.2.5 Swaption . . . 258
19.2.6 Callable Swap . . . 265
19.2.7 Cash Position . . . 266
19.2.8 FX Forward . . . 266
19.2.9 FX Average Forward . . . 268
19.2.10 FX Swap . . . 270
19.2.11 FX Option . . . 271
19.2.12 FX Asian Option . . . 273
19.2.13 FX Barrier Option . . . 274
19.2.14 FX Digital Barrier Option . . . 277
19.2.15 FX Digital Option . . . 280
19.2.16 FX Double Barrier Option . . . 281
19.2.17 FX Double Touch Option . . . 284
19.2.18 FX European Barrier Option . . . 286
19.2.19 FX KIKO Barrier Option . . . 288
19.2.20 FX Touch Option . . . 291
19.2.21 FX Variance and Volatility Swap . . . 293

19.2.22 Equity Option . . . 295
19.2.23 Equity Futures Option . . . 297
19.2.24 Equity Forward . . . 298
19.2.25 Equity Swap . . . 300
19.2.26 Dividend Swap . . . 302
19.2.27 Equity Asian Option . . . 303
19.2.28 Equity Barrier Option . . . 305
19.2.29 Equity Digital Option . . . 307
19.2.30 Equity Double Barrier Option . . . 308
19.2.31 Equity Double Touch Option . . . 310
19.2.32 Equity European Barrier Option . . . 312
19.2.33 Equity Touch Option . . . 313
19.2.34 Equity Variance Swap . . . 315
19.2.35 Equity Cliquet Option . . . 317
19.2.36 Equity Position . . . 319
19.2.37 Equity Option Position . . . 320
19.2.38 CPI Swap . . . 322
19.2.39 Year on Year Inflation Swap . . . 323
19.2.40 Bond . . . 324
19.2.41 Bond Position . . . 327
19.2.42 Forward Bond . . . 328
19.2.43 Bond Forward / T-Lock / J-Lock (using ref. data) . . . 331
19.2.44 Bond Repo . . . 334
19.2.45 Bond Option . . . 335
19.2.46 Bond Option (using bond reference data) . . . 336
19.2.47 Bond Total Return Swap . . . 338
19.2.48 Convertible Bond . . . 342
19.2.49 Ascot . . . 363
19.2.50 Collateral Bond Obligation CBO . . . 364
19.2.51 Composite Trade . . . 367
19.2.52 Credit Default Swap / Quanto Credit Default Swap . . . 368
19.2.53 Index Credit Default Swap . . . 371
19.2.54 Index Credit Default Swap Option . . . 374
19.2.55 Synthetic CDO . . . 377
19.2.56 Risk Participation Agreement (RPA) . . . 380
19.2.57 Credit Linked Swap . . . 383
19.2.58 Commodity Forward . . . 384
19.2.59 Commodity Swap and Basis Swap . . . 387
19.2.60 Commodity Swaption . . . 387
19.2.61 Commodity Option . . . 389
19.2.62 Commodity Digital Option . . . 391
19.2.63 Commodity Spread Option . . . 391
19.2.64 Commodity Average Price Option . . . 393
19.2.65 Commodity Option Strip . . . 396
19.2.66 Commodity Variance and Volatility Swap . . . 398
19.2.67 Commodity Position . . . 398
19.2.68 Generic Total Return Swap / Contract for Difference (CFD) . . . 400
19.2.69 Equity Outperformance Option . . . 411

19.2.70 Double Digital Option . . . 414
19.2.71 European Option Contingent on a Barrier . . . 416
19.2.72 Autocallable Type 01 . . . 418
19.2.73 Performance Option Type 01 . . . 420
19.2.74 Window Barrier Option . . . 423
19.2.75 Generic Barrier Option . . . 427
19.2.76 Best Entry Option . . . 432
19.2.77 Basket Options . . . 434
19.2.78 Worst Of Basket Swaps . . . 448
19.2.79 Accumulators and Decumulators . . . 456
19.2.80 Target Redemption Forward (TaRF) . . . 469
19.2.81 Knock Out Swap . . . 477
19.2.82 Rainbow Options . . . 477
19.2.83 Exotic Variance and Volatility Derivatives . . . 499
19.2.84 Generic Scripted Products . . . 531
19.2.85 Flexi Swap . . . 531
19.2.86 Balance Guaranteed Swap (BGS) . . . 534
19.3 Trade Components . . . 536
19.3.1 Option Data . . . 537
19.3.2 Premiums . . . 544
19.3.3 Leg Data and Notionals . . . 545
19.3.4 Schedule Data (Rules, Dates and Derived) . . . 549
19.3.5 Fixed Leg Data and Rates . . . 554
19.3.6 Floating Leg Data, Spreads, Gearings, Caps and Floors . . . 555
19.3.7 Leg Data with Amortisation Structures . . . 562
19.3.8 Indexings . . . 563
19.3.9 Cashflow Leg Data . . . 567
19.3.10 CMS Leg Data . . . 568
19.3.11 Constant Maturity Bond Leg Data . . . 570
19.3.12 Digital CMS Leg Data . . . 571
19.3.13 Duration Adjusted CMS Leg Data . . . 573
19.3.14 CMS Spread Leg Data . . . 575
19.3.15 Digital CMS Spread Leg Data . . . 576
19.3.16 Equity Leg Data . . . 578
19.3.17 CPI Leg Data . . . 582
19.3.18 YY Leg Data . . . 586
19.3.19 ZeroCouponFixed Leg Data . . . 588
19.3.20 Commodity Fixed Leg . . . 590
19.3.21 Commodity Fixed Leg Data . . . 590
19.3.22 Commodity Floating Leg . . . 591
19.3.23 Commodity Schedules . . . 591
19.3.24 Commodity Floating Leg Data . . . 592
19.3.25 Equity Margin Leg . . . 599
19.3.26 Equity Margin Leg Data . . . 599
19.3.27 CDS Reference Information . . . 599
19.3.28 Basket Data . . . 600
19.3.29 Underlying . . . 602
19.3.30 StrikeData . . . 606

19.3.31 Barrier Data . . . 607
19.3.32 RangeBound . . . 610
19.3.33 Bond Basket Data for Cashflow CDO . . . 611
19.3.34 CBO Tranches . . . 611
19.3.35 Formula Based Leg Data . . . 612

20 Scripted Trade **613**
20.1 General Structure . . . 613
20.2 Data Types . . . 614
20.2.1 Event . . . 614
20.2.2 Number . . . 616
20.2.3 Index . . . 616
20.2.4 Currency . . . 618
20.2.5 Daycounter . . . 619
20.3 Compact Trade XML . . . 619
20.4 A comment on the Payment Currency in Scripted Trades . . . 621
20.5 Payoff Scripting Language . . . 622

1 Introduction

The core engine of Vannarho aims to provide a transparent platform for pricing and risk analysis that serves as

- a benchmarking, validation, training, and teaching reference,
- an extensible foundation for tailored risk solutions.

VRE is an application that provides

- a Monte Carlo simulation framework for contemporary risk analytics and value adjustments
- simple but increasingly structured interfaces for trade data, market data and system configuration
- simple launchers and result visualisation in Jupyter
- unit tests and various examples.

VRE is open source software, provided under the Modified BSD License. It is based on QuantLib, the open source library for quantitative finance [6] and Open Source Risk Engine [4].

Audience

VRE aims to reach quantitative risk management practitioners (be it in financial institutions, audit firms, consulting companies or regulatory bodies) who are looking for accessible software solutions, and quant developers in charge of the implementation of pricing and risk methods similar to those in VRE. Moreover, the project aims at reaching academics and students who would like to teach or learn quantitative risk management using a freely available, contemporary risk application. And in the meantime, as VRE is used in risk services at industrial scale since 2018 with over 150 users, VRE is aimed at firms that consider the replacement of third party licensed software.

1.1 Scope

VRE currently provides portfolio pricing, cash flow generation, market risk analysis and a range of contemporary derivative portfolio analytics. The latter are based on a Monte Carlo simulation framework which yields the evolution of various exposure measures:

- EE aka EPE (Expected Exposure or Expected Positive Exposure)
- ENE (Expected Negative Exposure, i.e. the counterparty's perspective)
- 'Basel' exposure measures relevant for regulatory capital charges under internal model methods
- PFE (Potential Future Exposure at some user defined quantile)

and derivative value adjustments (xVA)

- CVA (Credit Value Adjustment)
- DVA (Debit Value Adjustment)

- FVA (Funding Value Adjustment)
- COLVA (Collateral Value Adjustment)
- MVA (Margin Value Adjustment)

for portfolios with netting, variation and initial margin agreements.

The market risk framework provides

- sensitivity analysis, also in the "par" domain
- stress testing, also in the "par" domain
- several parametric VaR versions (Delta VaR, Delta-Gamma Normal VaR, Delta-Gamma VaR with Cornish-Fisher expansion and Saddlepoint method)
- historical simulation VaR
- P&L and P&L explain

across all asset classes and products.

In the VRE v0.11.2 release line, this is not just a historical feature list. It describes the current engine scope:

- broad multi-asset pricing and risk coverage across IR, FX, EQ, COM, INF, CR, bond and hybrid products, including structured and scripted trades;
- portfolio valuation, cashflow generation, sensitivities, stress, historical and parametric market-risk workflows, P&L and P&L explain;
- xVA and exposure analytics spanning uncollateralised and collateralised portfolios, close-out modelling, American Monte Carlo and dynamic initial margin workflows;
- regulatory capital analytics for Basel III "end game" style runs, including CCR (**SA-CCR**, **IMM**), credit risk (**SA-CR**, **IRB**), CVA (**BA-CVA** reduced and full, **SA-CVA** with both SA-CCR and IMM sensitivity support), and market risk (**SA-MR**, **IMA**);
- **RWA** and scope-aware regulatory reporting flows for jurisdictions and reporting scopes such as **EU**, **PRA**, **US**, **SG** and **AU**, with the maintained examples and reporting layers showing how the same deterministic engine is configured differently per scope;
- banking-book Pillar 2 execution for **IRRBB**, **CSRBB**, governed stress-testing compute support, and bounded capital-floor companion outputs;
- compliance, evidence and operational tooling around the engine itself, including `vre_compliance`, `config_target`, scope/reporting publication, and `vre_config_doctor`.

The product and analytic breadth now sits in a clearer repository and runtime structure:

- `vre` remains the deterministic core engine and source of engine-owned semantics;
- `vre-raas` is **Results as a Service**: the companion Google Cloud Platform service that is designed to transform credit-risk and market-risk middle- and back-office operations

through AI-enabled workflows, while keeping `vre` as the deterministic core risk engine for pricing, exposure, capital and reporting computation;

- `vre-raas-contracts` carries the shared versioned boundary between the two repositories, so that engine and control-plane releases can evolve independently while remaining pinned to an explicit compatibility bundle.

The current VRE scope is clearer when read through three complementary views:

1. a regulatory Pillar view (what families and methods are in scope),
2. a jurisdiction/regime view (where true-flow support is currently claimed),
3. a product view grouped by asset class (what instrument families feed those analytics).

Pillar	Scope family / method set	Scope status	Maintained execution surface
P1	Counterparty Credit Risk (`SA-CCR`, `IMM`)	primary_scope	Maintained entrypoints under `examples/regulatory/pillar_1/credit/ccr`, including netting-set exposure and capital-ready outputs.
P1	Standardised Credit Risk (`SA-CR`)	partial_scope	Direct-credit `SA-CR` true-flow runs are maintained; first-class full banking-book `SA-CR` runtime promotion remains an active closure item.
P1	IRB Credit Risk	partial_scope	Direct-credit `FIRB`/`AIRB` true-flow runs and disclosure feeds are maintained; full bank-wide IRB runtime promotion remains active.
P1	Direct Credit Risk	bounded_in_scope	Starter-scope bond and credit-position capital flows under `examples/regulatory/pillar_1/credit/direct_credit`.
P1	Credit Risk Mitigation	partial_scope	CRM haircut and substitution application is supported when institution-validated CRM facts are supplied at runtime.
P1	CVA Risk (`SA-CVA`, `BA-CVA`, APRA overlay)	primary_scope	Maintained `SA-CVA`/`BA-CVA` runs with APRA overlay support through governed compatibility surfaces.
P1	Market Risk (`SA-MR`, `SimplifiedMR`, `IMA`, APRA overlays)	primary_scope	`SA-MR`, `SimplifiedMR`, and `IMA` runs are maintained; overlay/doc-contract convergence is still tracked in closure backlog items.
P1	Output Floor / aggregation inputs	partial_scope	VRE publishes bounded component outputs (for example `basel__capital_floor_basis`) for downstream capital-floor aggregation.
P2	IRRBB	bounded_in_scope	Common measurement plus AU APS 117 overlay/raw-component true-flow, with maintained EU/GB/SG jurisdiction profiles.
P2	CSRBB	bounded_in_scope	Common measurement true-flow with AU/EU/GB/SG maintained regime-profile runs.
P2	Stress Testing Compute Support	bounded_in_scope	Governed scenario/revaluation compute support with evidence sidecars; management actions remain bank-owned.
P3	Quantitative disclosure feeds	derived_support	Row/cell-level disclosure feed mapping is derived from governed Pillar 1 and Pillar 2 outputs.
P3	Qualitative disclosure support	bounded_in_scope	Methodology and evidence-backed supporting text inputs are produced; signed narrative ownership remains bank-owned.
P3	Disclosure reconciliation support	partial_scope	Reconciliation artifacts and lineage are generated; financial-statement and group-signoff ownership remains outside VRE.

Table 1: Current scope by regulatory Pillar and method family.

Within the Pillar view, jurisdiction adoption maturity is an important qualifier for how quickly local overlays can be operationalised. A compact snapshot (from `product/ref/reg/basel_3_1_adoption_status.md`) is:

- implemented or largely in force: EU, SG, HK;

- finalised direction, effective date still ahead: AU, GB;
- proposal-stage and most policy-fluid: US.

This is intentionally a current-state snapshot, not an exhaustive policy survey. The jurisdiction/regime table below then shows where maintained VRE true-flow support is currently claimed.

Jurisdiction / regime profile	Current maintained coverage	Adoption maturity and VRE status
AU (`AU_BANKS_BASEL31`, APRA)	P1 overlays (`APRA CVA`, `APRA MR`), P2 IRRBB overlay + APS 117 raw components, CSRBB, stress-testing compute support, APRA prudential-return packs (`ARS116`/`ARS180`).	Basel 3.1 direction is finalised (effective date ahead); supported regime perimeter with deepest local-overlay coverage.
GB (`GB_BANKS_BASEL31`, PRA)	P2 IRRBB/CSRBB true-flow profiles, plus PRA slices in scoped-client disclosure packs.	Finalised rulebook with effective dates ahead; supported regime perimeter.
EU (`EU_BANKS_CRR3_CRD6`)	P2 IRRBB/CSRBB guideline true-flow profiles, plus EU disclosure slices and cross-entity pack coverage.	Broadly implemented/in-force with live FRTB timing overlays; supported regime perimeter.
SG (`SG_BANKS_NOTICE637`, MAS)	P2 IRRBB/CSRBB true-flow profiles and MAS scoped disclosure slices in cross-border packs.	Implemented with output-floor phase-in still progressing; supported regime perimeter.
US (`US_BANKS_BASEL_-ENDGAME`)	Jurisdiction slices exist in scoped-client/disclosure packs; P2 IRRBB/CSRBB true-flow is not in the supported completion perimeter.	Proposal-stage and most policy-fluid jurisdiction; VRE does not claim full US support in the current v0.11.2 release line.
HK (`HK_BANKS_BCR_BDR`)	Jurisdiction and authority are registered in compliance objects; no maintained P2 true-flow claim is made yet.	Local regime is implemented; VRE runtime localisation remains registry-seeded / WIP.

Table 2: Jurisdiction and regime-profile coverage view.

Finally, the product view is grouped by asset class so the reader can map instrument families to analytics families without relying on one very wide cross-check matrix.

Asset-class group	Representative product families in VRE	Typical analytic / regulatory linkage
Rates and inflation	IRS/OIS/XCCY swaps, caps/floors, swaptions/callables, CMS, CPI and YoY inflation swaps/caps	Pricing, sensitivities, stress, xVA, CCR/CVA feeds, market-risk sensitivities, and reporting decomposition.
FX	Forwards, swaps, vanilla/asian options, barrier and touch exotics, variance products	Pricing + xVA + CCR/CVA + market-risk scope with jurisdictional reporting profiles.
Equity	Forwards, swaps, vanilla/asian options, futures options, barriers and other exotics, variance and cliquet structures	Pricing/risk and xVA coverage; SA-CVA and market-risk sensitivities where configured.
Commodity	Forwards, swaps, swaptions, vanilla/asian/digital options, APO and strip structures, variance products	Pricing/risk and scenario coverage with market-risk and xVA linkage in supported run bundles.
Credit and bonds	Bonds/loans, repos, bond options, convertibles, CDS/index CDS/options, credit-linked swaps, tranches/synthetic CDO	Direct-credit `SA-CR`/IRB starter scope, CRM application, CCR/CVA linkage, and bounded output-floor component outputs.
Hybrid / portfolio composites	Composite trades, TRS/CFD, ASCOTs, Flexi/BGS, scripted trades (single and multi-asset)	Cross-asset valuation, stress, xVA, and disclosure-feed support where governed mappings are configured.

Table 3: Product scope grouped by asset class and linked analytics surfaces.

The simulation models applied in VRE's risk factor evolution implement the models discussed in detail in *Modern Derivatives Pricing and Credit Exposure Analysis* [15]: The IR/FX/INF/EQ risk factor evolution is based on a cross currency model consisting of an arbitrage free combination of Linear Gauss Markov models for all interest rates and

lognormal processes for FX rates and EQ prices, Dodgson-Kainth (or Jarrow-Yildirim) models for inflation. The model components are calibrated to cross currency discounting and forward curves, Swaptions, FX Options, EQ Options and CPI caps/floors. With the 8th release, Commodity simulation has been added, as well as the foundation for a multi-factor Hull-White based IR/FX/COM simulation model.

The current release line, VRE v0.11.2, still retains the long-established XML compatibility surface for existing portfolios and engine readers, but the preferred authoring direction is now ADR/native:

- `TOML` for authored runtime launch bundles and net-new engine configuration,
- `YAML` for curated policy and mapping documents,
- `JSON` for resolved bundles, machine contracts, manifests and evidence,
- retained `XML` for compatibility readers and selected payload families that are intentionally still XML.

This user guide therefore has two jobs. First, it explains the current maintained execution surface of `vre` as it exists today, including scope/reporting flows, current examples, compliance evidence, Config Doctor and the shared boundary with `vre-raas`. Second, it preserves the XML-era reference material further below so that no engineering detail is lost during the migration.

`vre_config_doctor` is now part of the normal operating picture. It combines typed deterministic validation with machine-learning-assisted pattern recognition and verification-gated LLM guidance to help repair existing configuration packs and create new ones. Its advice does not replace the deterministic engine contract: trust still comes from the resolved runtime bundle and the engine's own validation rules.

1.2 VRE in Python

VRE is written in C++23 and comes with a command line executable `vre.exe` that supports batch processes. We provide first-class Python bindings via a PyBind11 module shipped as prebuilt *wheels* for each release. Install the most recent VRE Python module by calling

```
pip install vannarho-risk-engine
```

See section 9.16 on how to use VREPython.

1.3 Roadmap

VRE grows with community contributions and the demand of clients who utilise VRE to replace existing risk applications with the help of the Vannarho Services team.

The forward roadmap is centred on extending the same engine architecture rather than creating a separate analytics stack:

- Further jurisdictional variants of Basel 3.1 analytics, including additional reporting and capital variations; the US remains the most policy-fluid Basel end-game jurisdiction and has the most limited support;

- continued widening of scope-based reporting and RWA aggregation so that jurisdiction-specific runs remain deterministic and reproducible from the same authored input surface;
- native reverse/conditional-expectation pipelines on **CUDA** for fast sensitivities, especially in SA-CVA, SA-MR and related large-factor runs;
- fast market-risk scenario generation and scenario consumption on **CUDA**, complementing the existing host and device execution back-ends;
- further performance optimisation for large portfolios, large simulation counts and higher-throughput RaaS deployment patterns.

Performance: In the VRE v0.11.2 release line, production **AAD** pipelines couple with LLVM/Enzyme so that the CG random-variable algebra can be JIT-compiled into deterministic execution plans. Conditional expectations are solved via damped least-squares projections; the resulting Cholesky factors are cached and reused during the backward sweep so that adjoint sensitivities inherit the same numerical conditioning without recomputation. On the CPU, the CE solver now ships with scalar, NEON, and AVX2 back-ends that are selected at run time via policy flags, delivering sizeable speed-ups on fast sensitivities (e.g. Sensitivities-Based Methods for SA-CVA and SA-MR) while sharing the numerical scaffolding with the interpreter path. The host forward sweep now executes level-by-level across a thread pool (`immCgForwardThreads`), and cube generation can be sharded safely across workers (`immCgCubeThreads`) so that large SA-CVA portfolios no longer bottleneck on serial write-outs. Beyond the host, VRE supports **Metal** (macOS) and **CUDA** (Linux/Windows) execution back-ends: the same plan abstraction feeds device kernels, guarded math intrinsics keep the implementations portable, and telemetry mirrors the CPU path so that performance gains can be audited easily. The updated `examples/performance`, `examples/XvARisk` and `examples/regulatory/pillar_1/market_risk` workflows illustrate typical acceleration factors and document the configuration switches. Extended SIMD coverage and additional device enhancements remain active work streams and will continue to land in future releases.

Legacy key names `xvaCgForwardThreads` and `xvaCgCubeThreads` remain accepted as compatibility aliases.

VRE Python: The Python wrapper is now treated as a first-class operating surface for examples, notebooks, automation and service integration around the same engine contracts.

1.4 Further Resources

- Main site: http://www.vannarho.com
- Source code and releases: https://github.com/Vannarho/vre.git
- VRE Product Catalogue [1]
- VRE Methodology [2]
- VRE Compliance Guide in `docs/compliance/compliance.tex` and machine evidence packs under `vre_compliance/`
- Shared VRE / VRE-RaaS bundle repository: `vre-raas-contracts`

Organisation of this document

This document focuses on instructions how to use VRE to cover basic workflows from individual deal analysis to portfolio processing. After an overview over the core VRE data flow in section 2, and compliance operation guidance in section 5, and installation instructions in section 8 we start in section 9 with a series of examples that illustrate how to launch VRE using its command line application, and we discuss typical results and reports.

The description of products, pricing and trade representation in VRE XML has been carved out and moved to [1].

And finally, a summary of methodologies applied in VRE can be found in [2].

2 VRE Data Flow

The core processing steps followed in VRE to produce risk analytics results are sketched in Figure 1. All VRE calculations and outputs are generated in three fundamental process steps as indicated in the three boxes in the upper part of the figure. In each of these steps appropriate data (described below) is loaded and results are generated, either in the form of a human readable report, or in an intermediate step as pure data files (e.g. NPV data, exposure data).

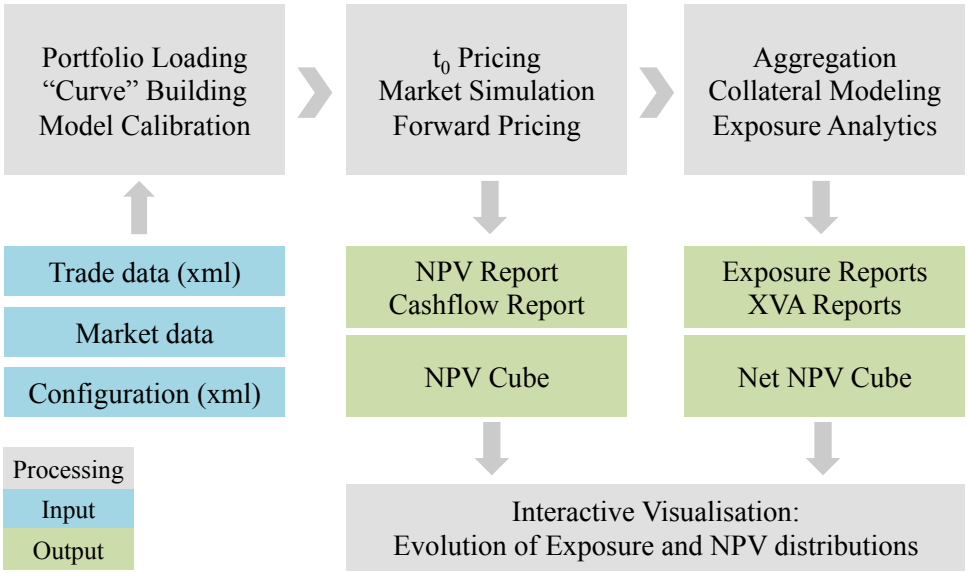

Figure 1: Sketch of the VRE process, inputs and outputs.

The process sketched in Figure 1 is still the right computational picture for VRE, but the way runs are prepared has changed. In VRE v0.10.0 two launch layers coexist:

- a current ADR/native surface built around authored `TOML` bundles, curated `YAML` metadata, resolved `JSON` contracts, published evidence, and `input_native` package roots,
- a legacy-compatible XML surface that remains supported for existing engine readers, example continuity and retained payload families.

The current execution surface is summarised in sections 3, 4 and 10. The detailed XML family reference remains in section 12 and the following legacy-compatible chapters. Portfolio payloads are still predominantly XML and are explained in detail in [1] and 15. Market data is still commonly provided as simple row-oriented files as explained in section 16. The first processing step (upper left box) then comprises

- loading the portfolio to be analysed,
- building any yield curves or other 'term structures' needed for pricing,
- calibration of pricing and simulation models.

The second processing step (upper middle box) is then

- portfolio valuation, cash flow generation,
- going forward - conventional risk analysis such as sensitivity analysis and stress testing, standard-rule capital calculations such as SA-CCR, etc,
- and in particular, more time-consuming, the market simulation and portfolio valuation through time under Monte Carlo scenarios.

This process step produces several reports (NPV, cashflows etc) and in particular an **NPV cube**, i.e. NPVs per trade, scenario and future evaluation date. The cube is written to a file in both condensed binary and human-readable text format.
The third processing step (upper right box) performs more 'sophisticated' risk analysis by post-processing the NPV cube data:

- aggregating over trades per netting set,
- applying collateral rules to compute simulated variation margin as well as simulated (dynamic) initial margin posting,
- computing various XVAs including CVA, DVA, FVA, MVA for all netting sets, with and without taking collateral (variation and initial margin) into account, on demand with allocation to the trade level.

The outputs of this process step are XVA reports and the 'net' NPV cube, i.e. after aggregation, netting and collateral.
The example section 9 demonstrates for representative product types how the described processing steps can be combined in a simple batch process which produces the mentioned reports, output files and exposure evolution graphs in one 'go'.

3 Current Execution Surface

VRE v0.11.2 has a current maintained execution surface and a bounded compatibility surface.

The ADR/native work across configuration, scope/reporting and documentation has made the current execution model more explicit. In practice, the current surface is organised around:

- human-authored launch bundles in `TOML`,
- curated policy and registry material in a controlled `YAML` subset,
- resolved machine contracts, manifests and evidence in `JSON`,

- runtime-ready package roots under `input_native`,
- archival or compatibility XML readers where specific engine seams still require them; they are outside the maintained v3 authoring surface.

The current execution model is therefore better described as *authored contracts → resolved runtime bundle → engine execution → evidence and reports* than as a single master wrapper-file hand-off.

Authoring Layers

Layer	Format	Operational role
Authored run bundle	`TOML`	Human-authored launch intent, source-bundle references, runtime switches and maintained example entry points
Curated specification / registry layer	`YAML`	Rulebooks, mappings, scope/reporting metadata, curated registries and similar slow-moving controlled specifications
Resolved runtime / publication layer	`JSON`	Resolved bundles, machine contracts, manifests, run evidence, job evidence and output sidecars
Runtime-ready package root	directories under `input_native`	Deterministic runtime package assembled from the authored and published layers
Compatibility reference layer	retained `XML`	Archival compatibility payloads and explicit compatibility bridges; not part of the maintained v3 launch surface

Table 4: Current execution layers in VRE v0.11.2.

Scope-Centric Runtime Direction

The scope/reporting program introduces an execution model in which run intent is no longer defined only by a legal-entity-specific wrapper. Instead, the run contract increasingly captures client, reporting scope, rulebook, as-of date and configuration-set identity, with explicit lineage and method-resolution artifacts generated alongside the run.

For users this means that the maintained execution surface increasingly exposes:

- run-bundle authoring separated from runtime publication,
- explicit scope, rulebook and reporting metadata,
- deterministic run evidence and job evidence,
- report outputs that can be traced back to resolved runtime inputs rather than only to an XML wrapper.

This is particularly relevant for current maintained regulatory examples:

- `examples/regulatory/pillar_1/*`,
- `examples/regulatory/pillar_2/*`,
- `examples/regulatory/pillar_3/*`,
- `examples/regulatory/cross_pillar/*`.

Compatibility Reference Boundary

Later XML family chapters remain archival compatibility reference, not maintained execution guidance. They are kept so existing payload boundaries and historical regression surfaces

remain inspectable while maintained v3 launch surfaces continue to use TOML, YAML, JSON, and Parquet contracts.

4 Current Outputs, Reporting and Evidence

The output surface in current VRE runs is broader than the traditional collection of CSV reports and diagnostic logs. Current runs publish not only results, but also the runtime evidence needed to explain how those results were produced.

Output Classes

The user should distinguish four output classes:

- business reports such as NPV, cashflow, sensitivity, stress, exposure and capital reports,
- runtime machine outputs such as resolved bundle manifests and structured diagnostics,
- evidence artifacts such as `run_evidence.json` and `job_evidence.json`,
- report sidecars and metadata files that capture lineage, publication and compatibility context.

This distinction becomes important as the engine moves further into scope-native reporting and regulator-facing output abstractions.

Scope Reporting

The scope/reporting program preserves the established report estate while moving the runtime onto scope-native contracts. For users, this means:

- existing report outputs remain important and are preserved during migration,
- current reporting flows increasingly expect explicit scope, rulebook and method-resolution context,
- runtime lineage is no longer an optional side concern but part of the output contract for regulatory runs.

For regulatory and cross-entity flows, the output set should therefore be read as:

1. calculation outputs,
2. supporting lineage and method evidence,
3. compliance-pack evidence and validation status.

RWA Publication Direction

The RWA roadmap now matters to the user guide because the output boundaries are no longer uniform across the major regulatory domains.

- Market Risk currently publishes capital and RWA outputs in the maintained examples and report flows.

- Credit Risk / CCR currently publishes exposure primitives (for example EAD, RC, PFE, EEPE) and related disclosure outputs; explicit CCR RWA publication is a distinct reporting layer being formalised.
- CVA Risk currently publishes capital outputs and disclosure routes; explicit CVA RWA publication likewise sits in the reporting layer and must remain provenance-tagged and reconcilable to the underlying capital primitives.

The design rule is that the exposure engines produce exposure or capital primitives, and the reporting layer publishes RWA outputs with explicit provenance rather than burying risk-weight logic inside the exposure calculation itself.

Compliance Operating Model

The current compliance model in VRE is intentionally explicit:

1. execute the run with captured configuration and output artifacts,
2. generate or refresh the relevant compliance-pack evidence,
3. validate the pack status and trace completeness,
4. consume only validated pack status in release and governance decisions.

The machine-readable authority for this layer lives under `vre_compliance/`. The user-guide role is to explain how current users should read and operate with those artifacts, not to duplicate the full compliance-pack specification.

New Regulatory Output Families

Representative regulated output families added in the current release line include:

- banking-book Pillar 2 facts such as `irrbb__base_measurement_facts`, `irrbb__outlier_facts`, `csrbb__base_measurement_facts`, and `stress_testing__scenario_result_facts`;
- AU APS 117 raw-component and capital outputs for prospective charge, embedded loss / gain, optionality charge, and capital summary views;
- direct-credit true-flow reporting outputs for SA-CR and IRB starter runs;
- IMA validation and publication outputs such as `marketrisk__ima__validation_metrics` and `marketrisk__ima__backtesting_exception_register`;
- bounded capital-floor outputs such as `basel__capital_floor_basis`.

5 Regulatory Compliance and Auditability

Transition note: This section is intentionally concise. For active v3 architecture authority, use `docs/compliance/traceability/`. For post-Pillar-2 and post-IMA convergence status, use `docs/compliance/traceability/post_pillar2_v3_convergence.md`. For release-grade compliance claims, use validated pack artifacts under `vre_compliance/packs/`. Detailed user-facing coverage for the delivered banking-book

Pillar 2 surface and maintained regulatory examples now lives in the main user-guide chapters.

VRE publishes compliance evidence through a dedicated root-level framework:

- `vre_compliance/README.md`
- `vre_compliance/IMPLEMENTATION_PLAN.md`
- method packs under `vre_compliance/packs/`
- compliance-first development guidance under `docs/compliance/traceability/`
- curated and generated compliance objects under `vre_compliance/v3/`

The compliance packs are the authoritative source for release assertions. They are designed for deterministic, machine-verifiable audit trails and include requirement traces, report contracts, test links, run evidence, and disclosure mappings.

The current coexistence rule is:

1. `vre_compliance/packs/` governs release and supervisory claims.
2. `vre_compliance/v3/` governs compliance-first machine-readable architecture, registry generation, and new migration waves.
3. Maintained regulatory examples and committed expected outputs that feed packs or retrieval remain controlled evidence surfaces, not casual demo fixtures.

For users running regulatory analytics (for example SimplifiedMR, SA-MR, IMA, SA-CCR, IMM, SA-CVA, BA-CVA, IRRBB, CSRBB, and stress-testing compute support), the expected operational model is:

1. execute analytics with explicit run configuration and output capture,
2. generate/update compliance pack evidence from those outputs,
3. validate packs through schema and consistency checks,
4. consume only validated pack status in governance and release decisions.

Method naming in this guide is standardized to:

- Market Risk: `SimplifiedMR`, `SA-MR`, and `IMA`
- CCR: `SA-CCR` and `IMM`
- CVA: `SA-CVA` and `BA-CVA`
- Banking-book Pillar 2: `IRRBB`, `CSRBB`, and `StressTestingComputeSupport`

Legacy market-risk labels are retired for active runtime and operational documentation surfaces. They are retained only in archived evidence trails and direct regulatory quotations where required.

Detailed architecture, governance rules, and migration design are documented in the standalone compliance guide:

- `docs/compliance/compliance.tex`

- `docs/compliance/traceability/vre_compliance_compliance_first_-operating_model.md`
- `docs/compliance/traceability/vre_compliance_runtime_registry_-evidence_boundary.md`
- `docs/compliance/traceability/post_pillar2_v3_convergence.md`

Figure 2: Jurisdiction-first compliance operating model from regime profile through execution and evidence.

6 Shared Boundary With VRE-RaaS

Current VRE builds no longer treat the VRE / VRE-RaaS boundary as implicit repo knowledge. `vre-raas` means **Results as a Service**: the companion service layer that uses `vre` as its deterministic core risk engine while wrapping that engine in headless and UI-driven workflows for credit-risk and market-risk operations on Google Cloud Platform. The shared machine-readable boundary is published in the separate repository `vre-raas-contracts`, and VRE consumes a pinned bundle from that repository as a versioned input dependency.

Operational Meaning

For the VRE user, the important operational rules are:

- the shared boundary is read-only at runtime from the point of view of `vre`,
- VRE validates the pinned bundle and consumes it as an input dependency,
- VRE does not write back into the shared-boundary repository during runtime execution,
- shared-boundary policy changes are managed through maintained source material and bundle publication, not by ad hoc edits inside runs.

The boundary matters most where VRE and VRE-RaaS need a stable published contract, for example:

- analytic intent and job-contract metadata,
- output-contract and output-projection registries,
- parity and migration-seam declarations,
- Config Doctor and compliance registry publication.

Why It Appears In The User Guide

The shared-boundary repository is not a maintainer-only curiosity. It affects how current VRE users should understand:

- versioned contract adoption,

- cross-repo compatibility,
- Config Doctor retrieval authority,
- compliance and output registry validation.

The user guide therefore records the existence and purpose of `vre-raas-contracts`, while the detailed ownership, release and maintainer workflows remain documented in that repository itself.

7 Config Doctor

Config Doctor is now a first-class component of `vre` rather than an incidental helper script. It is the configuration-intelligence and verification layer for current authored bundles, runtime packs and migration-equivalence checks.

Purpose

Config Doctor is designed to:

- validate packs and run bundles,
- explain dependency and publication failures,
- construct typed config graphs and evidence bundles,
- support safe repair planning and verification,
- support configuration creation and migration-equivalence checks over time.

In current operating terms, Config Doctor combines deterministic contract validation with ML-assisted pattern recognition and verification-gated LLM guidance. The ML/LLM layers are there to advise on repair and creation; they do not replace the deterministic engine contract.

Its package home is `vre_config_doctor/`, while compatibility command-line surfaces and generated artifacts remain under `tools/config_validator/` where path stability is needed.

Authority Model

Config Doctor is intentionally deterministic-first. Its authority order is:

1. runtime truth from VRE code and published configuration contracts,
2. published migration and shared-boundary artifacts from `vre` and `vre-raas-contracts`,
3. typed knowledge derived from documentation and reference material,
4. advisory ML and LLM layers that remain verification-gated.

This means that Config Doctor may explain or propose, but it does not override the deterministic contract surface.

Current Practical Use

In day-to-day usage Config Doctor is relevant wherever users need confidence that a run bundle or config pack is both syntactically valid and semantically coherent. Typical use cases are:

- validating maintained example packs and authored launch bundles,
- proposing repairs or new pack skeletons from typed current surfaces,
- comparing migrated native packs against XML-era equivalents,
- checking published input-contract layers and manifests,
- surfacing shared-boundary or documentation drift before a run is trusted.

The known-good corpus is now derived from current example runners rather than from stale legacy example trees. That is important because it ties verification to the example estate that is still actively exercised.

8 Getting and Building VRE

VRE's source code is hosted at `https://github.com/Vannarho/vre.git` (this is offered to clients on a private basis). VRE's examples and pre-built wheel binaries are available at `https://github.com/Vannarho/examples`.

You can get VRE sources in two ways, either by downloading a release bundle as described in section 8.1 or by checking out the source code from the github repository as described in section 8.2. In both cases, the user usually needs to build (compile) from sources in a platform dependent way. To avoid that build process, one can either get the pre-built VRE Python module as mentioned in section 1.2, or try the executables built on github automatically for a few platforms for testing purposes, see the "Actions" menu.

Operationally, a current VRE checkout also contains companion runtime and governance surfaces that are now part of normal use:

- `vre_compliance/` for compliance packs and release evidence,
- `vre_config_doctor/` for deterministic configuration validation and repair planning,
- vendored shared-boundary contracts consumed from `vre-raas-contracts`.

8.1 VRE Releases

VRE releases are snapshots of the `vre` codebase, regularly provided in the form of source code archives.

Unpacking the downloaded release archive creates a source tree rooted at `vre` or a release-tagged source directory, with the following files resp. subdirectories

1. `vre_app/`
2. `cmake/`
3. `CMakeLists.txt`

4. `CMakePresets.json`
5. `cmake/`
6. `Docker/`
7. `docs/`
8. `examples/`
9. `install/`
10. `vre_analytics/`
11. `vre_data/`
12. `vre_python/`
13. `vre_test/`
14. `vre_quantlib_ext/`
15. `vre_quantlib/`
16. `third_party_libs/`
17. `tools/`
18. `Tutorials/`
19. `xsd/`

Note:

- Each release also contains the QuantLib source version that VRE depends on; this is usually the latest QuantLib release that precedes the VRE release including a small number of patches.
- Since 2025, Python bindings are provided via the ‘vre_python/‘ module (PyBind11) with prebuilt wheels available at https://github.com/Vannarho/examples.

8.2 Access via Git

To access the evolving code base on GitHub or specific versions conveniently, one should clone VRE’s git repository.

1. Install Git on your machine following instructions at [8]
2. Fetch VRE from github by running the following:

   ```
   % git clone https://github.com/Vannarho/vre.git vre
   ```

   This will create a folder ’vre’ in your current directory that contains the codebase.
3. Initially, the `vre_quantlib` subdirectory under `vre` is empty as it is a submodule pointing to the Vannarho QuantLib repository. To pull it down locally, use the following commands:

   ```
   % cd vre
   % git submodule init
   % git submodule update
   ```

Note that one can also run

```
% git clone --recurse-submodules https://github.com/Vannarho/vre.git vre
```

in step 2, which also performs step 3 for `vre_quantlib`. Read-only libraries under `third_party_libs/` are vendored directly in the repository and do not require submodule initialization.

The above fetches VRE from the master branch, if you want to fetch a specific release then right after `% cd vre` do `% git checkout tags/release_name`, e.g. `% git checkout tags/v1.8.12.0`.

8.3 Prerequisites for Building VRE

To build VRE from sources one needs to install the following collection of tools

- C++ compiler
- Boost libraries
- CMake
- Ninja (optional)
- zlib (optional)
- Eigen
- Python
- PyBind11 toolchain (for building the Python module from source) and scikit-build-core
- LLVM/Enzyme (optional, for building VRE with compiled CPU kernels)

C++ Compiler

QuantLib and VRE are written in C++, so we need a C++ compiler – this comes on Windows with Visual Studio, it is gcc on Linux, clang on macOS.

On Windows, download and install Visual Studio Community Edition (Version 2019 or later, 2022 is recommended), and during the installation, make sure to

- install the Visual C++ support under the Programming Languages features (disabled by default)
- install the feature 'C++ CMake Tools for Windows'

CMake

The VRE build requires CMake (https://cmake.org), to be installed on Linux and macOS using the usual package managers (apt, homebrew). Visual Studio has CMake support in VS2019 or later, and you can install the feature 'C++ CMake Tools for Windows' with VS instead of installing CMake as standalone program on Windows.

Boost

QuantLib and VRE depend on the Boost C++ libraries, to be installed before kicking off the build process. At least Boost version 1.72.0 is required.

On Linux and macOS, one can install boost conveniently using apt or homebrew

```
sudo apt install libboost-all-dev
brew install boost
```

Otherwise, when building boost following instructions at [9] and when installing boost in a non-standard path, take note of the boost include and library directories on your system. These paths need to be passed to the CMake configure step, see below.

Compiler / Boost Versions

Ensure consistency of compiler and boost version: The following table 5 reflects the compiler / boost version combinations that the users/developers at Quaternion/Acadia/LSEG can confirm as working combinations with the current VRE 0.10.0 release line.

Compiler	Boost	VRE
AppleClang version 16.0.0	1.87.0	0.10.0
gcc 11.4.0	1.74.0	0.10.0
VS2022	1.86.0	0.10.0

Table 5: Supported compiler and boost versions for VRE 0.10.0, every boost version between 1.72 and 1.89 should work.

LLVM / Enzyme (optional)

The compiled CPU path (CG plan JIT kernels) is guarded by the CMake option `VRE_ENABLE_CPU_KERNELS`. Leaving this option at its default `OFF` value keeps the legacy interpreter and requires no additional dependencies. Enabling it pulls in LLVM/Clang and the Enzyme automatic differentiation plugin:

- LLVM 21 is supported. Provide the installation prefix via `VRE_LLVM_ROOT` (pointing to the directory that contains `lib/cmake/llvm/LLVMConfig.cmake`) or arrange for CMake to find `LLVM_DIR` on its own.
- Enzyme must be built against the same LLVM version. Set `VRE_ENZYME_ROOT` to the prefix that contains `lib/LLVMEnzyme-${LLVM_VERSION_MAJOR}.(dylib|so)`. The build copies the plugin path into the generated configuration so the runtime can load it on demand.
- On Darwin/arm64 the top-level presets define `VRE_COMPILED_CPU_NEON` so the runtime prefers NEON kernels. On x86_64 the SIMD flavour is selected dynamically (AVX2/AVX512) based on host capabilities.

The shipped CMake presets (e.g. `macos-clang-ninja-release-metal`) already toggle `VRE_ENABLE_CPU_KERNELS=ON`; they will emit a descriptive error if LLVM/Enzyme cannot be located so it is clear why compiled kernels are unavailable.

Install Scripts

The `install/` folder contains helper scripts to stage toolchains and dependencies per platform. They align with the shipped presets:

- macOS: e.g. `install/mac/inst_mac.sh` installs core build deps; `install/mac/inst_mac.sh` and `install/mac/inst_mac_enzyme.sh` Metal presets (`macos-*-metal`) assume Xcode/SDKs are present.
- Linux: e.g. (`install/linux/risk-debian-12.sh` and `install/linux/risk-ubuntu`); use CUDA presets after installing the matching NVIDIA toolkit/driver. Wheel scripts in `vre_python/` stage deps inside cibuildwheel containers.
- Windows: PowerShell helpers in `install/` document MSVC/CUDA setup; Visual Studio reads `CMakePresets.json` to pick the appropriate generator/toolset.
- CUDA: Example custom install script `install/cuda/cuda.sh` installs CUDA toolkit + dev libs (e.g. cuBLAS/cuSOLVER) if needed. The CUDA presets assume the matching NVIDIA toolkit/driver is installed on the system; see https://developer.nvidia.com/cuda-downloads for details.

Ninja

The installation of Ninja is optional.

When running the build process in the command line on a Linux or macOS system, the default "generator" is `GNU make` which is expected to be available on the system. As an alternative and for build speed we recommend `Ninja` (https://ninja-build.org). Moreover, Ninja is also available on Windows, so that we can run command line builds with same commands on Windows, Linux and macOS.

However, Windows users can rely on Visual Studio's collaboration with CMake (see below) without additional generator.

zlib

VRE can use zlib for writing compressed data, to be installed on Linux or macOS using the usual package manager. It can be installed on Windows with the open source c++ library manager VCPKG. To get VCPG, see https://vcpkg.io/en/getting-started.html. And to install ZLIB with VCPKG:

```
vcpkg install --triplet x64-windows zlib
```

To make VCPKG visible to CMake, create an environment variable `VCPKG_ROOT` pointing to the root of the vcpkg directory and configure VRE with the flag

`-DCMAKE_TOOLCHAIN_FILE=%VCPKG_ROOT%/scripts/buildsystems/vcpkg.cmake`.

To use VCPKG with Visual Studio add the toolChainFile to the configurePresets in the CMakePresets.json:

```
"toolchainFile":  "$env{VCPKG_ROOT}/scripts/buildsystems/vcpkg.cmake",
```

To enable ZLIB support in VRE, configure CMake with the flag `-DVRE_USE_ZLIB=ON`.

Eigen

VRE requires the Eigen libraries (https://gitlab.com/libeigen/eigen) for some Credit Risk analytics.

Pybind11 and Python

To build the VRE Python module from source you need Python 3.12 or newer plus pybind11 headers and scikit-build-core. We provide helper scripts per platform that stage dependencies and run cibuildwheel:

- Linux: `vre_python/linux_build_wheels_cpu_kernels.sh` (or `linux_build_wheels_no_kernels.sh`)
- macOS: `vre_python/mac_build_wheels_cpu_kernels.sh` (or `mac_build_wheels_no_kernels.sh`)
- Windows: `vre_python/build_wheels_no_kernels.ps1`

Each script pins the required Python ABI(s) via `CIBW_BUILD` / preset selection and bundles pybind11 and scikit-build-core automatically inside the wheel build. If you need to build manually outside these scripts, install pybind11 and scikit-build-core first (e.g. `python -m pip install pybind11 scikit-build-core`). Using the prebuilt wheels from PyPI (section 9.16) does not require a source build.

8.4 Building VRE

VRE uses CMake presets for common configurations (see `CMakePresets.json`). Typical invocations are:

- `cmake -preset linux-gcc-ninja-release` (Linux, GCC, Release)
- `cmake -build . -preset linux-gcc-ninja-release -parallel` (build)
- `ctest -preset linux-gcc-ninja-release` (tests)

Choose the preset closest to your platform/toolchain (macOS, Windows, CUDA/OpenCL, clang-tidy, etc.) and override options with `-DVAR=VALUE` if needed. The tables below list the key cache variables you may override explicitly.

VRE uses the CMake build system with the following essential parameters (see vre/CMakeLists.txt, vre/vre_quantlib_ext/CMakeLists.txt, vre/vre_quantlib/CMakeLists.txt) which affect the build:

Option	Description	Preset usage
VRE_BUILD_DOC	Build PDF documentation	ON (OFF in wheel/CI presets)
VRE_BUILD_EXAMPLES	Build examples	ON (OFF in wheel presets)
VRE_BUILD_TESTS	Build test suites	ON (OFF in wheel presets)
VRE_BUILD_APP	Build the VRE commandline application `vre[.exe]`	ON (OFF in wheel presets)
VRE_ENABLE_OPENCL	Build the compute environment using OpenCL, to utilise a GPU	OFF (enable in OpenCL presets)
VRE_ENABLE_CUDA	Build the compute environment using CUDA, to utilise a GPU	OFF (enable in CUDA presets)
VRE_USE_ZLIB	Use compression for boost::iostreams, e.g. to write compressed cube files	OFF (enabled in release presets)
VRE_MULTITHREADING _CPU_AFFINITY	Set cpu affinitity in multithreaded calculations	OFF
VRE_ENABLE_PARALLEL _UNIT_TEST_-RUNNER	Enable the parallel unit test runner	OFF (enable on Linux if Boost supports)
MSVC_LINK_DYNAMIC_RUNTIME	Link against dynamic runtime	ON (MSVC presets)
MSVC_PARALLELBUILD	Use flag /MP	ON
QL_USE_PCH	Use precompiled headers	OFF (ON in clang-tidy presets)
QL_ENABLE_SESSIONS	Singletons return different instances for different sessions; turn ON for VRE's multi-threading	OFF (ON in VRE defaults)
QL_BUILD_EXAMPLES	Build QuantLib examples	ON (OFF in wheel/CI presets)
QL_BUILD_TEST_SUITE	Build QuantLib test suite	ON (OFF in wheel/CI presets)
VRE_ENABLE_ENZYME	Enable Enzyme/LLVM AAD backend	OFF (ON in AAD/enzyme presets; requires Enzyme/LLVM paths)
VRE_ENABLE_METAL	Enable Apple Metal GPU backend	OFF (ON in macOS Metal presets)
BOOST_INCLUDEDIR, BOOST_ROOT	directory containing the "boost" folder, root of all boost includes	Set in presets when staging deps; user-provided otherwise
BOOST_LIBRARYDIR	folder containing compiled boost libraries	Set in presets when staging deps; user-provided otherwise
Python_ROOT	directory containing Python.h	Set in wheel presets; user-provided otherwise
Python_LIBRARY	Python shared library name including full path	Set in wheel presets; user-provided otherwise

The latter BOOST and Python variables have been added to the list, because CMake may use them to locate the BOOST and Python development files, if CMake does not find them automatically.

Linux and macOS

Use presets from the repo root; CMake will place build trees under `build/` automatically:

- Configure: `cmake -preset linux-gcc-ninja-release`
- Build: `cmake -build . -preset linux-gcc-ninja-release -parallel`
- Test: `ctest -preset linux-gcc-ninja-release`

Pick the preset matching your toolchain (clang, OpenCL/CUDA, clang-tidy, wheel builds, etc.). To override a cache option for a preset, append `-DVAR=VALUE` to the configure step (e.g. `cmake -preset linux-gcc-ninja-release -DVRE_USE_ZLIB=ON`). Paths to staged Boost/Python/Enzyme are already wired into the wheel presets; otherwise provide them via the variables listed above.

Windows

Visual Studio 2019+ reads `CMakePresets.json` automatically when you open the folder:

- Configure/Build/Test via VS using the desired preset (e.g. `windows-msvc-ninja-release`, `windows-msvc-ninja-cuda`, `windows-msvc-ninja-opencl`, `windows-msvc-ninja-clang-tidy`).
- Command line equivalent: `cmake -preset windows-msvc-ninja-release`, then `cmake -build . -preset windows-msvc-ninja-release -parallel`, `ctest -preset windows-msvc-ninja-release`.

If you need a different generator (e.g. VS2019 instead of VS2022), adjust the preset's generator in `CMakePresets.json` rather than crafting raw `-G` command lines.

9 Examples

VRE examples are organized around a forward regulatory and domain ownership model. Table 6 describes the current maintained homes.

Use each subsection and its associated directory to inspect portfolio setup, market/configuration setup, analytics selection, outputs and validation workflows.

Example family	Description
Getting Started	Minimal onboarding cases in `examples/getting_started`: minimal setup, equity option, IR swap and reporting currency
Analytic Demos	Product coverage, curve building, python, and performance examples under the forward demo/economic surfaces
Regulatory Pillar 1	Maintained entrypoints for direct credit, CCR, CVA, market risk, and IMA validation under `examples/regulatory/pillar_1`
Regulatory Pillar 2	Banking-book IRRBB, CSRBB, and stress-testing examples under `examples/regulatory/pillar_2`
Cross-Pillar Capital Floor	Maintained capital-floor true-flow example under `examples/regulatory/cross_pillar/capital_floor`
Regulatory Pillar 3	Forward disclosure and prudential-return entrypoints under `examples/regulatory/pillar_3`
Cross-Pillar Scoped Clients	Reviewer-facing packs such as Southern Cross, Harbour Bridge, and APRA-vs-Basel under `examples/regulatory/cross_pillar`
Domain And Demo Families	Exposure, initial margin, XVA, curve building, products, and performance examples under `examples/economic/exposure`, `examples/economic/initial_margin`, `examples/economic/xva`, `examples/curve_building`, `examples/products`, and `examples/performance`

Table 6: VRE example topics.

Running any example in this section requires Python 3.

All example results can be produced using Python scripts in each example subdirectory. In older VRE releases these scripts were almost always thin wrappers around a master XML input file. In the current VRE v0.11.2 release line, the picture is broader. The example estate now contains:

- native `TOML` launch bundles,
- runtime-ready package roots under `input_native`,
- published machine contracts and manifests,
- governed expected-output evidence surfaces.

Current Example Execution Model

The examples should now be read in three layers:

1. the maintained forward entrypoint under `examples/regulatory/...` or maintained domain homes such as `examples/economic/exposure/...` and `examples/economic/xva/...`,
2. the underlying authored bundle or runtime package that the runner feeds into VRE,
3. the committed expected-output/evidence surface where applicable.

For current examples, the preferred path is:

`python runner.py` → `TOML source bundle / input_native package` → VRE runtime → reports + evidence

The maintained regression surface is intentionally split between lean CI lanes, focused domain lanes, and reporting/governed proof lanes. The heavy manual lane is still `e2e-core-full`, but it is no longer the only meaningful reading of the maintained example estate. The lean maintained lane is now `e2e-core`, with additional focused lanes for `credit_risk`, `cva_risk`, `market_risk`, `xva_risk`, reporting scope, IRRBB-only reporting, Pillar 2 banking-book true flow, and the capital-floor true-flow example.

APRA and broader regulatory example coverage is maintained as a scope-native inventory and committed expected-output estate under `examples/common/expected_output/reporting`, `examples/common/expected_output/reporting/apra`, and the governed reference-pack goldens under `examples/regulatory/cross_pillar/scoped_clients/.../expected_output`. These files are reviewer goldens and evidence surfaces; release claims still come from `VRECompliance` packs and validators.

Outside that heavy core gate, `economic/exposure` remains one of the most important large reference families in the repository and is still documented in detail below because it is the clearest long-form illustration of the engine's exposure, collateral and XVA behaviour.

Current maintained examples use the forward target-state tree:

- `examples/regulatory/pillar_1/...`

- examples/regulatory/pillar_2/banking_book/...
- examples/regulatory/cross_pillar/capital_floor/...
- examples/regulatory/pillar_3/...
- examples/regulatory/cross_pillar/...
- examples/economic/exposure/..., examples/economic/xva/..., examples/economic/initial_margin/...
- examples/regulatory/pillar_1/credit/ccr/migrated_examples/..., examples/regulatory/pillar_1/credit/cva/migrated_examples/..., examples/regulatory/pillar_1/market_risk/migrated_examples/... as implementation/reference homes only; they are not part of the maintained forward launch contract
- build/examples_runtime/... for default generated runtime output
- committed reviewer goldens under examples/common/expected_output/... and examples/regulatory/cross_pillar/scoped_clients/.../expected_output

The XML file families listed in table 9 are archival compatibility reference only. They are outside the maintained v3 launch model for current example execution.

9.1 Maintained Regulatory and Reporting Examples

The maintained regulatory example estate is centered on:

- forward regulatory entrypoints under examples/regulatory/pillar_1, examples/regulatory/pillar_2, examples/regulatory/pillar_3, and examples/regulatory/cross_pillar;
- evidence-bearing reporting packs under examples/regulatory/pillar_3/prudential_returns/migrated_examples.

Domain source runners such as examples/regulatory/pillar_1/market_risk/migrated_examples, examples/regulatory/pillar_1/credit/cva/migrated_examples, and examples/regulatory/pillar_1/credit/ccr/migrated_examples remain implementation/reference homes only and are not part of the maintained forward launch guidance.

These are not ordinary demo fixtures. Many of their committed expected outputs are consumed as evidence surfaces by vre_compliance, so they should be read as governed regression assets.

Forward regulatory entrypoints now execute directly from examples/regulatory without topic-runner delegation.

9.1.1 Maintained reporting true-flow estate

The current maintained regulatory examples include:

- direct-credit true-flow packs for SA-CR, FIRB, and AIRB;

- banking-book Pillar 2 true-flow packs for IRRBB common measurement, AU capital overlay, AU APS 117 raw components, CSRBB, and stress-testing compute support;
- APRA reporting and comparison packs such as `apra_ars116_reference_pack`, `apra_ars180_reference_pack`, and `apra_basel_comparison`;
- scoped-client reporting packs such as `southern_cross_cross_border` and `harbour_bridge_multientity`;
- forward Pillar 1 entrypoints for IMM, SA-CCR, SA-CVA, BA-CVA, IMA, SA-MR, and Simplified MR.

9.1.2 Current E2E lanes

The maintained CTest lanes are now intentionally split:

- `e2e-core`: lean representative domain runs plus selected governed reporting E2Es;
- `e2e-credit-risk`, `e2e-cva-risk`, `e2e-market-risk`, `e2e-xva-risk`: focused full domain lanes;
- `e2e-scope`: reporting-example-only lane;
- `e2e-irrbb`: executable IRRBB reporting-example lane;
- `e2e-pillar2`: banking-book Pillar 2 true-flow lane;
- `e2e-capital-floor`: capital-floor true-flow lane;
- `e2e-core-full`: manual heavy regression lane.

The important shift is that reporting is now a first-class maintained example surface rather than an afterthought attached to domain demos.

9.1.3 How to read the examples

The maintained estate is best read in five layers:

1. **regulatory entrypoints**: forward homes under `examples/regulatory/...` for maintained Pillar 1, Pillar 2, Pillar 3, and cross-pillar runs;
2. **domain source runners**: implementation or reference homes for selected analytic families; not part of the maintained forward launch contract;
3. **reporting evidence assets**: pack manifests and expected outputs coupled to compliance evidence;
4. **scoped-client examples**: multi-pack or reviewer-facing narratives that demonstrate supervisory/reporting compositions;
5. **archive examples**: historical compatibility material under `examples/archive/...`, not part of maintained launch guidance.

9.1.4 Pillar 2 and IMA examples

For the v0.11.x regulatory estate, the most important maintained examples are:

- IRRBB common-core, AU overlay, and AU APS 117 raw-component true-flow runs under `examples/regulatory/pillar_2/banking_book/irrbb/...`;
- CSRBB and stress-testing compute-support true-flow runs under `examples/regulatory/pillar_2/banking_book/...`;
- IMA validation and IMA capstone runs under `examples/regulatory/pillar_1/market_risk/ima`;
- capital-floor synthetic true-flow runs under `examples/regulatory/cross_pillar/capital_floor/...` that demonstrate the bounded VRE-owned part of the joint delivery model.

These examples prove:

- flow-job routing;
- country/regime-profile selection;
- governed canonical output contracts;
- release-pack and evidence integration.

Maintained Example Toggle Inventory

The maintained example/testrunner toggle inventory is generated from runner source. The machine-readable source of truth is `configurations/inputs/coverage/example_toggle_inventory.json`.

Toggle	Category	Defaults	First observed usage
EXTERNAL_COMPUTE_DEVICE	other	-	examples/vre_examples_helper.py:843
MARKETRISK_PARITY_ENFORCE_FD	market_risk_lane	0	examples/regulatory/pillar_1/market_risk/migrated_examples/run_marketrisk_ima_samr_parity.py:273
MARKETRISK_PARITY_GPU_DYNAMIC	market_risk_lane	1	examples/regulatory/pillar_1/market_risk/migrated_examples/run_marketrisk_ima_samr_parity.py:297
MARKETRISK_PARITY_INCLUDE_GPU	market_risk_lane	0, 1	examples/regulatory/pillar_1/market_risk/migrated_examples/run_marketrisk_ima_samr_parity.py:274
MARKETRISK_PARITY_PUBLISH_MIXED	market_risk_lane	1	examples/regulatory/pillar_1/market_risk/migrated_examples/run_marketrisk_ima_samr_parity.py:275
MARKETRISK_PARITY_REUSE_FD	market_risk_lane	0	examples/regulatory/pillar_1/market_risk/migrated_examples/run_marketrisk_ima_samr_parity.py:272

Toggle	Category	Defaults	First observed usage
MARKETRISK_PARITY_SAMPLES	market_risk_lane	8	examples/regulatory/pillar_1/market_risk/migrated_examples/run_marketrisk_ima_samr_parity.py:271
MARKETRISK_PROGRESSIVE_FAST_SAMPLES	market_risk_lane	256	examples/regulatory/pillar_1/market_risk/migrated_examples/run_marketrisk_fd_aad_progressive_parity.py:318
MARKETRISK_PROGRESSIVE_LIMIT_IMA	market_risk_lane	-	examples/regulatory/pillar_1/market_risk/migrated_examples/run_marketrisk_fd_aad_progressive_parity.py:327
MARKETRISK_PROGRESSIVE_LIMIT_SA	market_risk_lane	-	examples/regulatory/pillar_1/market_risk/migrated_examples/run_marketrisk_fd_aad_progressive_parity.py:326
OVERWRITE_SCENARIOGENERATOR_SAMPLES	test_harness	20, 50	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_backend_parity.py:230
VRE_AD_CE_FREEZE_BASIS	runtime_tuning	1	examples/regulatory/pillar_1/market_risk/migrated_examples/run_marketrisk_fd_aad_progressive_parity.py:316
VRE_AD_PRECE_LINEARIZATION	runtime_tuning	1	examples/regulatory/pillar_1/market_risk/migrated_examples/run_marketrisk_fd_aad_progressive_parity.py:317
VRE_CAM_EXT_DRY	other	0	examples/regulatory/pillar_1/credit/cva/migrated_examples/run.py:16
VRE_CAM_STRICT_SAMPLES	other	16	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:70
VRE_CG_EXTERNAL_COMPUTE_DEVICE	runtime_tuning	CUDA/NVIDIA GB10	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:72
VRE_CG_FULL_DEVICE_POLICY	runtime_tuning	Strict	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:483
VRE_CG_FULL_FORWARD_POLICY	runtime_tuning	Strict	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:484
VRE_CG_FULL_REVERSE_POLICY	runtime_tuning	Strict	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:485

Examples/NVIDIA

Toggle	Category	Defaults	First observed usage
VRE_CG_SENSI_-TRACE_ENABLE	runtime_-tuning	-	examples/regulatory/pillar_1/cred-it/cva/migrated_examples/run_-sacva_eqcom.py:52
VRE_CG_SENSI_-TRACE_FILE	runtime_-tuning	cg_trace_-aad.csv	examples/regulatory/pillar_1/cred-it/cva/migrated_examples/run_-sacva_eqcom.py:66
VRE_CG_SENSI_-TRACE_SCEN	runtime_-tuning	*	examples/regulatory/pillar_1/cred-it/cva/migrated_examples/run_-sacva_eqcom.py:64
VRE_CG_SENSI_-TRACE_TOP	runtime_-tuning	20, 2000	examples/regulatory/pillar_1/cred-it/cva/migrated_examples/run_-sacva_eqcom.py:65
VRE_CG_SKIP_-HOST_FORWARD	runtime_-tuning	1	examples/regulatory/pillar_1/cred-it/cva/migrated_examples/run_-sacva_cuda_reverse_dod.py:507
VRE_CMAKE_PRE-SET	runner_con-trol	-	examples/regulatory/pillar_1/cred-it/ccr/migrated_examples/run_-cre_plan_examples.py:56
VRE_CUDA_DOD_-CAM_CONFIG	runtime_-tuning	input/vre_-sacva_cam_-ext_cg_-ad_all_fac-tors_cuda_-dod.toml	examples/regulatory/pillar_1/cred-it/cva/migrated_examples/run_-sacva_cuda_reverse_dod.py:49
VRE_CUDA_DOD_-CAM_PATCH_CON-FIGS	runtime_-tuning	-	examples/regulatory/pillar_1/cred-it/cva/migrated_examples/run_-sacva_cuda_reverse_dod.py:58
VRE_CUDA_DOD_-CE_CACHE	runtime_-tuning	-	examples/regulatory/pillar_1/cred-it/cva/migrated_examples/run_-sacva_cuda_reverse_dod.py:73
VRE_CUDA_DOD_-FAST_MATH	runtime_-tuning	-	examples/regulatory/pillar_1/cred-it/cva/migrated_examples/run_-sacva_cuda_reverse_dod.py:78
VRE_CUDA_DOD_-FORCE_RUN	runtime_-tuning	-	examples/regulatory/pillar_1/cred-it/cva/migrated_examples/run_-sacva_cuda_reverse_dod.py:120
VRE_CUDA_DOD_-GPU_CONFIG	runtime_-tuning	input/vre_-sacva_cg_-gpu_eq-com_cuda_-dod.toml	examples/regulatory/pillar_1/cred-it/cva/migrated_examples/run_-sacva_cuda_reverse_dod.py:48
VRE_CUDA_DOD_-GPU_PATCH_CON-FIGS	runtime_-tuning	-	examples/regulatory/pillar_1/cred-it/cva/migrated_examples/run_-sacva_cuda_reverse_dod.py:53

Toggle	Category	Defaults	First observed usage
VRE_CUDA_DOD_HOST_CONFIG	runtime_tuning	input/vre_sacva_cg_ad_eq-com_cuda_dod.toml	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:47
VRE_CUDA_DOD_PRECISION_MODE	runtime_tuning	Float64	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:77
VRE_CUDA_MAX_OPS_PER_KERNEL	runtime_tuning	2048	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:71
VRE_CUDA_NVRTC_GPU_ARCH	runtime_tuning	-	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:91
VRE_CVARISK_FORCE_CUDA_DOD	other	-	examples/regulatory/pillar_1/credit/cva/migrated_examples/run.py:31
VRE_ENABLE_PHASE2_APRA	regulatory_scope	-	examples/regulatory/pillar_1/market_risk/migrated_examples/run_scope_multile_and_mixed.py:26
VRE_EXAMPLES_E2E_PROFILE	runner_control	full	examples/regulatory/pillar_1/credit/ccr/migrated_examples/run.py:31
VRE_EXAMPLES_GENERATED_CONFIG_DIR	runner_control	-	examples/run_examples_testsuite.py:155
VRE_EXAMPLES_INCLUDE_GPU	runner_control	0	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva.py:54
VRE_EXAMPLES_LOCK_TIMEOUT_SECONDS	runner_control	1800	examples/vre_examples_helper.py:245
VRE_EXAMPLES_REPO_ROOT	runner_control	REPO_ROOT, SCRIPT_DIR.parent.parent	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_backend_parity.py:28
VRE_EXAMPLES_SERIALIZE_RUNS	runner_control	-	examples/vre_examples_helper.py:220
VRE_EXAMPLES_USE_PYTHON	runner_control	-	examples/vre_examples_helper.py:194
VRE_EXAMPLE_SET	runner_control	all	examples/run_examples_testsuite.py:236
VRE_EXECUTABLE	runner_control	-	examples/regulatory/pillar_1/credit/ccr/migrated_examples/run_cre_plan_examples.py:58

Toggle	Category	Defaults	First observed usage
VRE_INCLUDE_DEFERRED_APRA_SCOPE	regulatory_scope	-	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_scope_mixed_methods.py:21
VRE_METAL_FORCE_HOST_REVERSE	other	1	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:503
VRE_SACVA_CG_SAMPLES	other	50	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cg.py:28
VRE_SACVA_CUDA_DOD_SCENARIO_ABS_TOL	other	1.0e-8	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:68
VRE_SACVA_CUDA_DOD_SCENARIO_REL_TOL	other	1.0e-4	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:67
VRE_SACVA_CUDA_DOD_SENSI_ABS_TOL	other	1.0e-8	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:66
VRE_SACVA_CUDA_DOD_SENSI_REL_TOL	other	5.0e-5	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_cuda_reverse_dod.py:65
VRE_SACVA_LARGER_DRY	other	0	examples/regulatory/pillar_1/credit/cva/migrated_examples/run.py:17
VRE_SACVA_LARGER_SKIP_GPU	other	-	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_larger_portfolio.py:65
VRE_SACVA_LARGER_STRICT_GPU	other	-	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_larger_portfolio.py:72
VRE_SACVA_PARITY_FD_CONFIG	other	input/vre_sacva_fd_eqcom.toml	examples/regulatory/pillar_1/credit/cva/migrated_examples/run_sacva_backend_parity.py:263
VRE_SCENARIO_SUPERSET_ALLOWED	other	true	examples/economic/exposure/non_collateral/run_hw2f.py:13
VRE_STRICT_CORRELATIONS	other	false	examples/economic/exposure/non_collateral/run_hw2f.py:12
VRE_STRICT_DIMENSION_MATCH	other	false	examples/economic/exposure/non_collateral/run_hw2f.py:11

The typical list of output files and reports is shown in table 10. The names of output files are configured in the maintained run bundle, usually via the relevant `[analytics.<family>]` block in the topic-specific `vre*.toml` file. Whether these reports are generated depends on which analytics are active in that bundle. In current maintained examples, default generated output is written under `build/examples_runtime/...`, while repo-committed example output is reserved for governed expected-output fixtures.

Current surface	Operational meaning
`*.toml`	Authored run bundle or source bundle used by maintained example runners
`input_native/`	Runtime-ready package root used by current native example flows
`runtime_policy/` and published manifests	Published machine contracts, registries and evidence inputs where applicable
compatibility `*.xml`	Archival compatibility payloads retained for explicit engine seams only; not part of the maintained launch surface

Table 8: Current example launch surfaces in VRE v0.11.2.

File Name	Description
`vre.xml`	Master input file, selection of further inputs below and selection of analytics
`portfolio.xml`	Trade data
`netting.xml`	Collateral (CSA) data
`simulation.xml`	Configuration of simulation model and market
`market.txt`	Market data snapshot
`fixings.txt`	Index fixing history
`dividends.txt`	Dividends history
`curveconfig.xml`	Curve and term structure composition from individual market instruments
`conventions.xml`	Market conventions for all market data points
`todaysmarket.xml`	Configuration of the market composition, relevant for the pricing of the given portfolio as of today (yield curves, FX rates, volatility surfaces etc)
`pricingengines.xml`	Configuration of pricing methods by product

Table 9: Legacy XML compatibility input files.

Note: When building VRE from sources on Windows platforms, make sure that you copy your `vre.exe` to the binary directory `vre_app/bin/win32/` respectively `vre_app/bin/x64/`. Otherwise the examples may be run using the pre-compiled executables which come with the VRE release.

9.2 Getting Started

The onboarding examples live in `examples/getting_started` and provide concise, single-trade cases with transparent market and configuration setups:

- `00-MinimalSetup`: minimal configuration for the opening swap exposure batch;
- `01-EquityOption`: NPV of a single European equity option;
- `02-IRSwap`: NPV/cashflow/curve reporting for a single vanilla IR swap;
- `03-ReportingCurrency`: reporting-currency setup with USD-in-EUR collateralisation.

Each case runs via `python run.py` from its subdirectory.

9.3 Minimal Setup

The example in `examples/getting_started/00_minimal_setup` demonstrates the minimal market data setup required to run the opening vanilla swap exposure simulation in section 9.10. The market setup uses single points per curve where possible.

Run with: `python run.py`

9.4 Products

The product examples are organised under `examples/products`:

File Name	Description
npv.csv	NPV report
flows.csv	Cashflow report
curves.csv	Generated yield (discount) curves report
xva.csv	XVA report, value adjustments at netting set and trade level
exposure_trade_*.csv	Trade exposure evolution reports
exposure_nettingset_*.csv	Netting set exposure evolution reports
rawcube.csv	NPV cube in readable text format
netcube.csv	NPV cube after netting and collateral, in readable text format
*.csv.gz	Intermediate storage of NPV cube and scenario data
*.pdf	Graphics generated by Python helper scripts after VRE completed

Table 10: Typical report and output files.

- `Classic`: 150+ sample trades across asset classes, vanilla and complex, with the maintained run controlled through the native bundle surface and the trade payload retained as an XML exception;
- `scripted_trade`: scripted-payoff examples with classic-vs-scripted parity checks.

The classic sample trades are in `examples/products/classic/Example_Trades` and are concatenated into the maintained trade payload `examples/products/classic/input/portfolio.xml`. The same input folder contains the native run bundle, market data, conventions and published runtime artefacts for the valuation batch.

Run classic products with: `cd examples/products/classic && python run.py`

See the **Trade Data And Trade Components** chapter in this guide for the maintained portfolio payload, trade payload and trade-component reference, and the standalone `docs/user_guide/products.tex|pdf` catalogue for the companion pricing-methodology volume.

In addition to `npv` and `cashflow`, the maintained products batch enables `portfolioDetails` in `examples/products/classic/input/vre.toml`:

```
[analytics.npv]
active = "Y"
baseCurrency = "USD"
outputFileName = "npv.csv"
additionalResults = "Y"
additionalResultsReportPrecision = "12"

[analytics.cashflow]
active = "N"
outputFileName = "flows.csv"

[analytics.portfolioDetails]
active = "Y"
riskFactorFileName = "riskFactors.csv"
marketObjectFileName = "marketObjects.csv"
```

This leads to the reporting of portfolio composition in a series of extra files in `examples/products/classic/output`:

- counterparties.csv
- marketObjects.csv
- netting_sets.csv

- riskFactors.csv
- swap_indices.csv
- trade_types.csv
- underlying_indices.csv

For scripted products run:

```
cd examples/products/scripted_trade && python run.py
```

This produces standard reports plus `npv_parity.csv` comparing adjacent scripted and classic representations. See section 9.4.1 for details.

9.4.1 Scripted Trade

The scripted trade examples are maintained in `examples/products/scripted_trade`. They demonstrate how to represent path-dependent and hybrid payoffs using the scripted-trade data model while keeping classic XML trades in the same portfolio for parity checks.

See the separate documentation in folder `docs/scripted_trade` for an introduction to trade representation, scripting language, model setup and pricing engine configuration.

Run the packaged case with

```
cd examples/products/scripted_trade && python run.py
```

This runs `input/vre.toml` and then executes `parity.py` to generate `examples/products/scripted_trade/output/npv_parity.csv`. The report compares adjacent classic vs scripted trades by trade-id group and checks maturity/currency consistency.

The portfolio covers the following pairs and variants:

- Trade 1: Vanilla European Equity Option, represented as a standard trade-payload entry with analytical pricing
- Trade 2: Same Option as above, represented as "generic" scripted trade with scripted payoff embedded into the trade payload, pricing via Monte Carlo
- Trade 3: Same Option as above, same representation, pricing via Finite Differences triggered by a `ProductTag` assigned to the script and used in the published pricing policy
- Trade 4: Same Option as above, the scripted trade now refers to an "external" script in `scriptlibrary.xml`, MC pricing
- Trade 4b: Same as trade 4, but "compact" scripted trade representation (uncomment trade 4b in `portfolio.xml`)
- Trade 5: Barrier Option with single continuously observed Up & Out barrier, represented as a standard trade-payload entry with analytical pricing
- Trade 6: Same Barrier Option as above, approximated as generic scripted trade with daily barrier observation

- Trade 6b: Same Barrier Option as above, approximated as "compact" scripted trade with daily barrier observation (uncomment trade 6b in `portfolio.xml`)
- Trade 7: Same Barrier Option as above, represented as generic scripted trade with continuously observed barrier, i.e. adjusting for the probability of knock-out between daily observations
- Trade 7b: Same Barrier Option as of above, represented as "compact" scripted trade (uncomment trade 7b in `portfolio.xml`)
- Trade 8: Equity Accumulator, represented as generic scripted trade with external payoff script
- Trade 8b: Same Equity Accumulator as above, represented as compact scripted trade with external payoff script (uncomment trade 8b in `portfolio.xml`)

Note:

- In all cases we use the Black-Scholes model to drive the Equity process.
- The Barrier Option pricing using the scripted trade deviates noticeably from the analytical pricing when we use daily observations (trade 6 and 6b), but matches quite closely when we adjust for the probability of knock-out between observation dates (trade 7 and 7b)
- We are not aware of analytical pricing for the Accumulator product in trade 8 to benchmark against; trade 8 is priced with MC, FD pricing of the Accumulator is possible as well but requires a separate payoff script; only in the vanilla European option case we can utilize the same script for both MC and FD pricing
- This pack demonstrates MC and FD scripted pricing modes; cross-domain FD/AAD/GPU parity workflows are documented in sections 9.7.11 and 9.8.4.

Though this initial example shows only single-asset Equity cases, the scripted trade in its current version is significantly more versatile, more examples and scripts to follow.

9.5 Curve Building

This directory demonstrates several curve building cases

- Bootstrap Consistency: `python run_consistency.py`
- Discount Ratio Curves: `python run_discountratio.py`
- Fixed vs Float Cross Currency Helpers: `python run_fixedfloatccs.py`
- USD-Prime Curve Building via Prime-LIBOR Basis Swaps: `python run_prime.py`
- Bond Yield Shifted Curves: `python run_bondyieldshifted.py`
- Central Bank Meeting Dates: `python run_centralbank.py`
- SABR Swaption and Cap/Floor Volatilities: `python run_sabr.py`

Run all with: `python run.py`.

9.5.1 Bootstrap Consistency

This example confirms that bootstrapped curves correctly reprice the bootstrap instruments (FRAs, Interest Rate Swaps, FX Forwards, Cross Currency Basis Swaps) using three pricing setups with

- EUR collateral discounting (configuration xois_eur)
- USD collateral discounting (configuration xois_usd)
- in-currency OIS discounting (configuration collateral_inccy)

The required portfolio files need to be generated from market and reference contracts in `examples/common/input/runtime_policy/published` and `examples/common/input/reference_registry/published`, and trade templates in `examples/curve_building/helpers`, calling

```
python TradeGenerator.py
```

at the `examples/curve_building` level.

This will place three portfolio files `*_portfolio.xml` in the input folder. Thereafter, the three consistency checks can be run calling

```
python run_consistency.py
```

at the `examples/curve_building` level.

Results are in three files `*_npv.csv` in `examples/curve_building/output/consistency` and should show zero NPVs for all benchmark instruments.

9.5.2 Discount Ratio Curves

This example shows how to use a yield curve built from a DiscountRatio segment. In particular, it builds a GBP collateralized in EUR discount curve by referencing three other discount curves:

- a GBP collateralised in USD curve
- a EUR collateralised in USD curve
- a EUR OIS curve i.e. a EUR collateralised in EUR curve

The implicit assumption in building the curve this way is that EUR/GBP FX forwards collateralised in EUR have the same fair market rate as EUR/GBP FX forwards collateralised in USD. This assumption is illustrated in the example by the NPV of the two forward instruments in the portfolio returning exactly 0 under both discounting regimes i.e. under USD collateralization with direct curve building and under EUR collateralization with the discount ratio modified "GBP-IN-EUR" curve.

Also, in this example, an assumption is made that there are no direct GBP/EUR FX forward or cross currency quotes available which in general is false. The example s merely for illustration.

Both collateralization scenarios can be run calling `python run.py`.

9.5.3 Fixed vs Float Cross Currency Helpers

This example demonstrates using fixed vs. float cross currency swap helpers. In particular, it builds a TRY collateralised in USD discount curve using TRY annual fixed vs USD 3M Libor swap quotes.

The portfolio contains an at-market fixed vs. float cross currency swap that is included in the curve building. The NPV of this swap should be zero when the example is run, using `python run_fixedfloatccs.py` at the `examples/curve_building` level or "directly" calling the maintained run bundle for that case.

9.5.4 USD-Prime Curve Building

This example demonstrates the implementation of the USD-Prime index in the VRE. The USD-Prime yield curve is built from USD-Prime vs USD 3M Libor basis swap quotes. The portfolio consists of two fair basis swaps (NPVs equal to 0):

- US Dollar Prime Rate vs 3 Month LIBOR
- US Dollar 3 Month LIBOR vs Fed Funds + 0.027

In particular, it is confirmed that the bootstrapped curves USD-FedFunds and USD-Prime follow the 3% rule observed on the market: `U.S. Prime Rate = (The Fed Funds Target Rate + 3%)`. (See http://www.fedprimerate.com/.)

Running VRE in directory `examples/curve_building` with `python run_prime.py` yields the USD-Prime curve in `examples/curve_building/output/prime/curves.csv`.

9.5.5 Bond Yield Shifted Curves

This example shows how to use a yield curve built from a BondYieldShifted segment, as described in the standalone yield-curve configuration reference in `docs/user_guide/curve_configurations/yieldcurves.tex`.

In particular, it builds the curve `USD.BMK.GVN.CURVE_SHIFTED` shifted by three liquid Bonds:

- Fixed rate USD Bond maturing in August 2023 with id `EJ7706660`.
- Fixed rate USD Bond maturing in September 2049 with id `ZR5330686`.
- Floating Rate Bond maturing in May 2025 with id `AS064441`.

The resulting curve is exhibited in the `curves.csv` output file. Moreover, the results can be cross checked against the NPVs, i.e. prices, of the ZeroBonds comprised in the portfolio.

- `ZeroBond_long`, maturing 2052-03-01 shows a price of 0.2080 akin to the 0.2080 in the curves output at the same date.
- `ZeroBond_short`, maturing 2032-06-01 shows a price of 0.5808 aktin to the 0.808 in the curves output at the same date.

The example can be run calling `python run_bondyieldshifted.py` at the `examples/curve_building` level.

9.5.6 Central Bank Meeting Dates

This example demonstrates the build of a GBP OIS curve using MPC Swaps at the short end, i.e. using OIS Swaps with concrete forward start and end date.

9.5.7 SABR Volatility Surfaces

This example demonstrates the pricing of a Swaption and a Cap on volatility surfaces that are interpolated in smile direction using a SABR model flavour. As usual the example is run by calling `python run.py`

The essential configuration is in the maintained curve-set and market-view contracts for the SABR example, where the Interpolation (Swaption) resp. StrikeInterpolation (Caps/Floors) allows the following new SABR types

- Hagan2002Lognormal
- Hagan2002Normal
- Hagan2002NormalZeroBeta
- Antonov2015FreeBoundaryNormal
- KienitzLawsonSwaynePde
- FlochKennedy

SABR parameters can be calibrated or have fixed externally provided values per option tenor and Swap tenor (Swaptions) resp. optionlet (Caps/Floors).

9.6 Reporting

The current cross-domain regulatory packs live under `examples/regulatory/cross_pillar/scoped_clients`, with shared runner utilities under `examples/regulatory/cross_pillar/reference_packs`. Unlike the single-domain folders, these packs combine maintained jobs from the forward Pillar 1, Pillar 2 and Pillar 3 example trees into one reporting narrative.

Run the default reference pack with:

```
python3 examples/regulatory/cross_pillar/reference_packs/run_reference_pack.py
```

or rebuild only the pack summary with:

```
python3 examples/regulatory/cross_pillar/reference_packs/run_reference_pack.py -summary-only
```

These packs are authored in YAML under `examples/regulatory/cross_pillar/scoped_clients`. They demonstrate:

- cross-domain orchestration over the maintained regulatory example families,
- explicit scope membership, jurisdiction routing and report-family selection,
- Basel, EU, PRA/PR, US and MAS reporting views built from the same deterministic engine runs,

- compact summary artifacts under the maintained scoped-client pack homes that can be reviewed without reading the full domain output estate.

Two maintained manifests are currently checked in:

- `examples/regulatory/cross_pillar/scoped_clients/southern_cross_-cross_border/reference_pack.yaml`: a consolidated cross-border reference pack spanning Basel CCR/CVA, Basel market risk, PRA CCP, EU IMM, US CCP and MAS IMM views;
- `examples/regulatory/cross_pillar/scoped_clients/harbour_bridge_-multientity/reference_pack.yaml`: a richer multi-entity pack with separate legal-entity jobs across EU, PRA, US and MAS jurisdiction slices.

Each manifest records the shared scope, the scenario bundles to execute, the regulator view, the metric-extraction rules, and the reviewer-facing summary outputs. The maintained summaries are written as both CSV and JSON so they are usable for human review and automated quality checks. Typical summary outputs are:

- `build/examples_runtime/regulatory/cross_pillar/scoped_-clients/southern_cross_cross_border/reference_pack_summary.csv`
- `build/examples_runtime/regulatory/cross_pillar/scoped_-clients/harbour_bridge_multientity/reference_pack_summary.csv`

The scope model is explicit in the manifest and is one of the main reasons this directory is useful for reviewers. The manifest first declares the cross-domain group perimeter in `shared_scope`. Each scenario then binds that shared population to one regulatory view by selecting a bundle, a regulator code and, where required, a jurisdiction slice. In the examples the member identifiers such as `LE_CVA_A` or `LE_CREDITRISK` are stable legal-entity keys. In a production estate these would normally be mapped to the firm's governed legal-entity master and ultimately to the relevant LEI population used in regulatory reporting.

The two maintained manifests illustrate two different scope patterns.

- `southern_cross_cross_border_reference_pack.yaml` declares one shared Basel-style cross-border group with members `LE_CVA_A`, `LE_CVA_B` and `LE_CVA_C`. Individual scenarios then narrow that population to a concrete reporting scope such as a PRA single-entity CCR run or an EU single-entity IMM run.
- `harbour_bridge_multientity_reference_pack.yaml` declares a wider shared perimeter with members `LE_CREDITRISK`, `LE_MARKETRISK`, `LE_CVA_A`, `LE_CVA_B`, `LE_CVA_C`, `LE_MR_A` and `LE_MR_B`. Each scenario then declares both a regulator-specific jurisdiction slice and the final legal-entity job that is executed inside that slice.

```yaml
reference_reporting_pack:
  pack_id: "southern_cross_cross_border"
  shared_scope:
    scope_id: "SCOPE__BASEL__CCR_CVA_MIXED__GROUP"
    scope_version: "2026Q1"
    member_legal_entity_ids:
      - "LE_CVA_A"
      - "LE_CVA_B"
      - "LE_CVA_C"
  scenarios:
    - scenario_id: "pra_uk_entity_ccr_ccp"
```

```
      regulator_code: "PRA"
      bundle: "examples/regulatory/cross_pillar/scoped_clients/southern_cross_cross_border/bundles/pra_uk_entity_ccr_ccp.toml
      metrics:
        - metric_id: "pra_qccp_trade_rwea"
          report_path: "build/examples_runtime/regulatory/cross_pillar/scoped_clients/southern_cross_cross_border/pra_uk_enti
          extract:
            kind: "row_match"
            match_column: "TemplateRowCode"
            match_value: "0020"
            value_column: "RiskWeightedExposureAmount"
```

```
reference_reporting_pack:
  pack_id: "harbour_bridge_multientity"
  shared_scope:
    scope_id: "SCOPE__REFERENCE__HBBG__CROSS_BORDER_GROUP"
    member_legal_entity_ids:
      - "LE_CREDITRISK"
      - "LE_MARKETRISK"
      - "LE_CVA_A"
      - "LE_CVA_B"
      - "LE_CVA_C"
      - "LE_MR_A"
      - "LE_MR_B"
  scenarios:
    - scenario_id: "pra_le_creditrisk_ccr_ccp"
      regulator_code: "PRA"
      jurisdiction_slice_id: "SCOPE__PRA__REFERENCE__HBBG__GROUP"
      jurisdiction_slice_member_legal_entity_ids:
        - "LE_CREDITRISK"
        - "LE_MARKETRISK"
        - "LE_CVA_B"
```

This distinction between `shared_scope`, `jurisdiction_slice` and the final executed job matters operationally. The shared scope is the declared cross-border perimeter for the pack. The jurisdiction slice is the regulator's subgroup inside that perimeter. The bundle's own `scope_id` is the exact run scope sent to VRE for the job that generates the report files. The summary artifacts written under `build/examples_runtime/.../reference_pack_summary.csv` preserve all three views so that a reviewer can check whether a number is being shown as a group metric, a jurisdiction subgroup metric or a single legal-entity metric.

The mapping back to regulation is equally explicit. Each reporting scenario points to a maintained run bundle whose `[run]` block records the regulatory rulebook and the reporting regime. For the PRA QCCP example the bundle is:

```
[run]
source_bundle = "examples/regulatory/cross_pillar/scoped_clients/southern_cross_cross_border/bundles/pra_uk_entity_ccr_ccp.to
client_id = "VRE_EXAMPLES"
scope_id = "SCOPE__PRA__CCR_CCP__LE_CREDITRISK"
scope_version = "2026Q1"
rulebook_id = "PRA__SACCR_CCP"
rulebook_version = "2026Q1"
config_set_id = "CFG__REPORTING_REFERENCE__SCBG__PRA_UK_CCR_CCP"
config_set_version = "2026Q1"
reporting_id = "PRA"
results_path = "build/examples_runtime/regulatory/cross_pillar/scoped_clients/southern_cross_cross_border/pra_uk_entity_ccr_c

[[jobs]]
job_id = "job_reference_pra_ccr_ccp"
flow_id = "CcrRiskFlow"
analytics = ["SACCR"]
```

This means that the reporting pack does not invent a second reporting engine. It reuses a maintained domain bundle, runs VRE for the declared regulatory scope, and then extracts the

report values required by the manifest. The report mapping can therefore be read as a short chain:

manifest scenario → bundle scope + rulebook → report file → row/column extraction → pack summary

A concrete PRA worked example is the scenario pra_uk_entity_ccr_ccp in the Southern Cross pack.

1. The manifest assigns this scenario to the PRA view, points it to bundles/pra_uk_entity_ccr_ccp.toml, and states that the metric of interest is pra_qccp_trade_rwea.
2. The bundle binds the job to SCOPE__PRA__CCR_CCP__LE_CREDITRISK with rulebook_id = PRA__SACCR_CCP. This is the exact declaration that the run should be interpreted as a PRA SA-CCR CCP capital slice for one legal entity.
3. The manifest then points to the resulting report pra_c34_10_01_01_exposures_to_ccps.csv and says how to read it: find TemplateRowCode = 0020 and take the RiskWeightedExposureAmount column. This is the bridge from engine output back to the regulatory template row.
4. The pack summary records the extracted number together with the source scope and bundle provenance, e.g.

```
scenario_id,pra_uk_entity_ccr_ccp
regulator_code,PRA
shared_scope_id,SCOPE__BASEL__CCR_CVA_MIXED__GROUP
scope_id,SCOPE__PRA__CCR_CCP__LE_CREDITRISK
report_file,build/examples_runtime/regulatory/cross_pillar/scoped_clients/southern_cross_cross_border/pra_uk_entity_ccr_ccp/p
metric_id,pra_qccp_trade_rwea
metric_value,67.46593792
job_id,job_reference_pra_ccr_ccp
bundle_path,examples/regulatory/cross_pillar/scoped_clients/southern_cross_cross_border/bundles/pra_uk_entity_ccr_ccp.toml
```

This is the essential reporting pattern used throughout the directory. A pack declares the institutional perimeter, narrows that perimeter to a jurisdiction and entity view, runs the relevant maintained VRE job, and then maps specific rows or aggregates back into a summary artifact that a reviewer can reconcile directly to the underlying report file.

Operationally, this directory sits between the domain examples and the broader scope/reporting program described elsewhere in the guide: the domain folders own the model-specific packs, while examples/regulatory/cross_pillar/scoped_clients owns the cross-domain orchestration, pack manifests and reviewer-facing summaries.

Current pack coverage is Basel, EU, PRA, US, SG and AU. APRA/AU coverage is no longer deferred in the example layer: authoritative APRA reviewer goldens live under examples/common/expected_output/reporting/apra, while governed reference-pack summaries live under examples/regulatory/cross_pillar/scoped_clients/.../expected_output. The APRA reviewer estate covers adjusted CEM, APRA CVA, APS 116 standard, APS 116 internal, and approved mixed use.

9.7 Market Risk

This directory demonstrates market-risk analytics and market-risk capital workflows. The non-capital analytics are complemented by regulatory SA-MR and IMA examples that align with MAR and DIS evidence outputs.

Run all default cases with `python run.py`. The run profile can be selected via `VRE_EXAMPLES_E2E_PROFILE` (`lean` or `full`).

The default collection includes:

- Sensitivity analysis in the "raw" and "par" domain: `python run_sensi.py`
- Sensitivity analysis when pricing with smile: `python run_sensismile.py`
- Parametric VaR: `python run_parametricvar.py`
- Stress testing in the "raw" domain: `python run_stress.py`
- Stress testing in the "par" domain: `python run_parstress.py`
- Stressed Sensitivity analysis: `python run_sensistress.py`
- Historical Simulation VaR: `python run_histsimvar.py`
- SA-MR Simplified: `python run_simplifiedmr_deltaplus.py` and `python run_simplifiedmr_scenario.py`
- SA-MR SBM: `python run_marketrisk_samr.py`
- IMA Pure: `python run_marketrisk_ima_pure.py`
- IMA/SA-MR mixed desk and backend parity: `python run_marketrisk_ima_samr_mixed.py` and `python run_marketrisk_ima_samr_parity.py`
- P&L and P&L Explanation: `python run_pnlexplain.py`
- Par conversion utility: `python run_parconversion.py`
- Base scenario utility: `python run_basescenario.py`
- Zero to par shift utility: `python run_zerotoparshift.py`

9.7.1 Sensitivity Analysis

This example (`python run_sensi.py`) demonstrates the calculation of sensitivities for a portfolio consisting of

- a vanilla swap in EUR
- a cross currency swap EUR-USD
- a resettable cross currency swap EUR-USD
- a FX forward EUR-USD
- a FX call option on USD/GBP
- a FX put option on USD/EUR

- an European swaption
- a Bermudan swaption
- a cap and a floor in USD
- a cap and a floor in EUR
- a fixed rate bond
- a floating rate bond with floor
- an Equity call option, put option and forward on S&P500
- an Equity call option, put option and forward on Lufthansa
- a CPI Swap referencing UKRPI
- a Year-on-Year inflation swap referencing EUHICPXT
- a USD CDS.

The maintained sensitivity run is authored in `examples/regulatory/pillar_1/market_risk/ima/input/vre_sensi.toml`. Its analytic block references the source-native scenario-market and sensitivity contracts that live beside the launcher. The resulting sensitivity configuration aims at computing the following sensitivities

- discount curve sensitivities in EUR, USD; GBP, CHF, JPY, on pillars 6M, 1Y, 2Y, 3Y, 5Y, 7Y, 10Y, 15Y, 20Y (absolute shift of 0.0001)
- forward curve sensitivities for EUR-EURIBOR 6M and 3M indices, EUR-EONIA, USD-LIBOR 3M and 6M, GBP-LIBOR 3M and 6M, CHF-LIBOR-6M and JPY-LIBOR-6M indices (absolute shift of 0.0001)
- yield curve shifts for a bond benchmark curve in EUR (absolute shift of 0.0001)
- FX spot sensitivities for USD, GBP, CHF, JPY against EUR as the base currency (relative shift of 0.01)
- FX vegas for USDEUR, GBPEUR, JPYEUR volatility surfaces (relative shift of 0.01)
- swaption vegas for the EUR surface on expiries 1Y, 5Y, 7Y, 10Y and underlying terms 1Y, 5Y, 10Y (relative shift of 0.01)
- caplet vegas for EUR and USD on an expiry grid 1Y, 2Y, 3Y, 5Y, 7Y, 10Y and strikes 0.01, 0.02, 0.03, 0.04, 0.05. (absolute shift of 0.0001)
- credit curve sensitivities on tenors 6M, 1Y, 2Y, 5Y, 10Y (absolute shift of 0.0001).
- Equity spots for S&P500 and Lufthansa
- Equity vegas for S&P500 and Lufthansa at expiries 6M, 1Y, 2Y, 3Y, 5Y
- Zero inflation curve deltas for UKRPI and EUHICPXT at tenors 6M, 1Y, 2Y, 3Y, 5Y, 7Y, 10Y, 15Y, 20Y
- Year on year inflation curve deltas for EUHICPXT at tenors 6M, 1Y, 2Y, 3Y, 5Y, 7Y, 10Y, 15Y, 20Y

Furthermore, mixed second order derivatives ("cross gammas") are computed for discount-discount, discount-forward and forward-forward curves in EUR.

The sensitivities are first computed in the "raw" (e.g. zero rate and optionlet) domain. The "raw" sensitivity analysis produces three output files.

The first, `scenario.csv`, contains the shift direction (`UP`, `DOWN`, `CROSS`), the base NPV, the scenario NPV and the difference of these two for each trade and sensitivity key. For an overview over the possible scenario keys see 12.4.

The second file, `sensitivity.csv`, contains the shift size (in absolute terms always) and first ("Delta") and second ("Gamma") order finite differences computed from the scenario results. Note that the Delta and Gamma results are pure differences, i.e. they are not divided by the shift size.

The second file also contains second order mixed differences according to the specified cross gamma filter, along with the shift sizes for the two factors involved.

Raw sensitivities are then converted into the "par" domain (e.g. Swap rates, CDS spreads) via a Jacobi transformation. See a sketch of the methodology in [2] and section 12.4 for configuration details.

To perform a par sensitivity analysis, the maintained run bundle enables the sensitivity analytic as follows:

```
[analytics.sensitivity]
active = "Y"
marketConfigFile = "simulation.xml"
sensitivityConfigFile = "sensitivity.xml"
pricingEnginesFile = "pricingengine.xml"
scenarioOutputFile = "scenario.csv"
sensitivityOutputFile = "sensitivity.csv"
outputSensitivityThreshold = "0.000001"
recalibrateModels = "Y"
parSensitivity = "Y"
optimiseRiskFactors = "Y"
parSensitivityOutputFile = "parsensitivity.csv"
outputJacobi = "Y"
jacobiOutputFile = "jacobi.csv"
jacobiInverseOutputFile = "jacobi_inverse.csv"
```

The usual "raw" sensitivity analysis is performed by bumping the "raw" rates (zero rates, hazard rates, inflation zero rates, optionlet vols). This is followed by the Jacobi transformation that turns "raw" sensitivities into sensitivities in the par domain (Deposit/FRA/Swap rates, FX Forwards, CC Basis Swap spreads, CDS spreads, ZC and YOY Inflation Swap rates, flat Cap/Floor vols). In the current surface the conversion rules are part of the published sensitivity contract referenced by the maintained `sensitivity.xml`. That contract carries the same logical information as the historical `ParConversion` blocks: shift tenors, shift types, assumed par instruments, and the convention identifiers used to rebuild the Jacobian in a reproducible way.

Finally note that par sensitivity analysis requires that the shift tenor grid in the sensitivity data above matches the corresponding grid in the simulation (market) configuration. See also section 12.4.

9.7.2 Sensitivity Analysis and Pricing with Smile

This example (`python run_sensismile.py`) demonstrates the current state of sensitivity calculation in VRE for European options where the volatility surface has a smile.

The portfolio used in this example consists of

- an equity call option denominated in USD ("SP5")
- an equity put option denominated in USD ("SP5")
- a receiver swaption in EUR
- an FX call option on EUR/USD

Refer to [2] for the current status of sensitivity implementation with smile. In this example the setup is as follows

- today's market is configured with volatility smile for all three products above
- simulation market has two configurations, to simulate "ATM only" or the "full surface"; "ATM only" means that only ATM volatilities are to be simulated and shifts to ATM vols are propagated to the respective smile section;
- the sensitivity analysis has two corresponding configurations as well, "ATM only" and "full surface"; note that the "full surface" configuration leads to explicit sensitivities by strike only in the case of Swaption volatilities, for FX and Equity volatilities only ATM sensitivity can be specified at the moment and sensitivity output is currently aggregated to the ATM bucket (to be extended in subsequent releases).

The respective output files end with "`_fullSurface.csv`" respectively "`_atmOnly.csv`".

9.7.3 Parametric VaR

This example (`python run_parametricvar.py`) demonstrates a parametric VaR calculation based on the sensitivity and cross gamma output from the sensitivity analysis (deltas, vegas, gammas, cross gammas) and an external covariance matrix input. The result in `var.csv` shows a breakdown by portfolio, risk class (All, Interest Rate, FX, Inflation, Equity, Credit) and risk type (All, Delta & Gamma, Vega). The results shown are Delta Gamma Normal VaRs for the 95% and 99% quantile, the holding period is incorporated into the input covariances. Alternatively, one can choose a Monte Carlo VaR which means that the sensitivity based P&L distribution is evaluated with MC simulation assuming normal respectively log-normal risk factor distribution.

9.7.4 Stress Testing

This example (`python run_stress.py`) uses the same portfolio as the sensitivity analysis in 9.7.1. The published stress pack defines two stress tests:

- `parallel_rates`: Rates are shifted in parallel by 0.01 (absolute). The EUR bond benchmark curve is shifted by increasing amounts 0.001, ..., 0.009 on the pillars 6M, ..., 20Y. FX Spots are shifted by 0.01 (relative), FX vols by 0.1 (relative), swaption and cap floor vols by 0.0010 (absolute). Credit curves are not yet shifted.

- `twist`: The EUR bond benchmark curve is shifted by amounts -0.0050, -0.0040, -0.0030, -0.0020, 0.0020, 0.0040, 0.0060, 0.0080, 0.0100 on pillars 6M, 1Y, 2Y, 3Y, 5Y, 7Y, 10Y, 15Y, 20Y.

The corresponding output file `stresstest.csv` contains the base NPV, the NPV under the scenario shifts and the difference of the two for each trade and scenario label.

9.7.5 Stress Testing in the Par Domain

The stress test in 9.7.4 is performed in the "raw" domain of zero rate shifts, hazard rate shifts, optionlet volatility shifts etc. To analyse the impact of market rate shifts (Swap rates, CDS spreads, flat vols), one would have to manipulate the market data input into VRE and re-run the entire VRE process multiple times.

This example (`python run_parstress.py`) demonstrates the extended stress testing framework that operates in the "par" rate domain in a configurable way i.e without the user's manipulation of the input market data.

9.7.6 Stressed Sensitivity Analysis

This example (`python run_sensistress.py`) demonstrates the stressed sensitivity analysis. The new analytic type *SENSITIVITY_STRESS* utilizes the existing stresstest framework, applies stress scenarios to T0 market and runs sensitivity analysis. The Stresstest scenarios are given in the same input format as for the regular stresstest. Dependent sensitivity analysis is currently supported only in zero domain for all asset classes.

The stressed sensitivity analytic replaces the todaysMarket with a SimulationMarket during the sensitivity calculation. For some risk factors the simulationMarket behaves different as the todaysMarket. Depending on the simulation and stress scenario settings it could use different tenors when building curves or use only the ATM volatilities. It is recommended to activate UseSpreadedTermStructures and simulate SwaptionVolatilities for the stressed sensitivity run.

9.7.7 Historical Simulation VaR

This example (`python run_histsimvar.py`) demonstrates a historical simulation VaR calculation given a portfolio and externally provided "market scenarios" covering one or several historical observation period(s). The analytic is specified in the current run bundle with the following parameters:

- outputFile: csv file name of the resulting VaR report
- tradePnl: boolean, if true the VaR report will contain a breakdown by tradeID, risk class and risk type, otherwise the report shows the portfolio-lvel VaR only.
- quantiles: comma separated list of quantiles to be reported
- portfolioFilter (optional): Only trades with `portfolioId` equal to the provided filter name are processed, see the maintained portfolio payload; the entire portfolio is processed, if omitted
- historicalPeriod: comma-separated date list, an even number of ordered dates is required (d1, d2, d3, d4, ...), where each pair (d1-d2, d3-d4, ...) defines the start and

end of historical observation periods used

- mporDays: Number of calendar days between historical scenarios taken from the observation periods in order to compute P&L effects (typically 1 or 10)
- mporCalendar: Calendar applied in the scenario date calculation
- mporOverlappingPeriods: Boolean, if true we use overlapping periods of length mporDays (t to t + 10 calendar days, t+1 to t+11, t+2 to t+12, ...), otherwise consecutive periods (t to t+10, t+10 to t+20, ...)
- simulationConfigFile: defines the structure of the simulation market applied in the P&L calculation, e.g. discount and index curves, yield curve tenor points used, FX pairs etc.
- historicalScenarioFile: csv file containing the market scenarios for each date in the observation periods defined below; the granularity of the scenarios (e.g. discount and index curves, number of yield curve tenors) needs to match the simulation market definition above; each yield curve tenor scenario is represented as a discount factor

9.7.8 SA-MR Simplified Market Risk Capital

This example runs the simplified SA-MR market risk capital workflow on the bundled multi-asset trading-book portfolio. Execute it via the helper scripts: `python run_simplifiedmr_deltaplus.py` (DeltaPlus) or `python run_simplifiedmr_scenario.py` (Scenario).

The maintained launch surface is `examples/regulatory/pillar_1/market_risk/simplified/input/vre_simplifiedmr.toml`. Its setup block binds the run to `input_native/vre_simplifiedmr`, the bundled market snapshot `samr_market_20160205.txt`, the compiled pricing policy, the trading desk registry, and the published sensitivity / simulation contracts. The portfolio itself remains a trade-payload exception and is carried in `samr_portfolio.xml`; it contains rates, FX, equity, credit-spread and commodity positions so that the simplified capital methods can be exercised across the main SA-MR supervisory risk classes.

This is a maintained regulatory example, not a reduced pedagogical toy. The native bundle controls the run, but the portfolio remains deliberately broad so that desk routing, supervisory bucket selection and option-treatment choices are all visible in the resulting capital outputs.

Analytically, the two runner variants demonstrate the two maintained simplified option treatments:

- `DeltaPlus`: option risk is represented through delta, gamma and vega style add-ons built from the maintained market-risk configuration;
- `Scenario`: option risk is represented through the simplified supervisory scenario grid, with worst-loss selection on the generated scenario surface.

Outputs are written to `output/market_risk/SimplifiedMR` and the runner specific output folders, each with a capital report `simplifiedmr_capital.csv`. These runs are useful because they show the same desk and reference-data estate being processed under a lighter-weight capital method than the full SA-MR SBM examples below.

The output set should be read as a family. The headline file `simplifiedmr_capital.csv` gives the final charge under the selected DeltaPlus or Scenario option treatment, while the

companion runtime and log files preserve the exact deterministic launch context used to produce it.

9.7.9 SA-MR Sensitivities-Based Method (SBM)

The SA-MR SBM example is run with:

```
python run_marketrisk_samr.py
```

and writes output to

`examples/regulatory/pillar_1/market_risk/samr/expected_output/SAMR/SBM`, including `samr_capital.csv`.

The maintained authored entry point is `examples/regulatory/pillar_1/market_risk/samr/input/vre_marketrisk_samr.toml`. It binds the run to the resolved simulation and sensitivity contracts under `input_native/vre_marketrisk_samr`, uses the AMCCG pricing policy, and loads the current desk and reference-data registries. The bundled `samr_portfolio.xml` is deliberately mixed: rates, FX, equity, commodity and credit-spread positions appear together so that the bucket mapping, desk aggregation and evidence files reflect a realistic supervisory capital assembly rather than a one-risk-class smoke test.

The primary maintained runner now launches through the scope-native surface, so the example also demonstrates the current regulatory job model. In addition to capital output, the run emits the resolved run bundle, scope population, scope-resolution material, and run/job evidence JSON files next to the market risk results.

Additional focused variants include:

- credit spread examples: `python run_marketrisk_samr_with_cds.py` and `python run_marketrisk_samr_cds.py`
- DRC/RRAO coverage example: `python run_marketrisk_samr_drc_rrao.py`

The maintained output estate under `output/SAMR/SBM` is intentionally rich:

- `samr_capital.csv`: legal-entity and scenario level capital summary;
- `samr_desk_comparator.csv`: desk aggregation and approval-state comparison;
- `samr_trade_detail.csv`: trade-level contribution detail;
- `samr_buckets.csv`: supervisory bucket decomposition;
- `market_risk_method_context.csv`: backend and method-routing evidence.

9.7.10 IMA Pure

The pure IMA example is run with:

```
python run_marketrisk_ima_pure.py
```

using the input pack `examples/regulatory/pillar_1/market_risk/ima/input/MARKET_RISK_IMA_PURE`. This case demonstrates desk-level IMA capital assembly on a scripted multi-asset setup. Output is written to `build/examples_runtime/regulatory/pillar_1/market_risk/ima`.

The pure IMA case is materially richer than a single summary number. It routes a scripted multi-asset trading book through the IMA stack and writes not only the headline capital files

(ima_capital_summary.csv, ima_capital_components.csv) but also the supporting audit and control artefacts: modellability evidence, P&L attribution evidence, backtesting evidence, liquidity-horizon decomposition, scenario summaries, disclosure extracts and legal-entity reconciliation outputs. It is the maintained example to read when reviewing how VRE assembles a production-style IMA evidence pack.

9.7.11 IMA and SA-MR Mixed Desk Parity

The mixed-desk example combines IMA and SA-MR desk approval states and runs the market risk analytic on a shared portfolio:

```
python examples/regulatory/pillar_1/market_risk/ima/run.py
```

using the maintained mixed-desk source estate under examples/regulatory/pillar_1/market_risk/ima/input/MARKET_RISK_IMA_SAMR_MIXED. Runtime output is written under build/examples_runtime/regulatory/pillar_1/market_risk/ima.

Conceptually this example mirrors the real Basel III end-game operating problem: one shared trading-book population, some desks approved for IMA, the remainder on SA-MR, and a requirement to prove that backend choice does not move the regulatory answer. The mixed-desk run therefore writes both IMA evidence and SA-MR desk capital outputs over the same legal-entity estate.

Useful environment switches include:

- MARKETRISK_PARITY_INCLUDE_GPU: include GPU leg in parity run
- MARKETRISK_PARITY_ENFORCE_FD: enforce full FD/AAD/GPU triangle parity
- MARKETRISK_PARITY_SAMPLES: override scenario sample count for deterministic parity smoke runs

In this chapter we use the following naming convention:

- FD: finite-difference bump/revalue baseline
- AAD: adjoint/CG path
- GPU: external compute device path where available

9.7.12 Multi-Entity Regulatory Scope (SA-MR)

The integrated multi-entity SA-MR scope example runs two regulatory jobs over a shared market-risk pack, one per legal entity:

```
python run_marketrisk_multi_entity_integrated.py
```

Input configurations:

- input/vre_marketrisk_samr_external_multi_le_a.toml
- input/vre_marketrisk_samr_external_multi_le_b.toml

Expected outputs are checked in at:

- examples/regulatory/pillar_1/market_risk/samr/expected_output/MULTI_ENTITY/LE_A/samr_capital.csv

- examples/regulatory/pillar_1/market_risk/samr/expected_output/MULTI_ENTITY/LE_A/samr_desk_comparator.csv
- examples/regulatory/pillar_1/market_risk/samr/expected_output/MULTI_ENTITY/LE_B/samr_capital.csv
- examples/regulatory/pillar_1/market_risk/samr/expected_output/MULTI_ENTITY/LE_B/samr_desk_comparator.csv

The importance of this example is architectural as much as numerical. It shows that one shared market-risk source population can be routed into separate legal-entity jobs without cloning the portfolio, market or desk-definition estate, which is the same fan-out pattern used by current scope/reporting workflows and by downstream RaaS orchestration.

9.7.13 P&L and P&L Explain

This example (python run_pnlexplain.py) demonstrates the P&L and P&L explain analytics on a very simple test portfolio that consists of two single-leg swaps. Main output is the P&L report in examples/regulatory/pillar_1/market_risk/samr/expected_output/Pnl/pnl.csv with the following columns

- TradeId
- Maturity and MaturityTime
- StartDate and EndDate of the P&L period, referred to as t0 and t1 below
- NPV(t0)
- NPV(asof=t0; mkt=t1)
- NPV(asof=t1; mkt=t0)
- NPV(t1)
- PeriodCashFlow: Aggregate of trade flows in the period, converted into the P&L currency below
- Theta: NPV(asof=t1; mkt=t0) - NPV(t0) + PeriodCashFLow
- HypotheticalCleanPnL: NPV(asof=t0; mkt=t1) - NPV(t0)
- CleanPnL: NPV(t1) - NPV(t0) + PeriodCashFlow
- DirtyPnL: NPV(t1) - NPV(t0)
- Currency

Moreover we write

- Four “flavours” of NPV reports used here
- Four related additional results reports
- Two reports for the market scenarios used in the two “lagged” NPV calculations

The second batch included in this example explains the P&L above in terms of portfolio sensitivities and changes in related market moves. The main output of this is in

`examples/regulatory/pillar_1/market_risk/samr/expected_output/PnlExplain/pnl_explain.csv`. The PnlExplain analytic contains the Pnl analytic as dependent analytic, i.e. the PnlExplain analytic is self-sufficient kicking off Pnl calculation internally. The only additional maintained analytic input for the explainer run is the source-native sensitivity contract referenced from `examples/regulatory/pillar_1/market_risk/ima/input/vre_pnlexplain.toml`.

9.7.14 Stand-alone Par Conversion Utility

This example (`python run_parconversion.py`) demonstrates VRE's capability to convert externally computed zero sensitivities (for example zero rates) to par sensitivities (for example swap rates) by means of a Jacobi transformation of the raw sensitivities. See [2] and section 12.4 for configuration details.

The maintained run bundle for this utility is `examples/regulatory/pillar_1/market_risk/ima/input/vre_parconversion.toml`. Its analytic block is:

```
[analytics.zeroToParSensiConversion]
active = "Y"
marketConfigFile = "simulation.xml"
sensitivityConfigFile = "sensitivity.xml"
pricingEnginesFile = "../../../common/input/shared_runtime/pricingengine.xml"
sensitivityInputFile = "sensitivity.csv"
outputThreshold = "0.000001"
outputFile = "parconversion_sensitivity.csv"
outputJacobi = "Y"
jacobiOutputFile = "jacobi.csv"
jacobiInverseOutputFile = "jacobi_inverse.csv"
```

The portfolio used in this example includes zero sensitivities of

- Discount and index curves
- Credit curves
- Inflation curves
- CapFloor volatilities

VRE reads the raw sensitivities from the csv input file *sensitivityInputFile*. The input file needs to have six columns, the column names can be user configured. Here is a description of each of the columns:

1. idColumn : Column with a unique identifier for the trade / nettingset / portfolio.
2. riskFactorColumn: Column with the identifier of the zero/raw sensitivity. The risk factor name needs to follow the VRE naming convention, e.g. DiscountCurve/EUR/5/1Y (the 6th bucket in the EUR discount-curve sensitivity grid specified by the maintained sensitivity contract)
3. deltaColumn: The raw sensitivity of the trade/nettingset / portfolio with respect to the risk factor
4. currencyColumn: The currency in which the raw sensitivity is expressed, need to be the same as the BaseCurrency in the simulation settings.
5. shiftSizeColumn: The shift size applied to compute the raw sensitivity, need to be consistent to the sensitivity configuration.

6. baseNpvColumn: The base npv of the trade / nettingset / portfolio in currency.

This is followed by the Jacobi transformation that turns raw sensitivities into sensitivities in the par domain (Deposit/FRA/Swap rates, FX Forwards, CC Basis Swap spreads, CDS spreads, ZC and YOY Inflation Swap rates, flat Cap/Floor vols). In the current surface the conversion rules are carried by the same published sensitivity contract that drives the standard sensitivity run: shift-tenor grids, shift types, assumed par instruments, and convention identifiers are all resolved before the utility runs, so the zero-to-par conversion is reproducible and reviewable from the native bundle.

Finally note that par sensitivity analysis requires that the shift tenor grid in the sensitivity data above matches the corresponding grid in the simulation (market) configuration. See also section 12.4.

9.7.15 Base Scenario Utility

This example (`python run_basescenario.py`) demonstrates the `Scenario` analytic which has been added to export the simulation market's base scenario as a file. This analytic is e.g. used internally in P&L analytics 9.7.13, and it can be used to extract historical scenarios for the HistSim Var 9.7.7.

9.7.16 Zero Shift to Par Shift Conversion Utility

This example (`python run_zerotoparshift.py`) demonstrates the conversion of zero shifts to par rate shifts. VRE applies the zero rate shifts to the zero curves and computes the resulting shifts in the implied fair rate of a given set of par instruments. The zero rate shifts are defined as stresstests and the par instruments are defined in the usual sensitivity configuration. This analytic is used internally in the par-sensitivity-based P&L explainer.

9.8 CVA Risk

The examples in `examples/regulatory/pillar_1/credit/cva` document regulatory CVA capital workflows and are the primary location for SA-CVA and BA-CVA examples in this release.

Run the default suite with:

```
cd examples/regulatory/pillar_1/credit/cva && python run.py
```

or run individual batches as shown below.

9.8.1 BA-CVA Variants

Run BA-CVA variants with:

```
python run_bacva.py
```

This script runs reduced and full BA-CVA cases, including larger-portfolio and IMM-linked variants:

- reduced BA-CVA: `input/vre_bacva_reduced.toml`
- full BA-CVA: `input/vre_bacva_full.toml`
- full BA-CVA (larger portfolio): `input/vre_bacva_full_largp.toml`

- full BA-CVA (IMM EEPE input): `input/vre_bacva_full_imm.toml`
- full BA-CVA (IMM MA grouping): `input/vre_bacva_full_imm_ma_group.toml`
- full BA-CVA (IMM MPOR/CCP/WWR overlay): `input/vre_bacva_full_imm_mpor_ccp.toml`

The configuration chain combines CCR exposure input (`saccr` or `imm`) with `sacva` in regulatory execution mode (`cvaExecutionMode=Regulatory`). The underlying portfolios are deliberately richer than the small smoke examples: they combine IRS, FX, equity and commodity-style exposure sources, CSA state, counterparty attributes and legal-entity overlays so that BA-CVA aggregation is driven by realistic counterparty groupings rather than by one isolated trade. The IMM-linked variants are especially important because they demonstrate the handoff from simulated exposure production into BA-CVA capital without changing the surrounding reporting and scope controls. Primary output is `bacva.csv` in the corresponding output directories (`examples/regulatory/pillar_1/credit/cva/ba_cva/expected_output/bacva_*`).

The maintained BA-CVA output estate is broader than the headline capital CSV. The runs also emit routing, perimeter, eligibility and RWA evidence such as `bacva_routing.csv`, `cva_capital_facts.csv`, `cva_capital_perimeter_resolution_facts.csv`, `cva_eligibility_facts.csv`, and `cva_rwa_facts.csv`. The SA-CCR-fed variants produce the underlying CCR evidence and DIS42 disclosures, while the IMM-fed variants add `imm_exposure.csv`, `imm_eepe_summary.csv`, selection-evidence files and, where relevant, CCP qualification resolution facts. This makes the example suitable for full regulatory evidence tracing.

9.8.2 SA-CVA Variants

Run the core SA-CVA batch with:

```
python run_sacva.py
```

This covers:

- AMC/flat reference: `input/vre_sacva_flat.toml`
- SA-CCR-CVA profile path: `input/vre_sacva_saccr_cva.toml`
- SA-CCR-CVA all-asset scripted case: `input/vre_sacva_saccr_all_asset.toml`
- IMM/CG-AAD case: `input/vre_sacva_cg_ad.toml`
- optional IMM/CG-GPU case: `input/vre_sacva_cg_gpu.toml`

The SA-CVA suite is one of the most important maintained native example families. The authored run bundles activate a regulatory execution mode, resolved counterparty/netting/legal-entity publications, and either SA-CCR or IMM exposure production depending on the chosen path. The portfolios are multi-asset and intentionally supervisory in flavour: IRS, FX, equity and commodity exposures appear together so that bucket mapping, hedge recognition, trade-level sensitivities and capital decomposition are all exercised in one run. The compiled AAD / GPU variants are not cosmetic duplicates; they are the performance and parity harnesses for the same regulatory workflow.

The portfolio design is deliberate. The SA-CCR-CVA route demonstrates the lighter-weight supervisory path with explicit CCR exposure assembly, while the IMM-linked CG/AAD and

GPU paths demonstrate compiled exposure production and compiled sensitivity generation over the same regulatory aggregation semantics. Together the examples cover the full maintained SA-CVA operating envelope: SA-CCR-fed capital, IMM-fed capital, backend parity and external-device acceleration.

Enable the optional GPU leg via:

```
VRE_EXAMPLES_INCLUDE_GPU=1 python run_sacva.py
```

Primary outputs are `sacva.csv`, `sacva_sensitivities.csv` and, where applicable, SACCR-CVA sensitivity files (e.g. `cva_sensitivities.csv`, `cva_sensitivities_saccr_raw.csv`). For disclosure-oriented runs, DIS51 projection files are also generated (e.g. `dis51_cva3_sacva.csv`, `dis51_cva4_sacva.csv`).

The maintained output folders also include routing files, legal-entity scope files, capital-facts tables, exposure facts and backend-specific sensitivity metadata. In practice this means a reviewer can move from the final SA-CVA number to the underlying exposure method, scope decision and sensitivity source without leaving the example estate.

9.8.3 Multi-Entity Regulatory Scope (SA-CCR → SA-CVA)

The integrated multi-entity CCR→CVA scope example runs one regulatory CVA job per legal entity using canonical setup-level scope controls:

```
python run_sacva_multi_entity_integrated.py
```

Input configurations:

- `input/vre_sacva_saccr_multi_le_a.toml`
- `input/vre_sacva_saccr_multi_le_b.toml`

Expected outputs are checked in at:

- `examples/regulatory/pillar_1/credit/cva/sa_cva/expected_output/sacva_multi_entity_integrated/LE_A/sacva.csv`
- `examples/regulatory/pillar_1/credit/cva/sa_cva/expected_output/sacva_multi_entity_integrated/LE_A/sacva_routing.csv`
- `examples/regulatory/pillar_1/credit/cva/sa_cva/expected_output/sacva_multi_entity_integrated/LE_B/sacva.csv`
- `examples/regulatory/pillar_1/credit/cva/sa_cva/expected_output/sacva_multi_entity_integrated/LE_B/sacva_routing.csv`

9.8.4 SA-CVA Backend Parity (FD, AAD, GPU)

Run backend parity diagnostics with:

```
python run_sacva_backend_parity.py
```

This compares FD/Classic and CG/AAD runs and can optionally include CG/GPU:

- include GPU leg: `python run_sacva_backend_parity.py -include-gpu`

- require GPU availability: `-include-gpu -require-gpu`
- strict threshold enforcement: `-strict`

Parity diagnostics are written to:
`examples/regulatory/pillar_1/credit/cva/output/sacva_backend_parity/parity_diagnostics.json`

This workflow is used as an assurance control for backend consistency. It should be read as a validation harness, not as a replacement for default production run configuration.

This parity harness matters because the SA-CVA charge is sensitivity-driven. The goal is therefore not just performance comparison; it is controlled proof that FD, compiled AAD and external-device execution remain within agreed tolerances on the same supervisory portfolio and the same regulatory aggregation logic.

9.8.5 MAR50 Suite Wrappers

For smoke/full MAR50 wrapper runs use:

- `python run_mar50_regression_smoke.py`
- `python run_mar50_full.py`

These wrappers orchestrate BA-CVA and SA-CVA runs using deterministic sample-control defaults and are aligned with compliance evidence packaging in `VRECompliance`.

They are the correct top-level wrappers for regression and release evidence because they drive both reduced and richer capital paths under fixed, deterministic controls and produce a bounded but representative MAR50 evidence estate for CI and release review.

9.9 Initial Margin

This section covers two groups of batches

- ISDA SIMM and IM Schedule
- a Dynamic Initial Margin case study

discussed in the following subsection.

9.9.1 ISDA SIMM and IM Schedule

This example (`python run_simm.py`) demonstrates the calculation of initial margin using ISDA's Standard Initial Margin Model (SIMM) based on a provided sensitivity file in ISDA's Common Risk Interchange Format (CRIF). In addition, we show how to use the standard "IM Schedule" method to compute initial margin.

VRE covers all SIMM versions since inception to date, i.e. 1.0, 1.1, 1.2, 1.3, 1.3.38, 2.0, 2.1, 2.2, 2.3, 2.4 (=2.3.8), 2.5, 2.5A, 2.6 (=2.5.6). All versions have been tested against the respective ISDA SIMM model unit test suites and pass these tests. Any new SIMM versions will be added with each VRE release.

For SIMM versions >= 2.2 we support SIMM calculation for both MPoR horizons, 1d and 10d.

Note that you need to purchase a SIMM model license from ISDA if you want to use the model in production, and the unit test suites mentioned above are provided to licensed vendors only. Therefore we unfortunately cannot share our VRE SIMM model test suite here either.

By running

```
python run_simm.py
```

VRE will pick up the small example CRIF file in `examples/economic/initial_margin/input/crif.csv` (i.e. par sensitivities rebucketed and reformatted to match the ISDA CRIF template) and generate the resulting SIMM report in a `simm.csv` file. This report shows ISDA SIMM results with the usual breakdown by product class, risk class, margin type, bucket and SIMM "side" (IM to call or post). The SIMM calculation in this example is done for SIMM version 2.4 and 2.6, with MPoR 1d and 10d:

- SIMM 2.4, 1-day MPoR
- SIMM 2.4, 10-day MPoR
- SIMM 2.6, 1-day MPoR
- SIMM 2.6, 10-day MPoR

The maintained SIMM launch surface is a family of native run bundles under `examples/economic/initial_margin/input/Simm`. The standard packs are `vre_SIMM2.4_1D.toml`, `vre_SIMM2.4_10D.toml`, `vre_SIMM2.6_1D.toml`, `vre_SIMM2.6_10D.toml`, with the calibration-focused extensions `vre_SIMM2.5A_1D_calibration.toml` and `vre_SIMM2.5A_10D_calibration.toml`. The relevant inputs in these bundles are:

- SIMM version
- name of the CRIF file to be loaded
- calculation currency - this determines which Risk_FX entries of the CRIF will be ignored in the SIMM calculation
- result currency (optional) - currency of the resulting SIMM amounts in the report, by default equal to the calculation currency
- MPoR horizon, in terms of days

The market data input and todays's market configuration required here is minimal - limited to FX rates for conversions from base/calculation currency into USD and into the result currency.

IM Schedule

As an additional case in this example we demonstrate how to use the IM Schedule method to compute initial margin. The related maintained run bundle is `input/Simm/vre_schedule.toml`. It is also run when calling `python run.py`, and results are written to folder `examples/economic/initial_margin/output/IM_SCHEDULE`. The basic input is provided in CRIF file format where VRE expects two lines per trade, one with RiskClass = PV and one with RiskClass = Notional, so that the amounts in these CRIF lines are interpreted as NPV respectively notional. Further required columns are product class and

end date, as shown in the example `examples/economic/initial_margin/input/crif_schedule.csv`. Note that the product class has to be in

- Rates
- FX
- Equity
- Credit
- Commodity

in contrast to SIMM where we use the combined RatesFX.

To run the IM Schedule analytic, the current bundle exposes the following block in `input/Simm/vre_schedule.toml`.

```
[analytics.imschedule]
active = "Y"
crif = "crif_schedule.csv"
calculationCurrency = "USD"
```

9.9.2 Dynamic Initial Margin and MVA

This example (`python run_dim.py`) demonstrates Dynamic Initial Margin calculations (see also [2]) for a number of basic products:

- A single currency Swap in EUR (case A),
- a European Swaption in EUR with physical delivery (case B),
- a single currency Swap in USD (case C),
- a EUR/USD cross currency Swap (case D),
- a EUR/USD FX Option (case E).

The essential results of each run are visualised in the form of

- evolution of expected DIM which feeds into the MVA calculation
- regression plots at selected future times

illustrated for cases A, B and E in figures 3 - 7.

The DIM evolution graphs compare a subset of the following Initial Margin projection methods in VRE

- Simple: 99% quantile of NPV changes (Δ) over the Margin Period of Risk across all paths, i.e. same IM applied across paths
- Zero Order "Regression": Standard deviation of Δs scaled to the 99% quantile with factor 2.33 (normal distribution assumption); same IM applied across paths
- First/Second Order Regression: Conditional standard deviation of Δ computed by polynomial first/second order regression of Δ variances, scaled to the 99% quantile as

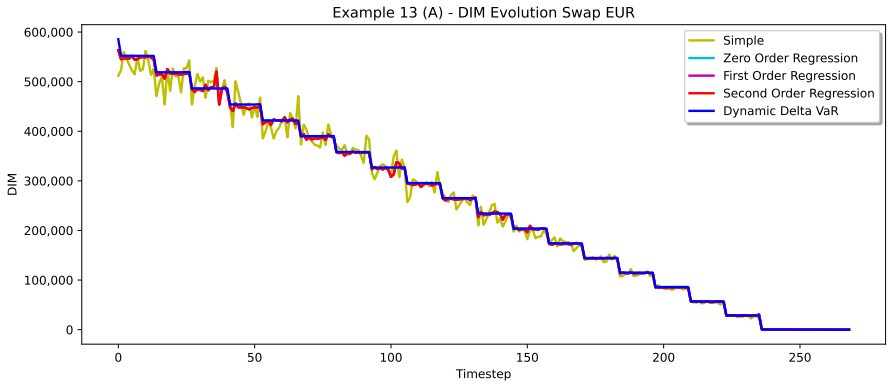

Figure 3: Evolution of expected Dynamic Initial Margin (DIM) for the EUR Swap of Example 13 A. Regression DIM is evaluated using regression of NPV change variances versus the simulated 3M Euribor fixing; regression polynomials are zero, first and second order (first and second order curves are not noticebly different in this case). The simulation uses 1000 samples and a time grid with bi-weekly steps in line with the Margin Period of Risk.

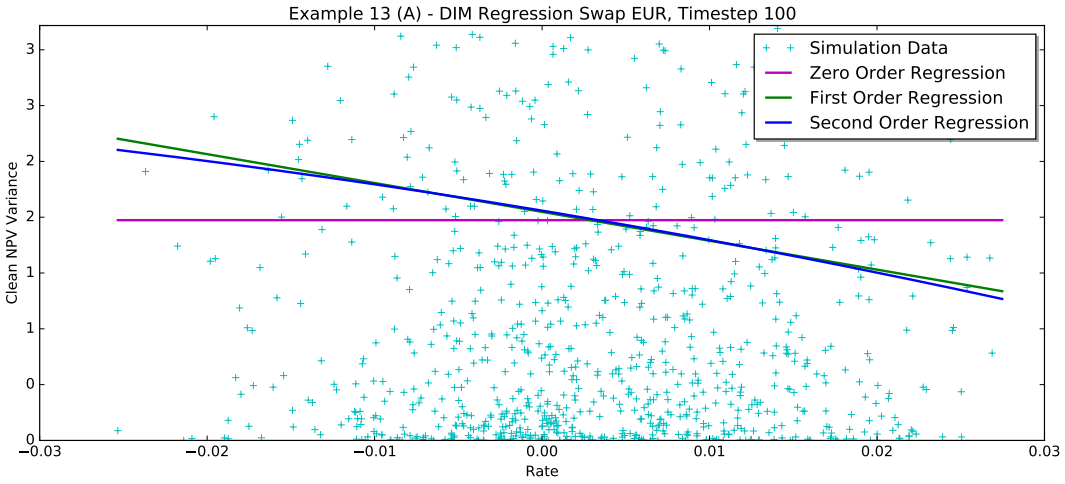

Figure 4: Regression snapshot at time step 100 for the EUR Swap of Example 13 A.

avove; different IM amounts applied across paths, graphs show the expected DIM i.e. avergae across paths

- Dynamic Delta VaR: 99% quantile VaR based on analytic deltas and vegas computed under scenarios; different IM amounts applied across paths, graphs show the expected DIM i.e. average across paths

For a discussion of the regression model performance see [16, 18]. The cases A–E are associated with the maintained native bundles in `examples/economic/initial_margin/input/Dim`, for example `vre_A0.toml`, `vre_B0.toml`, `vre_C1.toml`, `vre_D1.toml`, `vre_E0.toml`, which demonstrate the required simulation and XVA analytic configurations.

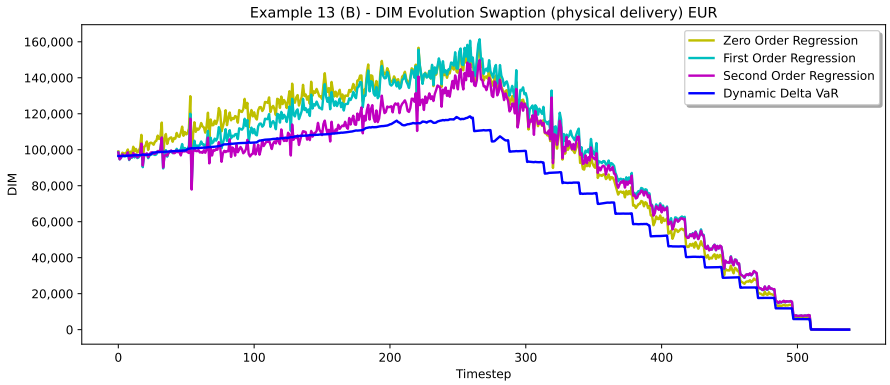

Figure 5: Evolution of expected Dynamic Initial Margin (DIM) for the EUR Swaption of Example 13 B with expiry in 10Y around time step 100.

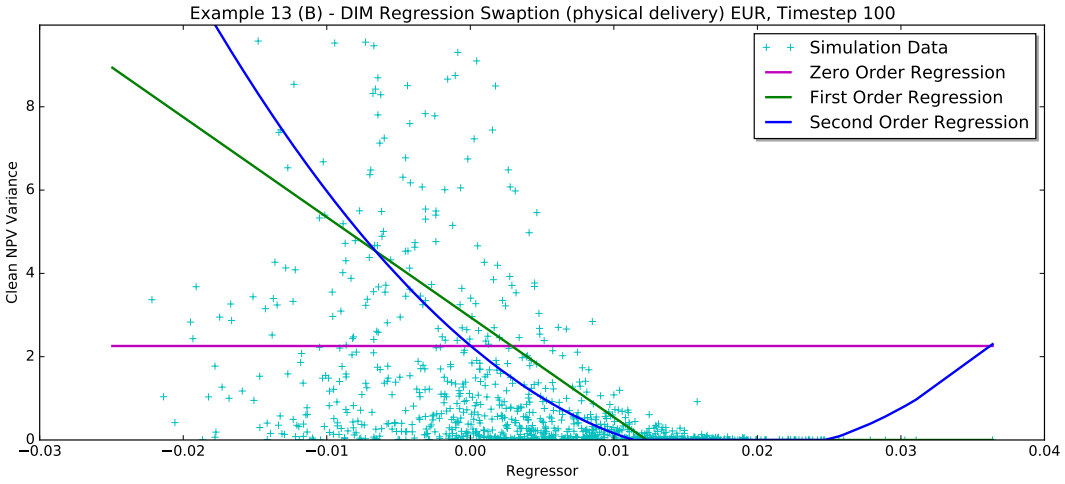

Figure 6: Regression snapshot at time step 100 (before expiry) for the EUR Swaption of Example 13 B.

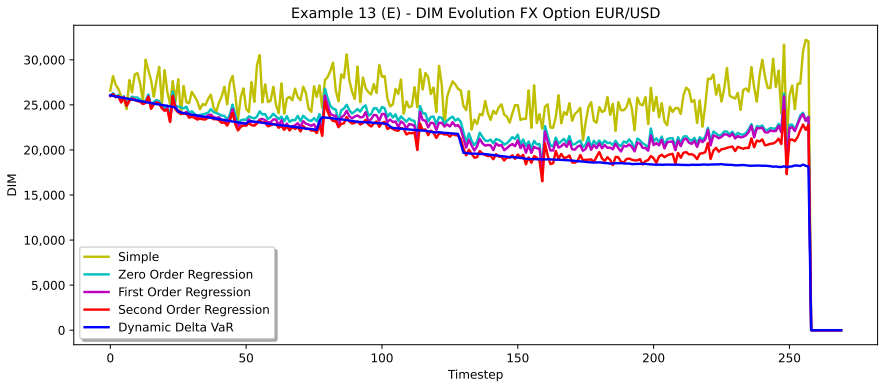

Figure 7: Evolution of expected Dynamic Initial Margin (DIM) for the EUR/USD FX Option of Example 13 (case E) with expiry in 10Y around time step 100.

9.9.3 Dynamic SIMM

The batch kicked off with (`python run_dim2.py`) demonstrates a prototype "Dynamic SIMM" implementation, see `input/Dim2/vre_amccg.toml` with

```
[analytics.xva]
active = "Y"
dim = "Y"
dimModel = "DynamicIM"
dimQuantile = "0.99"
dimHorizonCalendarDays = "14"
dimEvolutionFile = "dim_evolution.csv"
```

and several related compiled-CE controls in the same run bundle's `[analytics.simulation]` block. The new method is embedded into AMC simulation and uses Algorithmic Differentiation to generate sensitivities along paths which feed into the dynamic SIMM calculation. Results are – among others – a `dim_evolution.csv` report and a new `dim_distribution.csv` report in `examples/economic/initial_margin/output/Dim2`.

To validate the new method we apply the "conventional" SIMM calculator in VRE that we have enhanced with a CRIF generator limited to IR/FX risks for this purpose. The `run_dim2.py` script simulates a few paths, picks one of the paths and extracts the simulated market data (discount and index curves, FX rates, swaption volatilities) and writes them to `input/DimValidation/marketdata.csv`, a market data file in VRE format, and likewise simulated fixings to `input/DimValidation/fixings.csv`. The script then loops over simulation dates of that single path and performs a conventional SIMM run for each date (see `input/DimValidation/vre_simm.toml`). This is used to check the new method's output on individual paths: When switching off model calibration and setting model vols close to zero in the native simulation runtime contracts for `input_native/dim2__vre_classic/runtime_policy/simulation_runtime.toml` and `input_native/dim2__vre_amccg/runtime_policy/simulation_runtime.toml`, then we basically "roll down the forward curve", and both calculations should yield the same result. Figure 8 shows the comparison and expected outcome.

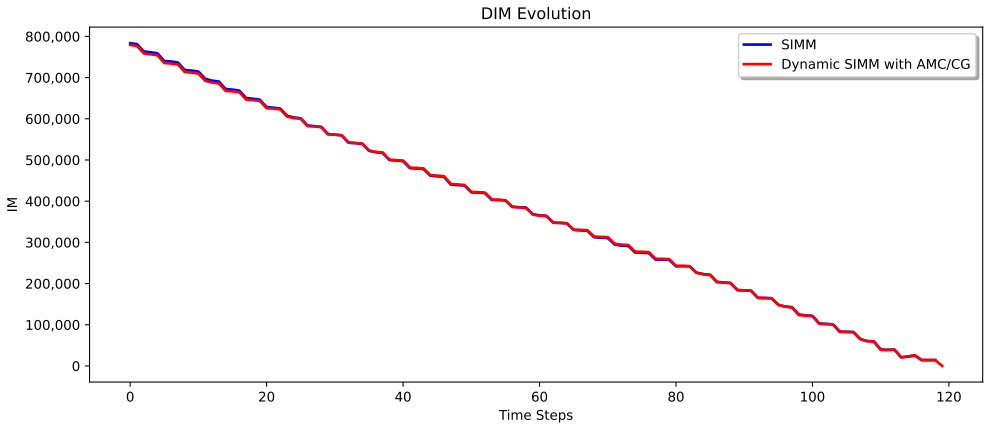

Figure 8: Evolution of SIMM vs Expected Dynamic Initial Margin (DIM) for a Euribor and SOFR-3M Swap, simulation with zero volatility, single path in case of the SIMM benchmark, 10k paths in case of DIM.

Next, we use a model with non-zero vols, i.e. re-activate calibration in the same native simulation runtime contracts. Running another script (`python run_dim2_cube.py`) we now generate a few SIMM benchmark paths and compare to the expected DIM from 10k paths, see Figure 9

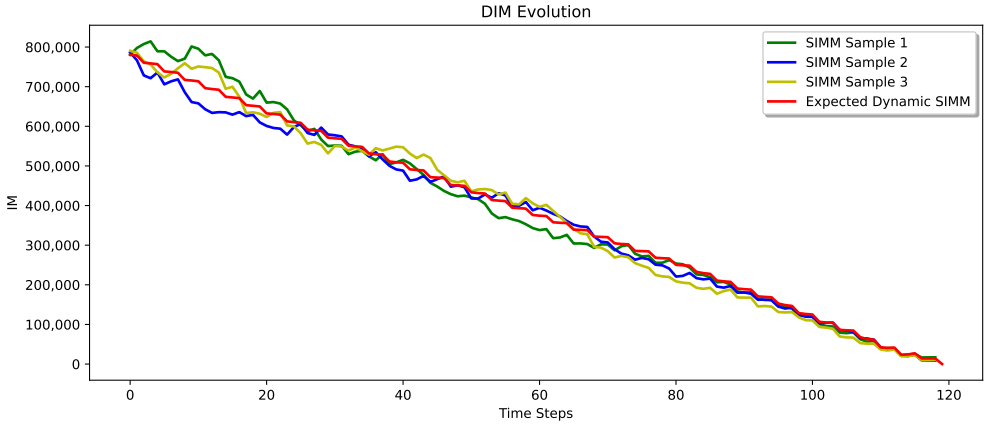

Figure 9: Evolution of SIMM vs Expected Dynamic Initial Margin (DIM) for a Euribor and SOFR-3M Swap; simulation with non-zero volatility, three paths in case of the SIMM benchmark, 10k paths in case of DIM.

Preliminary note on performance:

- The script takes about one minute to generate the benchmark SIMM evolution on a single path with 120 time steps for the example above
- Dynamic SIMM takes about 2 minutes for the expectation across 10k paths, on the same hardware (single core, Macbook Pro M2 Max), a speedup factor of 5000 compared to the pedestrian benchmark

We now zoom in on four future time points (1M, 1Y, 4Y, 8Y) and illustrate the SIMM distribution from the new report `dim_distribution.csv`. To benchmark the distributions we re-run `python run_dim2_cube.py` again, but with number of paths increased to 1000 and time steps reduced to the four dates we want to analyse, i.e. changing `input_native/dim2__vre_classic/runtime_policy/simulation_runtime.toml` accordingly. This generates an updated output in `examples/economic/initial_margin/output/DimValidation/simm_cube.csv`. Running the script `python plot_dim_distribution.py` then generates the comparison in Figure 10. Note that the Dynamic SIMM calculation takes a few seconds with 10k paths and a handful of time points, while the brute force benchmark calculation crunches 1k paths on four dates in about half an hour.

Further investigation of this new Dynamic SIMM method is work in progress:

- path-wise benchmarking
- more products - FX Options, Swaptions, FX TaRF as a complex scripted example
- investigation of alternative regression methods in AMC and their impact on quality of AAD sensitivities
- comparison to the simple regression DIM model of section 9.9.2

9.10 Exposure Family: Non-Collateral

The canonical uncollateralised exposure examples live under `examples/economic/exposure/non_collateral`. This section demonstrates exposure

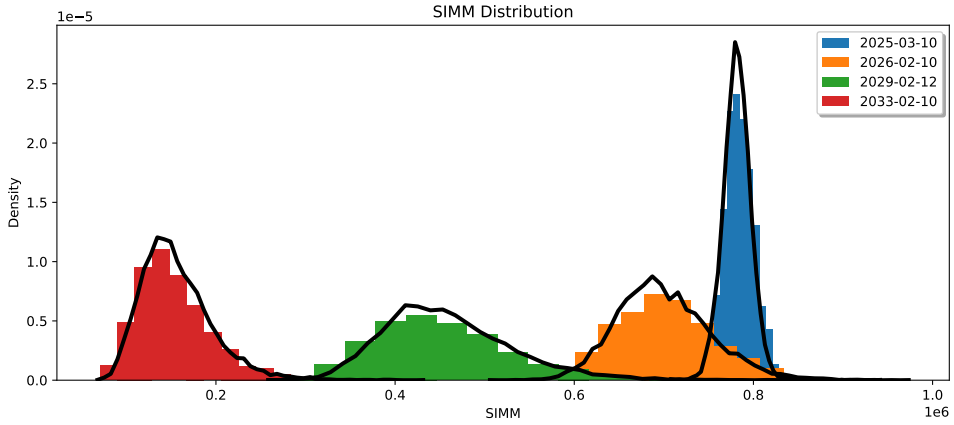

Figure 10: SIMM distribution from Dynamic SIMM (lines) vs benchmark (histograms) at time points 1M, 1Y, 4Y, 8Y from as of date.

simulation and CVA for mostly single-trade, mostly vanilla cases.

Run all non-collateral cases with:

```
cd examples/economic/exposure/non_collateral && python run.py
```

or run individual scripts as listed below.

- Swap with flat yield curve: `python run_swapflat.py`
- Swap with normal yield curves: `python run_swap.py`
- FRA: `python run_fra.py`
- European/American/Bermudan Swation and CallableSwap: `python run_swaption.py`
- Caps/Floors: `python run_capfloor.py`
- FX Forward and FX Option: `python run_fx.py`
- Resetting and Non-Resetting Cross Currency Swaps: `python run_ccs.py`
- Equity Forwards and Option: `python run_equity.py`
- Commodity Forward, Option, Swaption, APO: `python run_commodity.py`
- Inflation CPI and YOY Swap, the simulation is run twice using a Dodgson-Kainth and Jarrow-Yildirim model: `python run_inflation.py`
- Credit Default Swap: `python run_credit.py`

Somewhat more complex single-trade examples:

- Capped/Floored CMS Spread, Digital CMS Spread: `python run_cmsspread.py`
- Capped CMS Spread and Digital CMS Spread with formula-based payoff (slow): `python run_fbc.py`

Further exposure simulation features:

- Long-term simulation with and without horizon shift in the Linear Gauss Markov model: `python run_longterm.py`
- Simulation in different measures: `python run_measures.py`
- Simulation in the two-factor Hull-White model: `python run_hw2f.py`
- Wrong-Way-Risk: `python run_wwr.py`
- Flip View, switch perspectives easily for XVA: `python run_flipview.py`

All cases are discussed in the following subsections.

9.10.1 Swap with flat yield curve

We start with a vanilla single currency Swap (currency EUR, maturity 20y, notional 10m, receive fixed 2% annual, pay 6M-Euribor flat). The market yield curves (for both discounting and forward projection) are set to be flat at 2% for all maturities, i.e. the Swap is at the money initially and remains at the money on average throughout its life. Running VRE in directory `examples/economic/exposure/non_collateral` with

```
python run_swapflat.py
```

yields the exposure evolution in

```
examples/economic/exposure/non_collateral/output/swapflat/*.pdf
```

and shown in figure 11. Both Swap simulation and Swaption pricing are run with calls to the

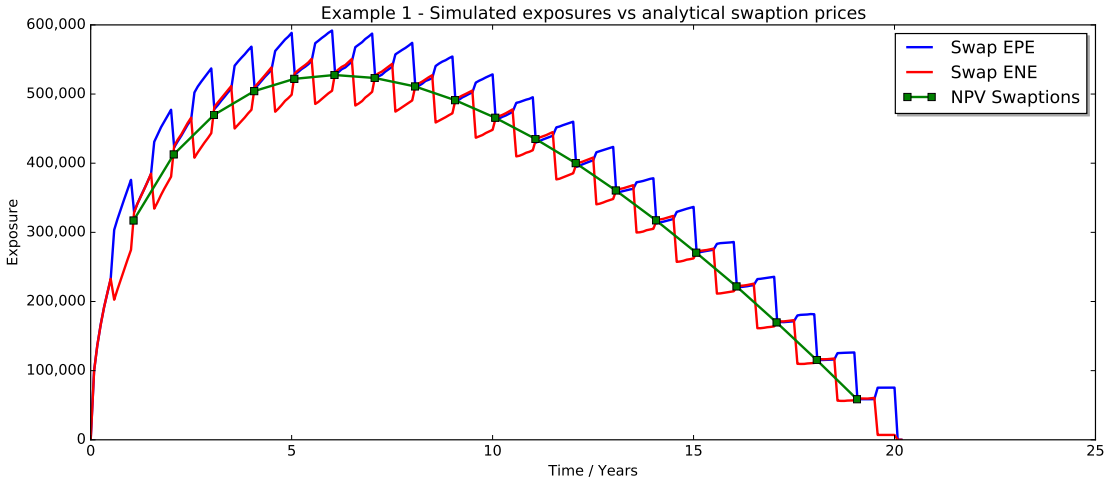

Figure 11: Vanilla ATM Swap expected exposure in a flat market environment from both parties' perspectives. The symbols are European Swaption prices. The simulation was run with monthly time steps and 10,000 Monte Carlo samples to demonstrate the convergence of EPE and ENE profiles. A similar outcome can be obtained more quickly with 5,000 samples on a quarterly time grid which is the default setting of this swap-flat case.

VRE executable, essentially

```
vre[.exe] input/vre_swapflat.toml
vre[.exe] input/vre_swaption.toml
```

which are wrapped into the script
`examples/economic/exposure/non_collateral/run_swapflat.py` provided with the VRE release. It is instructive to look into the maintained input folder in `examples/economic/exposure/non_collateral/input`, the run bundles `vre_swapflat.toml` and `vre_swaption.toml`, and the corresponding runtime packages under `input_native/vre_swapflat` and `input_native/vre_swaption`, together with the explanations in section 12.
This simple example is an important test case which is also run similarly in one of the unit test suites of VRE. The expected exposure can be seen as a European option on the underlying netting set, see also [2]. In this example, the expected exposure at some future point in time, say 10 years, is equal to the European Swaption price for an option with expiry in 10 years, underlying Swap start in 10 years and underlying Swap maturity in 20 years. We can easily compute such standard European Swaption prices for all future points in time where both Swap legs reset, i.e. annually in this case[1]. And if the simulation model has been calibrated to the points on the Swaption surface which are used for European Swaption pricing, then we can expect to see that the simulated exposure matches Swaption prices at these annual points, as in figure 11. In this case we use co-terminal ATM Swaptions for both model calibration and Swaption pricing. Moreover, as the yield curve is flat in this example, the exposures from both parties' perspectives (EPE and ENE) match not only at the annual resets, but also for the period between annual reset of both legs to the point in time when the floating leg resets. Thereafter, between floating leg (only) reset and next joint fixed/floating leg reset, we see and expect a deviation of the two exposure profiles.

9.10.2 Swap with normal yield curves

Moving to the next case we see what changes when using a realistic (non-flat) market environment. Running the example with

```
python run_swap.py
```

yields the exposure evolution in

`examples/economic/exposure/non_collateral/output/swap/*.pdf`

shown in figure 12. In this case, where the curves (discount and forward) are upward sloping, the receiver Swap is at the money at inception only and moves (on average) out of the money during its life. Similarly, the Swap moves into the money from the counterparty's perspective. Hence the expected exposure evolutions from our perspective (EPE) and the counterparty's perspective (ENE) 'detach' here, while both can still be be reconciled with payer or respectively receiver Swaption prices.

9.10.3 Forward Rate Agreement

The example in `run_fra.py` demonstrates pricing, cash flow projection and exposure simulation for two additional products

- Forward Rate Agreements
- Averaging Overnight Index Swaps

[1] Using closed form expressions for standard European Swaption prices.

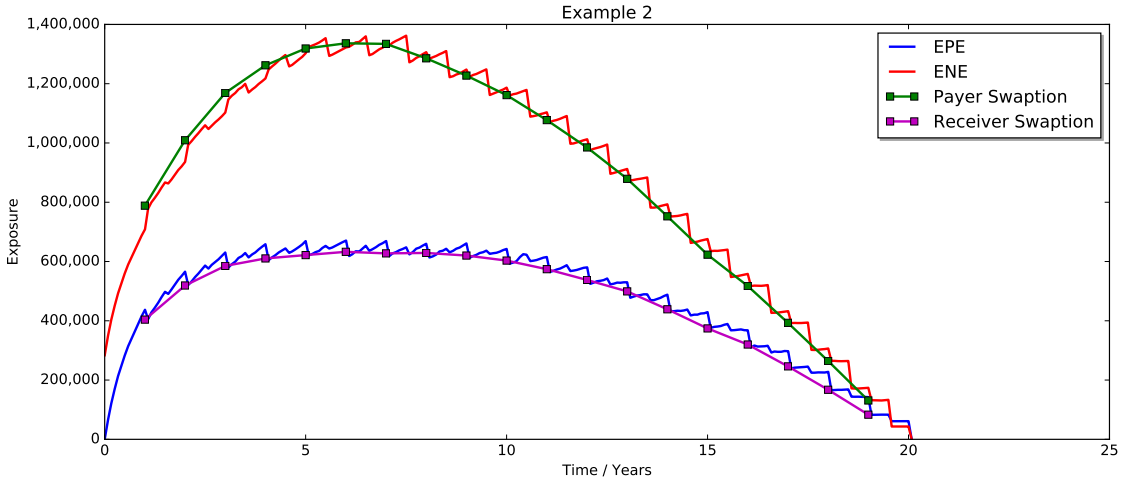

Figure 12: Vanilla ATM Swap expected exposure in a realistic market environment as of 05/02/2016 from both parties' perspectives. The Swap is the same as in figure 11 but receiving fixed 1%, roughly at the money. The symbols are the prices of European payer and receiver Swaptions. Simulation with 5000 paths and monthly time steps.

using a minimal portfolio of four trades, one FRA and three OIS. The essential results are in `npv.csv`, `flows.csv` and four `exposure_trade_*.csv` files in folder `examples/economic/exposure/non_collateral/output/fra`.

9.10.4 Swaptions

The batch in `python run_swaption.py` covers three cases

- European Swaptions with cash and physical settlement, compared to the underlying Forward Swap, see figure 13:
  The delivery type (cash vs physical) yields significantly different valuations as of today due to the steepness of the relevant yield curves (EUR). The cash settled Swaption's exposure graph is truncated at the exercise date, whereas the physically settled Swaption exposure turns into a Swap-like exposure after expiry. For comparison, the example also provides the exposure evolution of the underlying forward starting Swap which yields a somewhat higher exposure after the forward start date than the physically settled Swaption. This is due to scenarios with negative Swap NPV at expiry (hence not exercised) and positive NPVs thereafter. Note the reduced EPE in case of a Swaption with settlement of the option premium on exercise date.
- Bermudan and American Swaption, see figure 14:
  The underlying Swap is the same as in the European Swaption example above. Note in particular the difference between the Bermudan and European Swaption exposures with cash settlement: The Bermudan shows the typical step-wise decrease due to the series of exercise dates. Also note that we are using the same Bermudan option pricing engines for both settlement types, in contrast to the European case, so that the Bermudan option cash and physical exposures are identical up to the first exercise date. When running this example, you will notice the significant difference in computation time compared to the European case (ballpark 30 minutes here for 2 Swaptions, 1000 samples, 90 time steps). The Bermudan example takes significantly more computation

time because we use an LGM grid engine for pricing under scenarios in this case. In a realistic context one would more likely resort to American Monte Carlo simulation, feasible in VRE, but not provided in the current release. However, this implementation can be used to benchmark any faster / more sophisticated approach to Bermudan Swaption exposure simulation.

- European Callable Swap, represented as two trades – the non-callable Swap and a Swaption with physical delivery. We have sold the call option, i.e. the Swaption is a right for the counterparty to enter into an offsetting Swap which economically terminates all future flows if exercised. The resulting exposure evolutions for the individual components (Swap, Swaption), as well as the callable Swap are shown in figure 15. The example is an extreme case where the underlying Swap is deeply in the money (receiving fixed 5%), and hence the call exercise probability is close to one. Modify the Swap and Swaption fixed rates closer to the money ($\approx$ 1%) to see the deviation between net exposure of the callable Swap and the exposure of a 'short' Swap with maturity on exercise. We have added more recently the combined CallableSwap instrument representation in the maintained portfolio payload; compare the CallableSwap NPV in `npv.csv` to the package NPV of Swap and Swaption.

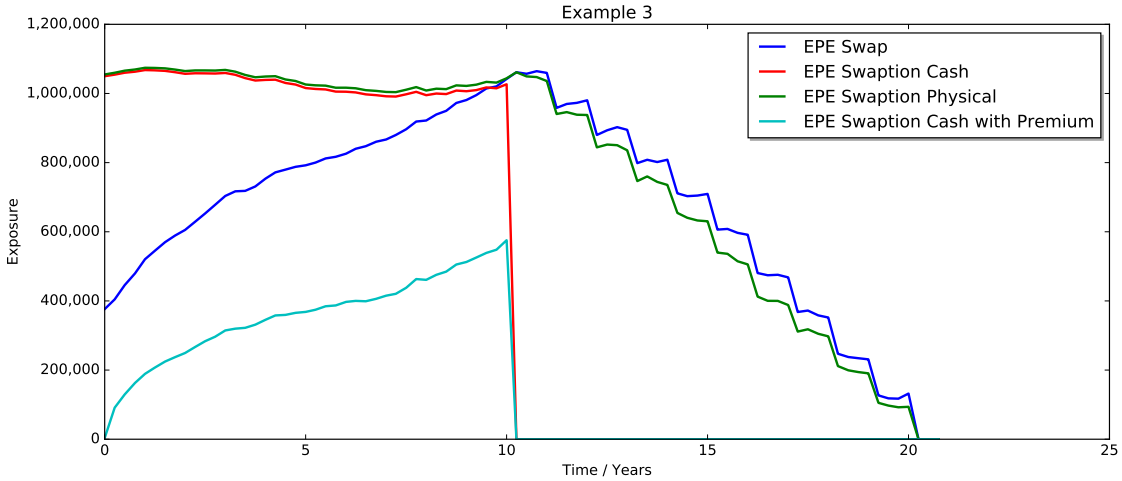

Figure 13: European Swaption exposure evolution, expiry in 10 years, final maturity in 20 years, for cash and physical delivery. Simulation with 1000 paths and quarterly time steps.

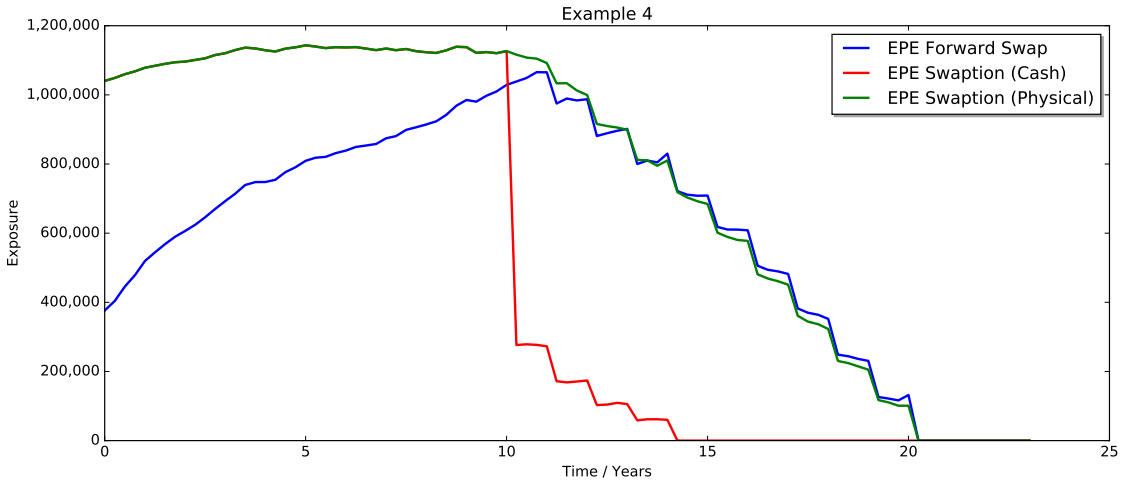

Figure 14: Bermudan Swaption exposure evolution, 5 annual exercise dates starting in 10 years, final maturity in 20 years, for cash and physical delivery. Simulation with 1000 paths and quarterly time steps.

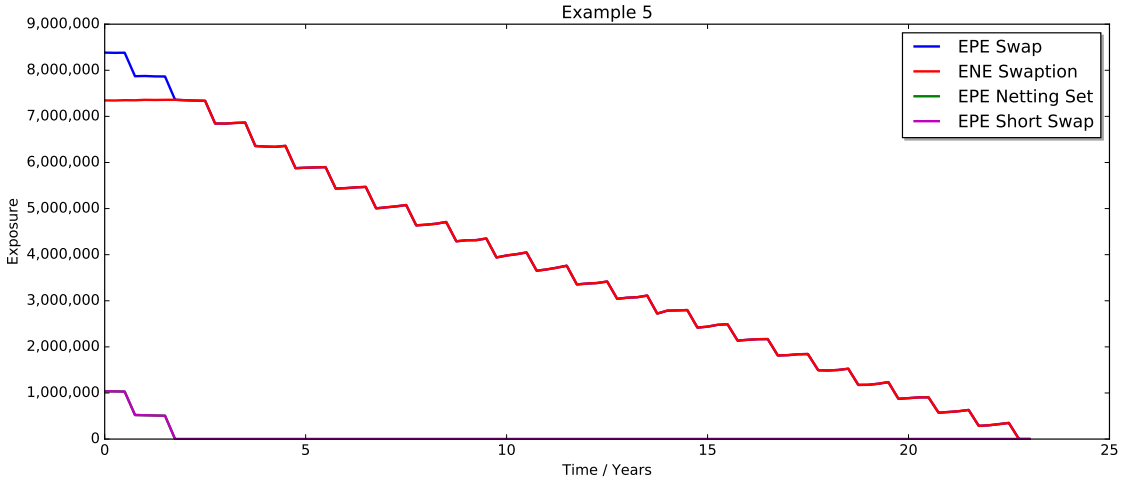

Figure 15: European callable Swap represented as a package consisting of non-callable Swap and Swaption. The Swaption has physical delivery and offsets all future Swap cash flows if exercised. The exposure evolution of the package is shown here as 'EPE Netting Set' (green line). This is covered by the pink line, the exposure evolution of the same Swap but with maturity on the exercise date. The graphs match perfectly here, because the example Swap is deep in the money and exercise probability is close to one. Simulation with 5000 paths and quarterly time steps.

9.10.5 Cap/Floor

The example `python run_capfloor.py` generates exposure evolutions of several Swaps, Caps and Floors. The example shown in figure 16 ('portfolio 1') consists of a 20y Swap receiving 3% fixed and paying Euribor 6M plus a long 20y Collar with both cap and floor at 4% so that the net exposure corresponds to a Swap paying 1% fixed.

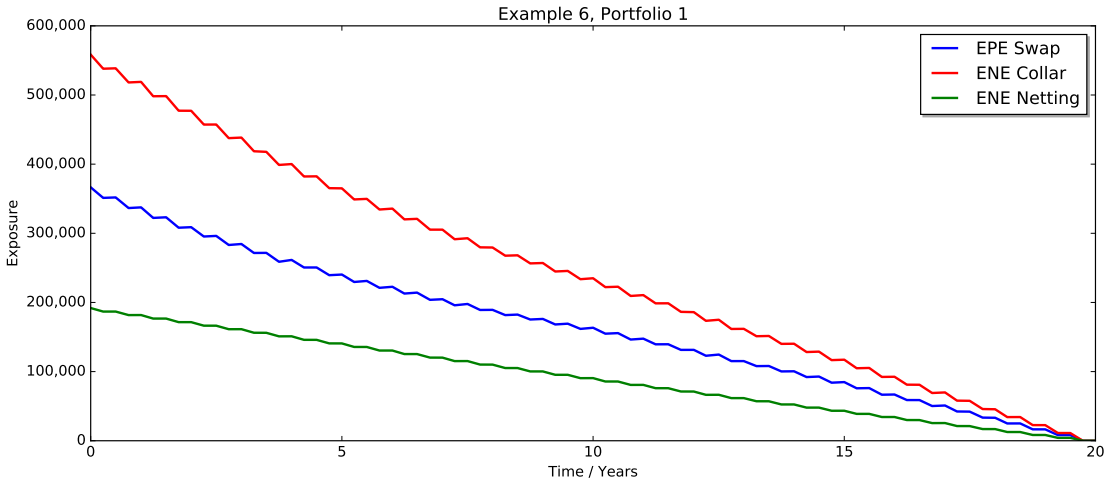

Figure 16: Swap+Collar, portfolio 1. The Collar has identical cap and floor rates at 4% so that it corresponds to a fixed leg which reduces the exposure of the Swap, which receives 3% fixed. Simulation with 1000 paths and quarterly time steps.

The second example in this folder shown in figure 17 ('portfolio 2') consists of a short Cap, long Floor and a long Collar that exactly offsets the netted Cap and Floor.

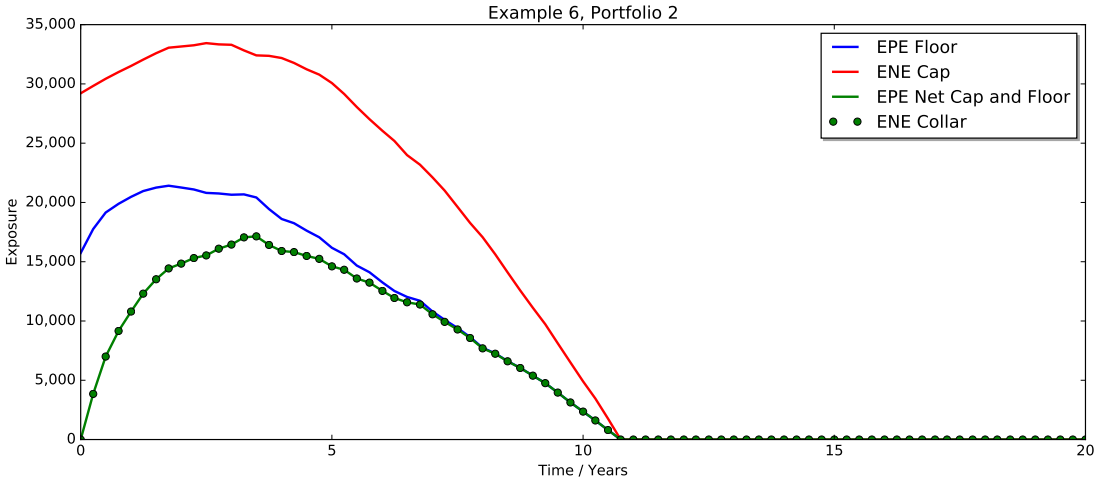

Figure 17: Short Cap and long Floor vs long Collar, portfolio 2. Simulation with 1000 paths and quarterly time steps.

Further three test portfolios are provided as part of this example. Run the example and inspect the respective output directories `examples/economic/exposure/non_collateral/output/capfloor/portfolio_#`.

9.10.6 FX Forward and FX Option

Example `python run_fx.py` generates the exposure evolution for a EUR / USD FX Forward transaction with value date in 10Y. This is a particularly simple show case because of the single cash flow in 10Y. On the other hand it checks the cross currency model implementation by means of comparison to analytic limits - EPE and ENE at the trade's value date must match corresponding Vanilla FX Option prices, as shown in figure 18.

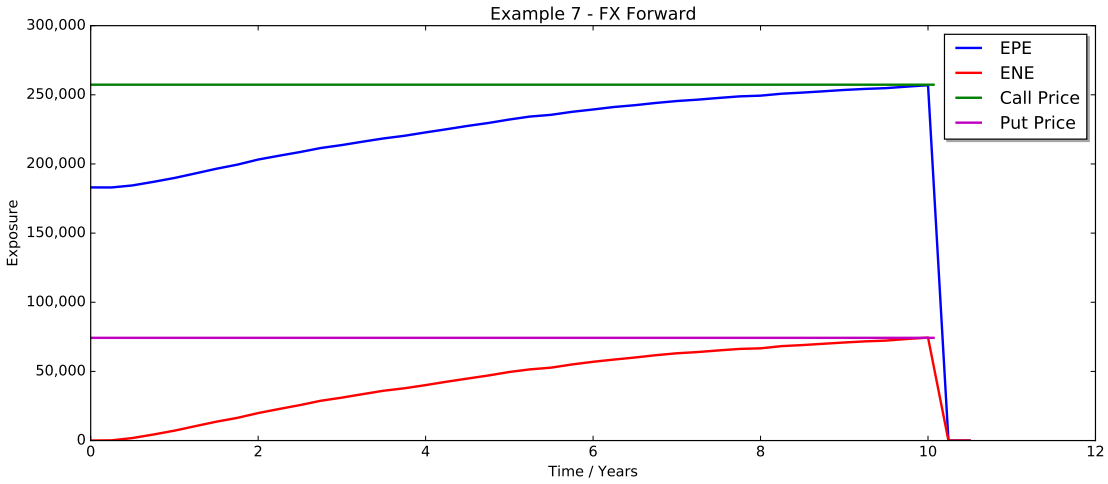

Figure 18: EUR/USD FX Forward expected exposure in a realistic market environment as of 26/02/2016 from both parties' perspectives. Value date is obviously in 10Y. The flat lines are FX Option prices which coincide with EPE and ENE, respectively, on the value date. Simulation with 5000 paths and quarterly time steps.

The same batch illustrates the exposure evolution for an FX Option, see figure 19. Recall that

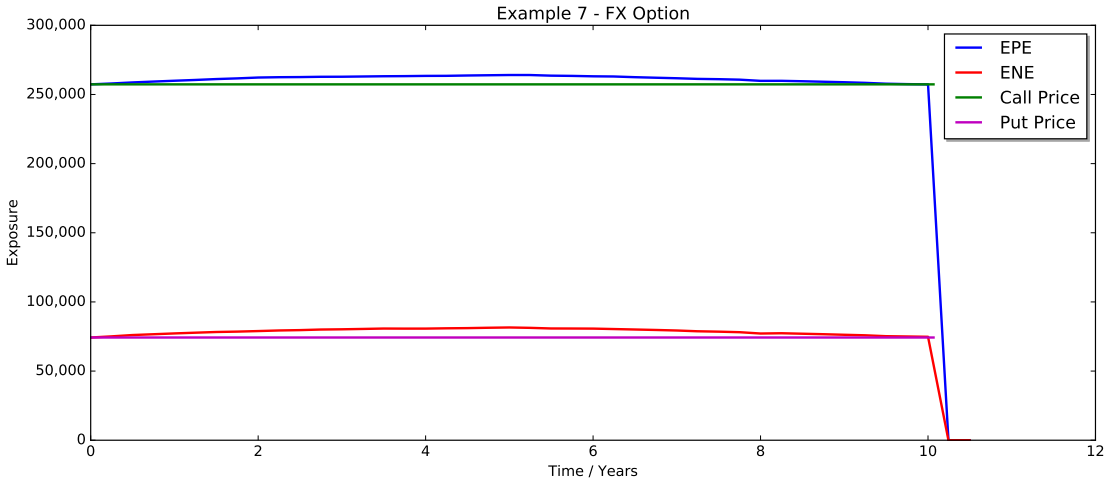

Figure 19: EUR/USD FX Call and Put Option exposure evolution, same underlying and market data as above, compared to the call and put option price as of today (flat line). Simulation with 5000 paths and quarterly time steps.

the FX Option value $NPV(t)$ as of time $0 \leq t \leq T$ satisfies

$$\frac{NPV(t)}{N(t)} = \text{Nominal} \times \mathbb{E}_t \left[\frac{(X(T) - K)^+}{N(T)} \right]$$
$$NPV(0) = \mathbb{E} \left[\frac{NPV(t)}{N(t)} \right] = \mathbb{E} \left[\frac{NPV^+(t)}{N(t)} \right] = EPE(t)$$

where $N(t)$ denotes the numeraire asset. One would therefore expect a flat exposure evolution up to option expiry. The deviation from this in VRE's simulation is due to the pricing approach chosen here under scenarios. A Black FX option pricer is used with deterministic Black volatility derived from today's volatility structure (pushed or rolled forward, see section 12.3.3). The deviation can be removed by extending the volatility modelling, e.g. implying model consistent Black volatilities in each simulation step on each path.

9.10.7 Non-Resetting and Resetting Cross Currency Swaps

Batch `run_ccs.py` demonstrates a vanilla non-resetting cross currency Swap exposure. It shows the typical blend of an Interest Rate Swap's saw tooth exposure evolution with an FX Forward's exposure which increases monotonically to final maturity, see figure 20.

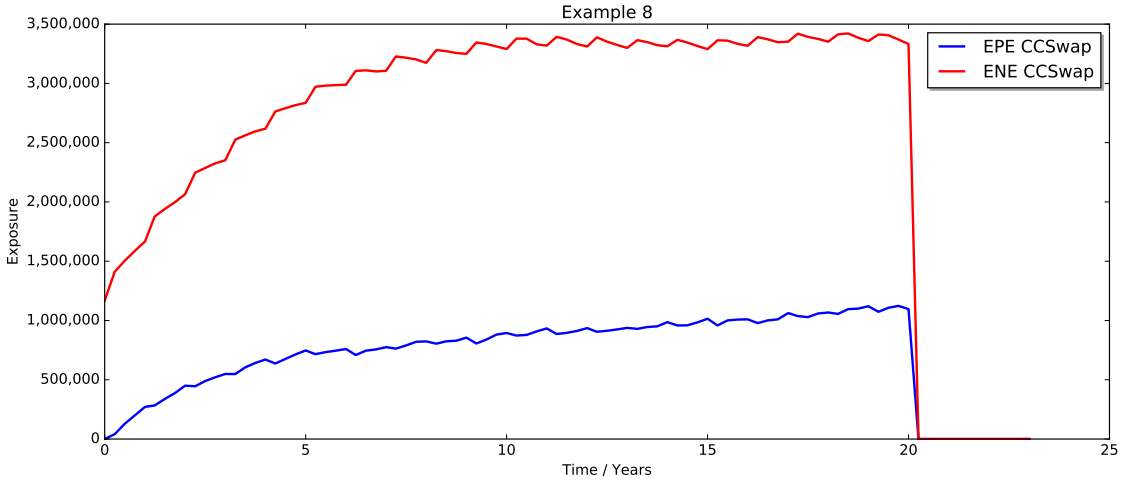

Figure 20: Cross Currency Swap exposure evolution without mark-to-market notional reset. Simulation with 1000 paths and quarterly time steps.

This run also demonstrates the serialisation of the calibrated simulation model, see the generated files `examples/economic/exposure/non_-collateral/output/swaption/calibration.xml` and `examples/economic/exposure/non_-collateral/output/swaption/calibration.csv`, as well as the brief documentation of the `calibration` analytic in section 12.1.1.

Finally, the effect of the FX resetting feature, common in Cross Currency Swaps nowadays, is also demonstrated here using a separate trade-payload instrument (see `portfolio_ccs.xml`). The example shows the exposure evolution of a EUR/USD cross currency basis Swap with FX reset at each interest period start, see figure 21. As expected, the notional reset causes an exposure collapse at each period start when the EUR leg's notional is reset to match the USD notional.

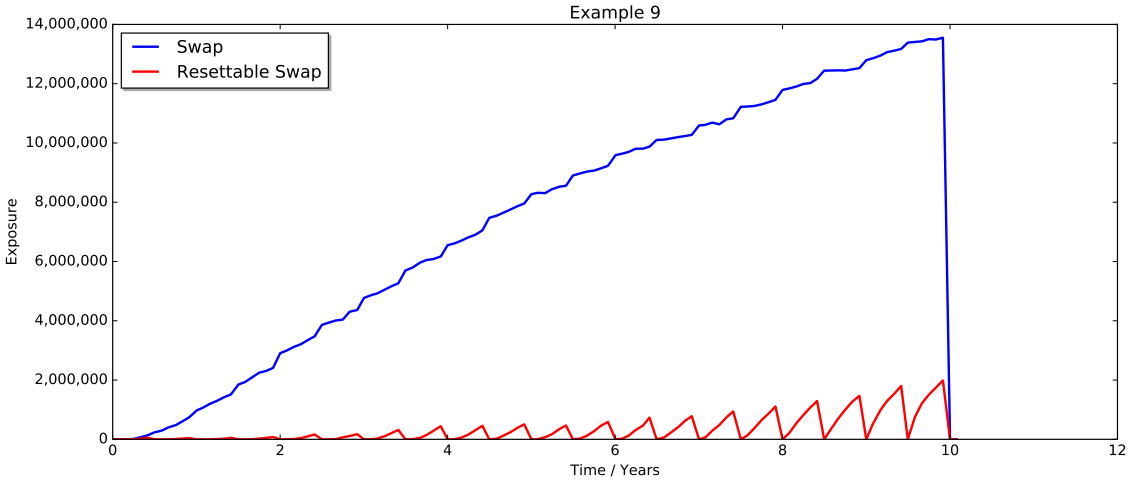

Figure 21: Cross Currency Basis Swap exposure evolution with and without mark-to-market notional reset. Simulation with 1000 paths and quarterly time steps.

9.10.8 Equity Derivatives

This example in `python run_equity.py` demonstrates the computation of NPV, sensitivities, exposures and XVA for a portfolio of OTC equity derivatives. The portfolio used in this example consists of:

- an equity call option denominated in EUR ("Luft")
- an equity put option denominated in EUR ("Luft")
- an equity forward denominated in EUR ("Luft")
- an equity call option denominated in USD ("SP5")
- an equity put option denominated in USD ("SP5")
- an equity forward denominated in USD ("SP5")
- an equity Swap in USD with return type "price" ("SP5")
- an equity Swap in USD with return type "total" ("SP5")

The step-by-step procedure for running VRE is identical for equities as for other asset classes; the same market and portfolio data files are used to store the equity market data and trade details, respectively. For the exposure simulation, the calibration parameters for the equity risk factors are carried by `input/vre_equity.toml` and the published simulation contract under `input_native/vre_equity/runtime_policy/simulation_runtime.toml`.

Looking at the MtM results in the output file `npv.csv` we observe that put-call parity ($V_{Fwd} = V_{Call} - V_{Put}$) is observed as expected. Looking at Figure 22 we observe that the Expected Exposure profile of the equity call option trade is relatively smooth over time, while for the equity forward trade the Expected Exposure tends to increase as we approach maturity. This behaviour is similar to what we observe in section 9.10.6.

9.10.9 Commodity Derivatives

Calling

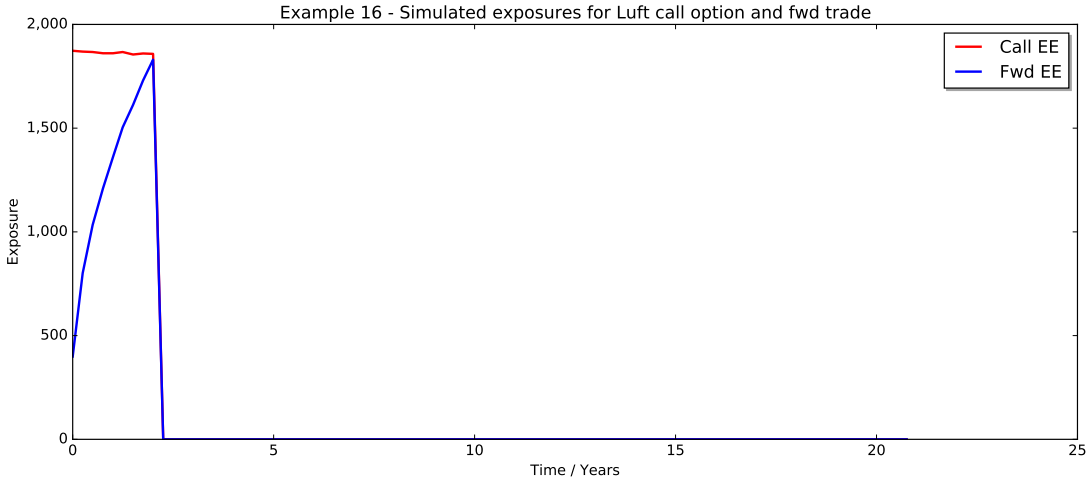

Figure 22: Equity ("Luft") call option and OTC forward exposure evolution, maturity in approximately 2.5 years. Simulation with 10000 paths and quarterly time steps.

```
python run_commodity.py
```

demonstrates pricing and exposure simulation for a portfolio including a

- Commodity Forward
- Commodity Swap
- European Commodity Option
- Commodity Average Price Option
- Commodity Swaption

with the usual results, exposure reports and graphs.

9.10.10 Inflation CPI and YOY Swap - TODO

The example called with `python run_inflation.py` runs two exposure simulation batches for CPI and YOY Inflation Swaps

- using the Dodgson-Kainth (DK) inflation model
- using the Jarrow-Yildirim (JY) inflation model

We use the same portfolio in both batches that comprises four trades

- EU and UK CPI Swaps
- EU and UK YOY Swaps

Figures 23 and 24 show the exposure (EPE) graphs for all trades.

TODO: Discuss differences, check model calibrations

9.10.11 Credit Default Swap

Calling

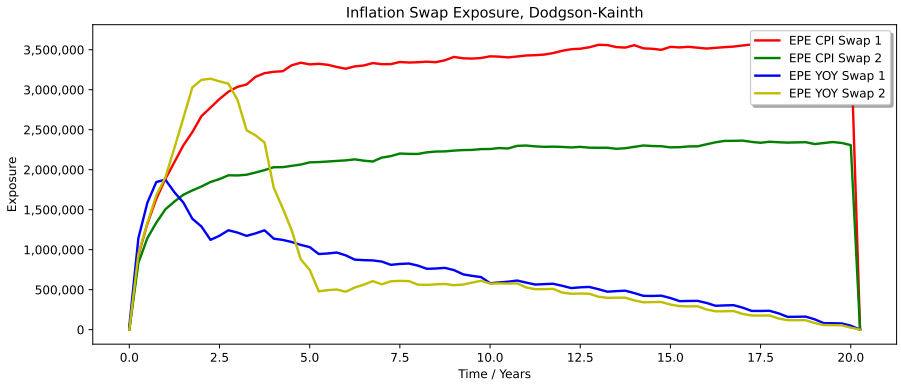

Figure 23: CPI and YOY exposure evolution using the Dodgson-Kainth model.

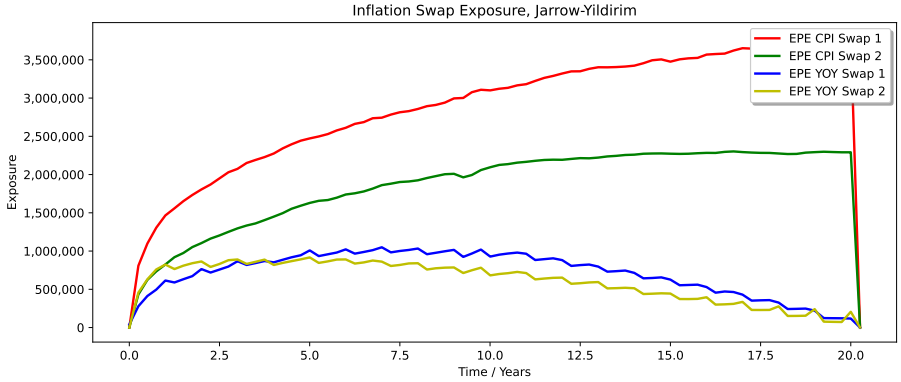

Figure 24: CPI and YOY exposure evolution using the Jarrow-Yildirim model.

```
python run_credit.py
```

runs the credit variant of the Swap exposure in 9.10.1. Running VRE here yields the exposure evolution shown in figure 25. Both CDS simulation and CDS Option pricing are run with

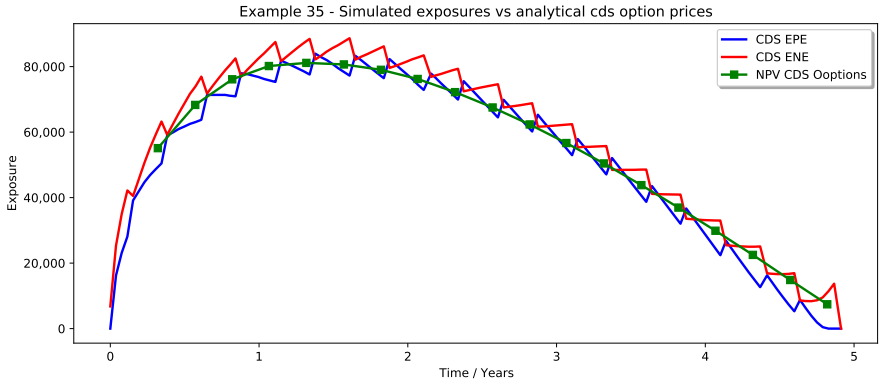

Figure 25: Credit Default Swap expected exposure in a flat market environment from both parties' perspectives. The symbols are CDS Option prices. The simulation was run with bi-weekly time steps and 10,000 Monte Carlo samples to demonstrate the convergence of EPE and ENE profiles. A similar outcome can be obtained more quickly with 5,000 samples on a monthly time grid which is the default setting here.

calls to the VRE executable, essentially

```
vre[.exe] input/vre_credit.toml
vre[.exe] input/vre_creditoptions.toml
```

which are wrapped into the script
`examples/economic/exposure/non_collateral/run_credit.py` provided with the VRE release.

This example demonstrates credit simulation using the LGM model and the calculation of Wrong Way Risk due to credit correlation between the underlying entity of the CDS and the counterparty of the CDS trade via dynamic credit. Positive correlation between the two names weakens the protection of the CDS whilst negative correlation strengthens the protection.

The following table lists the XVA result from the example at different levels of correlation.

Correlation	NettingSetId	CVA	DVA	FBA	FCA
-100%	CPTY_B	-2,638	2,906	486	-1,057
-90%	CPTY_B	-2,204	2,906	488	-1,053
-50%	CPTY_B	-485	2,906	493	-1,040
-40%	CPTY_B	-60	2,906	495	-1,037
-30%	CPTY_B	363	2,906	496	-1,033
-20%	CPTY_B	784	2,906	498	-1,030
-10%	CPTY_B	1,204	2,906	500	-1,027
0%	CPTY_B	1,621	2,906	501	-1,023
10%	CPTY_B	2,036	2,906	503	-1,020
20%	CPTY_B	2,450	2,906	504	-1,017
30%	CPTY_B	2,861	2,906	506	-1,013
40%	CPTY_B	3,271	2,906	507	-1,010
50%	CPTY_B	3,679	2,906	509	-1,017
90%	CPTY_B	5,290	2,906	515	-994
100%	CPTY_B	5,689	2,906	517	-991

Table 11: CDS XVA results with LGM model

9.10.12 Capped/Floored (Digital) CMS Spread

Calling

```
python run_cmsspread.py
```

runs pricing and exosure simulation for

- Capped/Floored CMS Spreads
- CMS Spreads with Digital Caps/Floors

Results are found in folder
`examples/economic/exposure/non_collateral/output/cmsspread/*` and exposure graphs in
`examples/economic/exposure/non_collateral/output/mpl_cmsspread.pdf`.

9.10.13 Capped (Digital) CMS Spread with Formula-based Payoff

Calling

```
python run_fbc.py
```

runs pricing, exosure simulation and XVA for Swaps with CMS Spread and Digital CMS Spread legs using both formula-based payoff and “classic” VRE XML. The example portfolio

also includes a Bond with formula-based payoff. Results are found in folder `examples/economic/exposure/non_collateral/output/fbc/*`

The formula-based leg can be seen as a predecessor of the more versatile scripted trade framework. However, the formula leg may be used to apply a multi-factor Log-normal Swap Rate model instead of the Gaussian interest rate models currently applied in the scripted trade framework.

9.10.14 Long-term simulation

Calling

```
python run_longterm.py
```

demonstrates an effect that, at first glance, seems to cause a serious issue with long term simulations. Fortunately this can be avoided quite easily in the Linear Gauss Markov model setting that is used here.
In the example we consider a Swap with maturity in 50 years in a flat yield curve environment. If we simulate this naively as in all previous cases, we obtain a particularly noisy EPE profile that does not nearly reconcile with the known exposure (analytical Swaption prices). This is shown in figure 26 ('no horizon shift'). The origin of this issue is the width of the risk-neutral NPV distribution at long time horizons which can turn out to be quite small so that the Monte Carlo simulation with finite number of samples does not reach far enough into the positive or negative NPV range to adequately sample the distribution, and estimate both EPE and ENE in a single run. Increasing the number of samples may not solve the problem, and may not even be feasible in a realistic setting.
The way out is applying a 'shift transformation' to the Linear Gauss Markov model, see the published simulation contract for `input/vre_longterm_2.toml`, `input_native/vre_longterm_2/runtime_policy/simulation_runtime.toml`:

```
parameter_transformations = [
  { shift_horizon = 30.0, scaling = 1.0 }
]
```

The effect of the 'ShiftHorizon' parameter T is to apply a shift to the Linear Gauss Markov model's $H(t)$ parameter (see [2]) *after* the model has been calibrated, i.e. to replace:

$$H(t) \rightarrow H(t) - H(T)$$

It can be shown that this leaves all expectations computed in the model (such as EPE and ENE) invariant. As explained in [15], subtracting an H shift effectively means performing a change of measure from the 'native' LGM measure to a T-Forward measure with horizon T, here 30 years. Both negative and positive shifts are permissible, but only negative shifts are connected with a T-Forward measure and improve numerical stability.
In our experience it is helpful to place the horizon in the middle of the portfolio duration to significantly improve the quality of long term expectations. The effect of this change (only) is shown in the same figure 26 ('shifted horizon'). Figure 27 further illustrates the origin of the problem and its resolution: The rate distribution's mean (without horizon shift or change of measure) drifts upwards due to convexity effects (note that the yield curve is flat in this example), and the distribution's width is then too narrow at long horizons to yield a sufficient

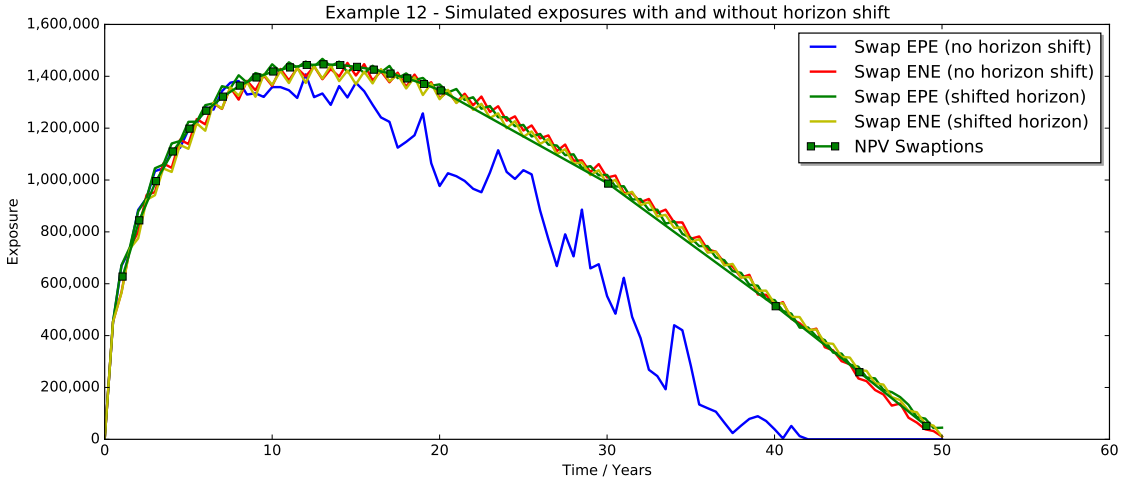

Figure 26: Long term Swap exposure simulation with and without horizon shift.

number of low rate scenarios with contributions to the Swap's *EPE* (it is a floating rate payer). With the horizon shift (change of measure), the distribution's mean is pulled 'back' at long horizons, because the convexity effect is effectively wiped out at the chosen horizon, and the expected rate matches the forward rate.

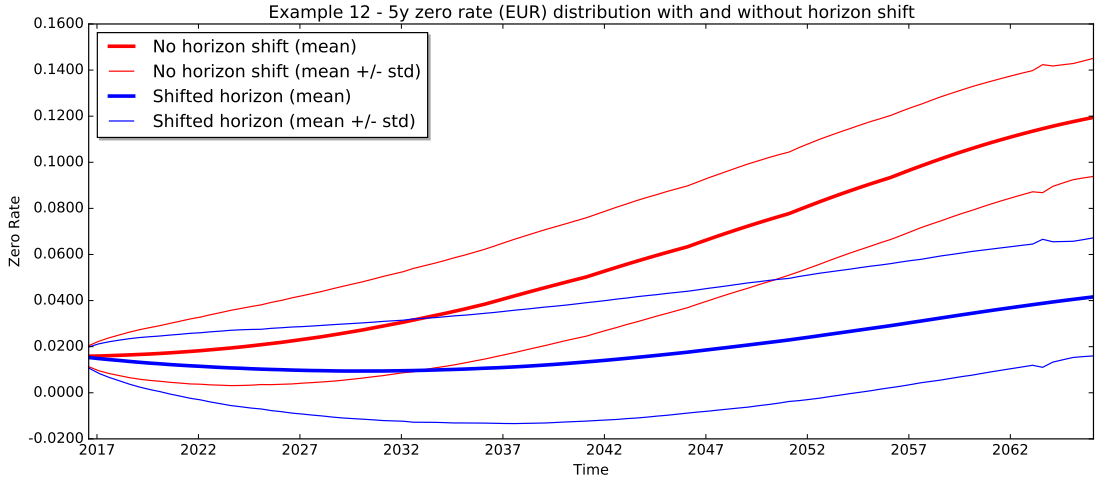

Figure 27: Evolution of rate distributions with and without horizon shift (change of measure). Thick lines indicate mean values, thin lines are contours of the rate distribution at ± one standard deviation.

9.10.15 Choice of Measure

Calling

```
python run_measures.py
```

illustrates the effect of measure changes on simulated expected and peak exposures. For that purpose we reuse the example in section 9.10.1 (un-collateralized vanilla swap exposure) and run the simulation three times with different risk-neutral measures,

- in the LGM measure as in Example 1, using `input/vre_measure_lgm.toml`; this is the default also if the measure field is omitted in the resolved simulation contract

- in the more common Bank Account measure, using `input/vre_measure_ba.toml`
- in the T-Forward measure with horizon T=20 at the Swap maturity, using `input/vre_measure_fwd.toml` and the related shift-horizon setting in the published simulation contract

The results are summarized in the exposure evolution graphs in figure 28. As expected, the expected exposures evolutions match across measures, as these are expected discounted NPVs and hence measure independent. However, peak exposures are dependent on the measure choice as confirmed graphically here. Many more measures are accessible with VRE, by way of varying the T-Forward horizon which was chosen arbitrarily here to match the Swap's maturity.

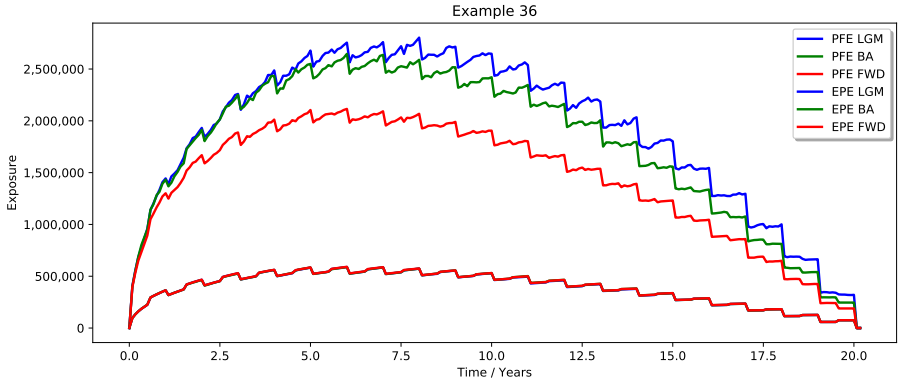

Figure 28: Evolution of expected exposures (EPE) and peak exposures (PFE at the 95% quantile) in three measures, LGM, Bank Account, T-Forward with T=20, with 10k Monte Carlo samples.

9.10.16 Simulation in the two-factor Hull-White model

Calling

```
python run_hw2f.py
```

kicks off two batches, a first model calibration batch and a second exposure simulation batch.

The first batch illustrates the model calibration and scenario generation under a Hull-White multifactor model with output in folder `examples/economic/exposure/non_collateral/output/hw2f_calibration`. The model is driven by two independent Brownian motions and has four states. The diffusion matrix sigma is therefore 2 x 4. The reversion matrix is a 4 x 4 diagonal matrix and entered as an array. Both diffusion and reversion are constant in time. Their values are not calibrated to the option market, but are hardcoded in the published simulation contract for the HW2F run bundle.

The values for the diffusion and reversion matrices were fitted to the first two principal components of a (hypothetical) analysis of absolute rate curve movements. These input principal components can be found in inputeigenvectors.csv in the input folder. The tenor is given in years, and the two components are given as column vectors, see table 12.

The first eigenvector represent perfectly parallel movements. The second eigenvector represent a rotation around the 7y point of the curve. Furthermore we prescribe an annual

tenor	eigenvector 1	eigenvector 2
1	0.353553390593	-0.537955502871
2	0.353553390593	-0.374924478795
3	0.353553390593	-0.252916811525
5	0.353553390593	-0.087587539893
10	0.353553390593	0.12267800393
15	0.353553390593	0.240659435416
20	0.353553390593	0.339148675322
30	0.353553390593	0.552478951238

Table 12: Input principal components

volatility of 0.0070 for the first components and 0.0030 for the second one. The values can be compared to normal (bp) volatilities.

We follow [14] chapter 12.1.5 "Multi-Factor Statistical Gaussian Model" to calibrate the diffusion and reversion matrices to the prescribed components and volatilities. We do not detail the procedure here and refer the interested reader to the given reference.

The example generates a single monte carlo path with 5000 daily steps and outputs the generated scenarios in scenariodump.csv. The python script pca.py performs a principal component analysis on this output. The model implied eigenvalues are given in table 13.

number	value
1	4.9144936649319346e-05
2	8.846877641067412e-06
3	5.82566039467854e-10
4	2.1298948225571415e-10
5	9.254913949332787e-11
6	1.0861256211767673e-11
7	8.478795662698618e-14
8	9.74468069377584e-13

Table 13: Input principal components

Only the first two values are relevant, the following are all close to zero. The square root of the first two eigenvalues is given in table 14.

number	sqrt(value)
1	0.007010344973631422
2	0.0029743701250966414

Table 14: Input principal components

matching the prescribed input values of 0.0070 and 0.0030 quite well. The corresponding eigenvectors are given in etable 15.

again matching the input principal components quite well. The second eigenvector is the negative of the input vector here (the principal component analysis can not distinguish these of course).

tenor	eigenvector 1	eigenvector 2
1	0.34688826736335926	0.5441204725042812
2	0.3489303472083185	0.380259707350115
3	0.350362134519783	0.2581408080614405
5	0.3523983915961889	0.09230899007104967
10	0.3550169593982022	-0.11856777284904292
15	0.35647835947136625	-0.23676104168229614
20	0.3577146190751303	-0.3354486339442275
30	0.36042236352102563	-0.549124709243042

Table 15: Input principal components

The example also produces a plot comparing the input eigenvectors and the model implied eigenvectors as shown in figure 29.

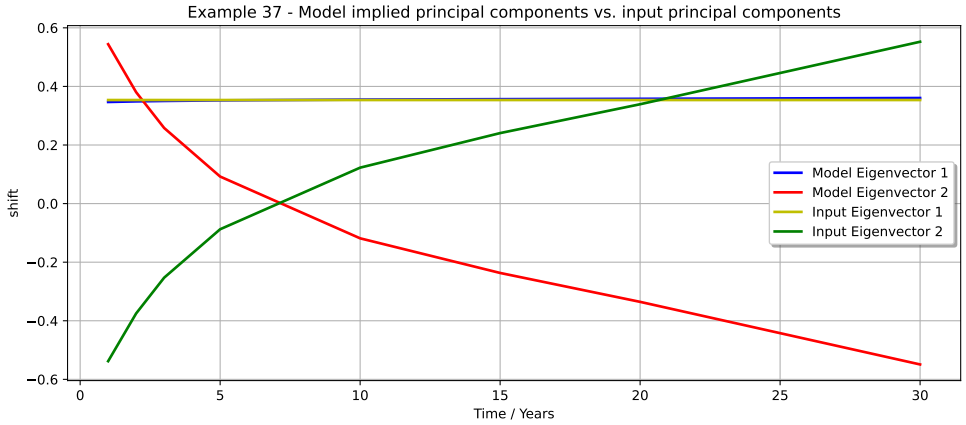

Figure 29: Input and model implied eigenvectors for a Hull-White 4-factor model calibrated to 2 principal components of rate curve movements (parallel + rotation). Notice that the model implied 2nd eigenvector is the negative of the input vector.

The second batch is similar to the Example in section 9.10.7 (EPE, ENE for a xccy swap), but uses a multifactor HW model for EUR and USD to generate scenarios. The parametrization of the HW models is taken from the previous run, resukts are found in folder `examples/economic/exposure/non_collateral/output/hw2f`.

Each of the two factors of each HW model is correlated with each of the two factors of the other currency's HW model and with the FX factors. Remember that the factors represent principal components of interest rate movements and so the correlations can be interpreted as correlations of these principal components with each other and the fx rate processes.

9.10.17 Wrong-Way-Risk

Calling

```
python run_wwr.py
```

runs an extension of the example in section 9.10.1 (single uncollateralised Swap) with dynamic credit and non-zero IR-CR correlation. Results are found in folder `examples/economic/exposure/non_collateral/output/wwr`. As we are paying float,

negative correlation implies that we pay more when the counterparty's credit worsens, leading to a surge of CVA.

The following table lists the XVA result from the example at different levels of correlation.

Correlation	NettingSetId	CVA	DVA	FBA	FCA
-30%	CPTY_A	105,146	68,061	31,519	-4,127
-20%	CPTY_A	88,442	68,061	30,976	-4,219
-10%	CPTY_A	71,059	68,061	30,439	-4,314
0%	CPTY_A	52,983	68,061	29,909	-4,411
10%	CPTY_A	34,199	68,061	29,386	-4,511
20%	CPTY_A	14,691	68,061	28,869	-4,614
30%	CPTY_A	-5,554	68,061	28,360	-4,719

Table 16: IR Swap XVA results with LGM model

9.10.18 Flip View

Calling

```
python run_flipview.py
```

demonstrates how VRE can be used to quickly switch perspectives in XVA calculations with minimal changes in the run bundle only. In particular it avoids manipulating the portfolio input or the netting set. The maintained packs `input/vre_flipview.toml`, `input/vre_flipview_normal.toml` and `input/vre_flipview_reversed.toml` vary only the flip-view and funding postfix settings in the `[analytics.xva]` block.

9.11 Exposure Family: Collateral

This section documents the canonical collateral branch `examples/economic/exposure/collateral`. We demonstrate exposure calculation and XVA at netting-set level for a small portfolio of three swaps in different currencies and illustrate the impact of collateral choices.

9.11.1 MPoR (Biweekly) Grid

Move to folder `examples/economic/exposure/collateral`. Calling

```
python run_biweekly.py
```

performs several calculations on a bi-weekly date grid with grid spacing that matches the Margin Period of Risk (MPoR). The results of the related VRE runs are found in sub-directories of `examples/economic/exposure/collateral/output/` (iah_0, iah_1, nocollateral, vm_threshold_*, vm_mta, vm_mpor)

The effect of the various collateral choices is illustrated in three plots

- no collateral - figure 30,
- collateral with threshold (THR) 1m EUR, minimum transfer amount (MTA) 100k EUR, margin period of risk (MPOR) 2 weeks - figure 31
- collateral with zero THR and MTA, and MPOR 2w - figure 32

The exposure graphs with collateral and positive margin period of risk show typical spikes. What is causing these? As sketched in [2], VRE uses a *classical collateral model* that applies collateral amounts to offset exposure with a time delay that corresponds to the margin period of risk. The spikes are then caused by instrument cash flows falling between exposure measurement dates d_1 and d_2 (an MPOR apart), so that a collateral delivery amount determined at d_1 but settled at d_2 differs significantly from the closeout amount at d_2 causing a significant residual exposure for a short period of time. See for example [17] for a recent detailed discussion of collateral modelling. The approach currently implemented in VRE corresponds to *Classical+* in [17], the more conservative approach of the classical methods. The less conservative alternative, *Classical-*, would assume that both parties stop paying trade flows at the beginning of the MPOR, so that the P&L over the MPOR does not contain the cash flow effect, and exposure spikes are avoided. Note that the size and position of the largest spike in figure 31 is consistent with a cash flow of the 40 million GBP Swap in the example's portfolio that rolls over the 3rd of March and has a cash flow on 3 March 2020, a bit more than four years from the evaluation date.

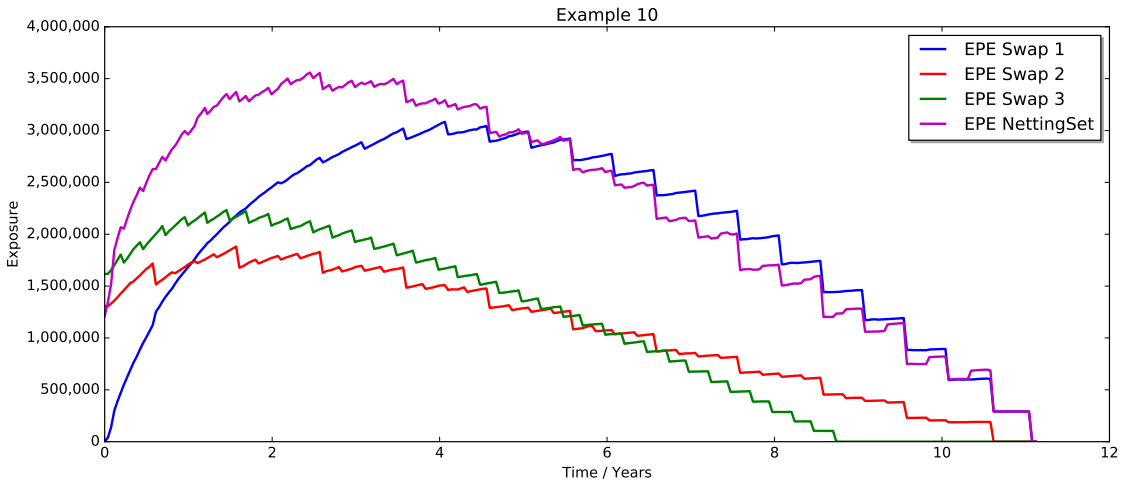

Figure 30: Three Swaps netting set, no collateral. Simulation with 5000 paths and bi-weekly time steps.

CVA, DVA, FVA, COLVA, MVA, Collateral Floor

We use one of the cases to demonstrate the XVA outputs, see folder `examples/economic/exposure/collateral/output/vm_threshold_dim`.

The summary of all value adjustments (CVA, DVA, FVA, COLVA, MVA, as well as the Collateral Floor) is provided in file `xva.csv`. The file includes the allocated CVA and DVA numbers to individual trades as introduced in the next section. The following table illustrates the file's layout, omitting the three columns containing allocated data.

TradeId	NettingSetId	CVA	DVA	FBA	FCA	COLVA	MVA	CollateralFloor	BaselEPE	BaselEEPE
	CPTY_A	6,521	151,193	-946	72,103	2,769	-14,203	189,936	113,260	1,211,770
Swap_1	CPTY_A	127,688	211,936	-19,624	100,584	n/a	n/a	n/a	2,022,590	2,727,010
Swap_3	CPTY_A	71,315	91,222	-11,270	43,370	n/a	n/a	n/a	1,403,320	2,183,860
Swap_2	CPTY_A	68,763	100,347	-10,755	47,311	n/a	n/a	n/a	1,126,520	1,839,590

The line(s) with empty TradeId column contain values at netting set level, the others contain uncollateralised single-trade VAs. Note that COLVA, MVA and Collateral Floor are only available at netting set level at which collateral is posted.

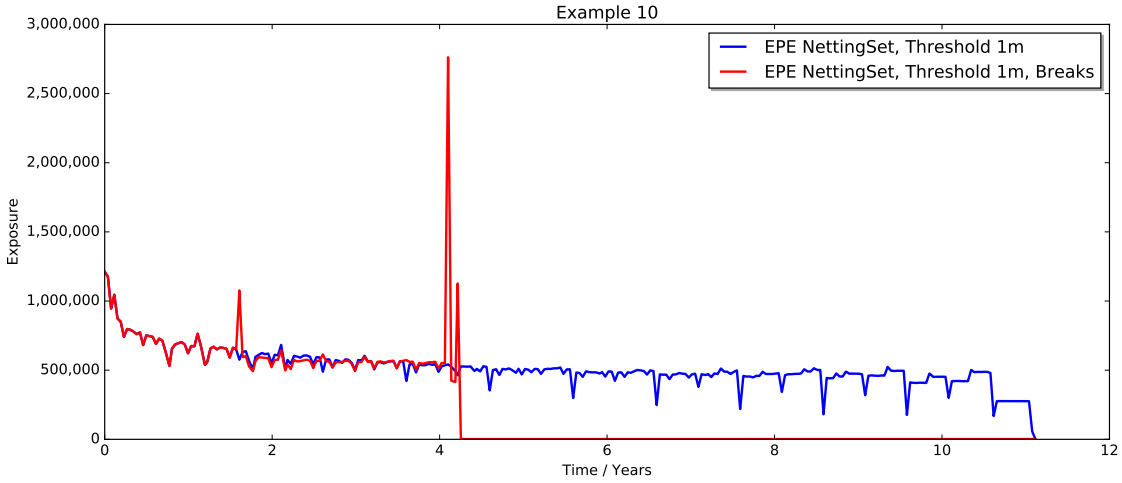

Figure 31: Three Swaps netting set, THR=1m EUR, MTA=100k EUR, MPOR=2w. The red evolution assumes that the each trade is terminated at the next break date. The blue evolution ignores break dates. Simulation with 5000 paths and bi-weekly time steps.

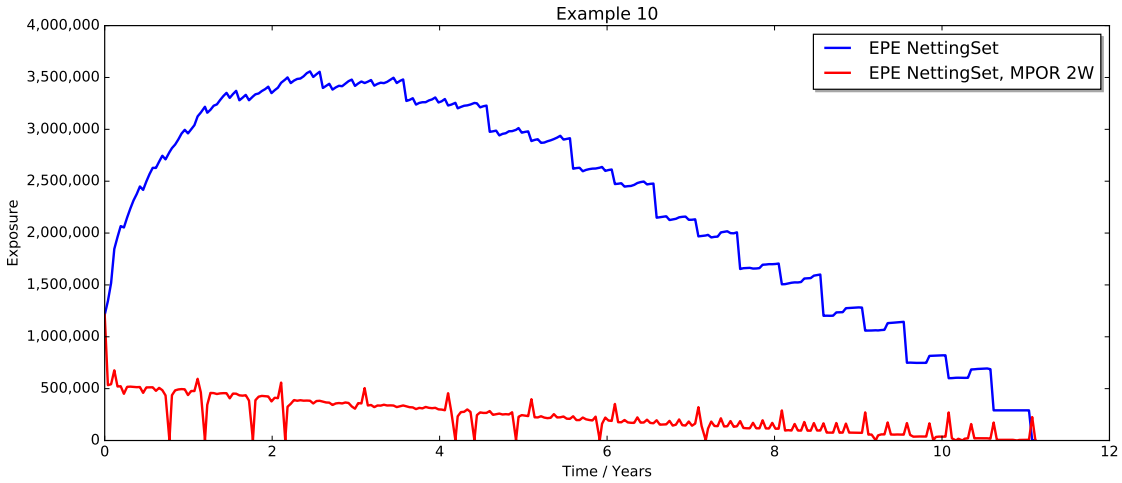

Figure 32: Three Swaps, THR=MTA=0, MPOR=2w. Simulation with 5000 paths and bi-weekly time steps.

Detailed output is written for COLVA and Collateral Floor to file `colva_nettingset_*.csv` which shows the incremental contributions to these two VAs through time.

Exposure Reports & XVA Allocation to Trades

We also illustrate here the layout of an exposure report produced by VRE. The report shows the exposure evolution of Swap_1 without collateral which is found in folder `examples/economic/exposure/collateral/output/nocollateral/exposure_trade_Swap_1.csv`:

TradeId	Date	Time	EPE	ENE	AllocEPE	AllocENE	PFE	BaselEE	BaselEEE
Swap_1	05/02/16	0.0000	0	1,711,748	0	0	0	0	0
Swap_1	19/02/16	0.0383	38,203	1,749,913	-1,200,677	511,033	239,504	38,202	38,202
Swap_1	04/03/16	0.0765	132,862	1,843,837	-927,499	783,476	1,021,715	132,845	132,845
Swap_1	...	...	...	...	...	...	...	...	...

The exposure measures EPE, ENE and PFE, the Basel exposure measures EE_B and EEE_B, as well as allocated exposures are defined in [2]. The PFE quantile and allocation method are chosen as described in section 12.1.1.
In addition to single trade exposure files, VRE produces an exposure file per netting set. The example from the same folder as above is:

NettingSet	Date	Time	EPE	ENE	PFE	ExpectedCollateral	BaselEE	BaselEEE
CPTY_A	05/02/16	0.0000	1,203,836	0	1,203,836	0	1,203,836	1,203,836
CPTY_A	19/02/16	0.0383	1,337,713	137,326	3,403,460	0	1,337,651	1,337,651
CPTY_A	...	...	...	...	...	...	...	...

Allocated exposures are missing here, as they make sense at the trade level only, and the expected collateral balance is added for information (in this case zero as collateralisation is deactivated in this example).

The allocation of netting set exposure and XVA to the trade level is frequently required by finance departments. This allocation is also featured here. We start again with the uncollateralised case in figure 33, followed by the case with threshold 1m EUR in figure 34. In both cases we apply the *marginal* (Euler) allocation method as published by Pykhtin and

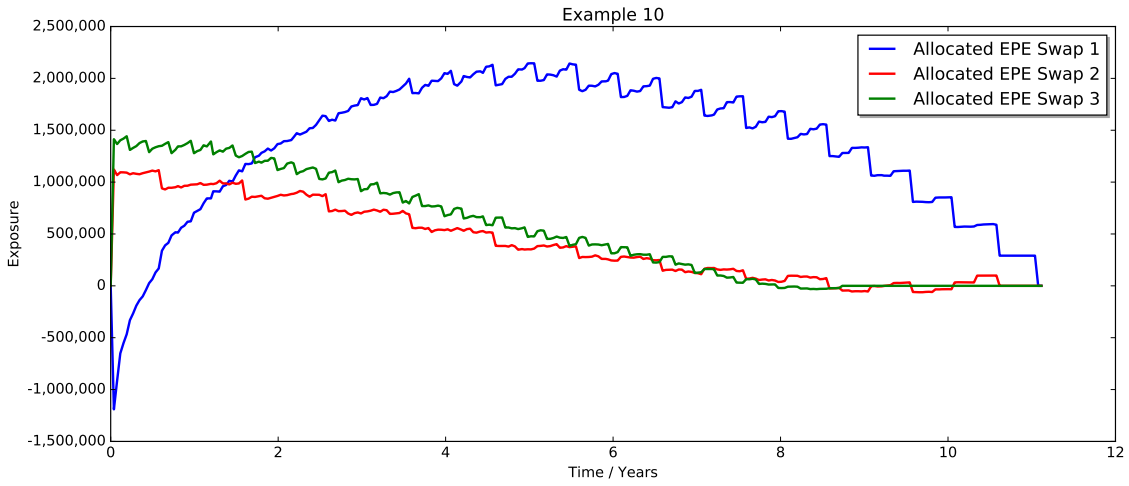

Figure 33: Exposure allocation without collateral. Simulation with 5000 paths and bi-weekly time steps.

Rosen in 2010, hence we see the typical negative EPE for one of the trades at times when it reduces the netting set exposure. The case with collateral moreover shows the typical spikes in the allocated exposures. The analytics results also feature allocated XVAs in file `xva.csv` which are derived from the allocated exposure profiles. Note that VRE also offers alternative allocation methods to the marginal method by Pykhtin/Rosen, which can be explored by modifying this Example.

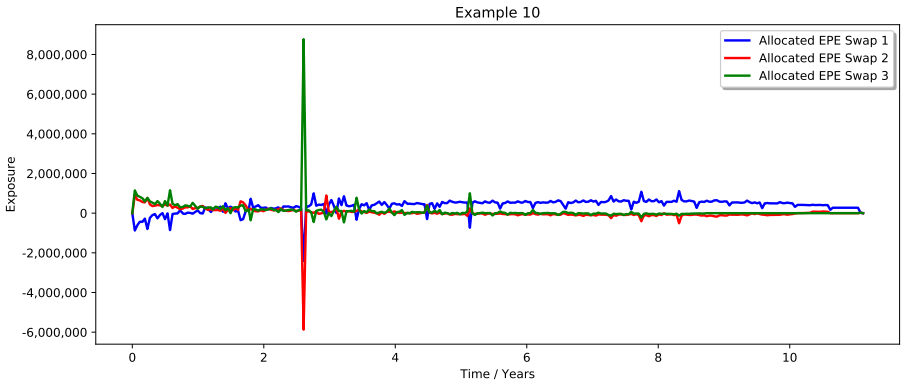

Figure 34: Exposure allocation with collateral and threshold 1m EUR. Simulation with 5000 paths and bi-weekly time steps.

9.11.2 Close-out Grid

So far we have used a "lagged" collateral approach, described at the end of the collateral section in [2], to take the Margin Period of Risk into account in exposure modelling. This used to have the disadvantage in VRE that we need to use equally-spaced time grids with time steps that match the MPoR, e.g. 2W, out to final portfolio maturity.

Calling

```
python run_closeout.py
```

we demonstrate the maintained nested-grid close-out approach, with results in folders `examples/economic/exposure/collateral/output/closeout_*`. In this approach we use two nested grids: The (almost) arbitrary main simulation grid is used to compute "default values" which feed into the collateral balance $C(t)$ filtered by MTA and Threshold etc; an auxiliary "close-out" grid, offset from the main grid by the MPoR, is used to compute the delayed close-out values $V(t)$ associated with time default time t. The difference between $V(t)$ and $C(t)$ causes a residual exposure $[V(t) - C(t)]^+$ even if minimum transfer amounts and thresholds are zero.

The close-out date value can be computed in two ways in VRE

- as of default date, by just evolving the market from default date to close-out date ("sticky date"), or
- as of close-out date, by evolving both valuation date and market over the close-out period ("actual date"), i.e., the portfolio ages and cash flows might occur in the close-out period causing spikes in the evolution of exposures.

We are reusing one case from Example 10 here, perfect CSA with zero threshold and minimum transfer amount, so that the remaining exposure is solely due to the MPoR effect. The portfolio consists of a single at-the-money Swap in GBP. The relevant configuration changes that trigger this modelling are in the published simulation contract for `input/vre_closeout_mpor.toml` as shown in Listing 1

Listing 1: Close-out grid specification: Simulation parameters

```
grid = "2W,1M,2M,3M,4M,5M,6M,7M,8M,9M,10M,11M,1Y,1Y3M,1Y6M,1Y9M,2Y,3Y,4Y,5Y,6Y,7Y,8Y,9Y,10Y,11Y"
calendar = "EUR,USD,GBP,CHF"
sequence = "SobolBrownianBridge"
scenario_generator = "Simple"
seed = 42
samples = 1000
close_out_lag = "2W"
mpor_mode = "StickyDate"
```

and moreover in the XVA analytics section of `input/vre_closeout_mpor.toml` as shown in Listing 2.

Listing 2: Close-out grid specification: XVA analytic

```
[analytics.xva]
active = "Y"
calculationType = "NoLag"
exposureProfiles = "Y"
exposureProfilesByTrade = "Y"
```

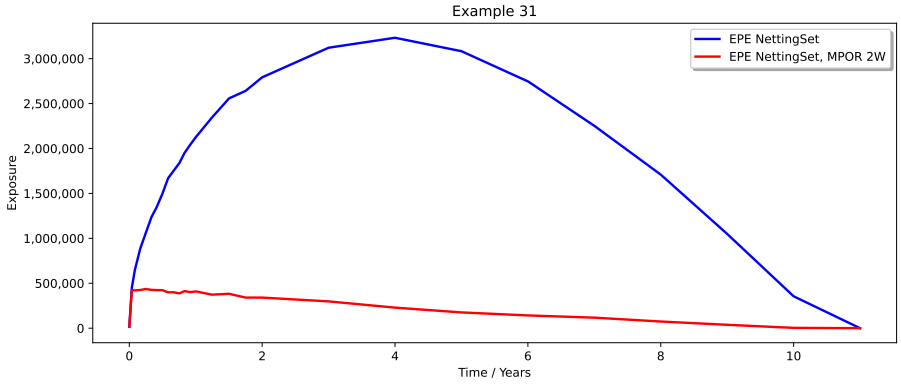

Figure 35: Uncollateralized Swap exposure vs exposure with Variation Margin, zero threshold and MTA. The simulation uses a variable grid with an auxiliary grid for closeout value calculation which is offset by the MPOR.

The resulting exposure graphs, with comparison of the uncollateralised to the collateralised case (with Variation Margin only) is shown in figure 35.

In figure 36 we have added Initial Margin based on VRE's regression model, and we compare the exposure with VM only to the exposure with both VM and IM. Figure 37 shows the related evolution of expected DIM, benchmarked against the Delta VaR DIM model, used in section 9.9.2 and described in [2].

9.11.3 First MPoR Adjustment

Calling

```
python run_firstmpor.py
```

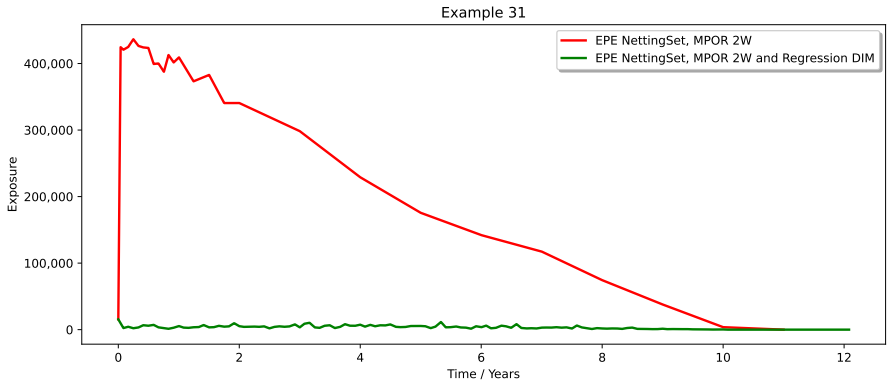

Figure 36: Comparison of Swap exposure with VM only as above vs VM+IM using first-order regression.

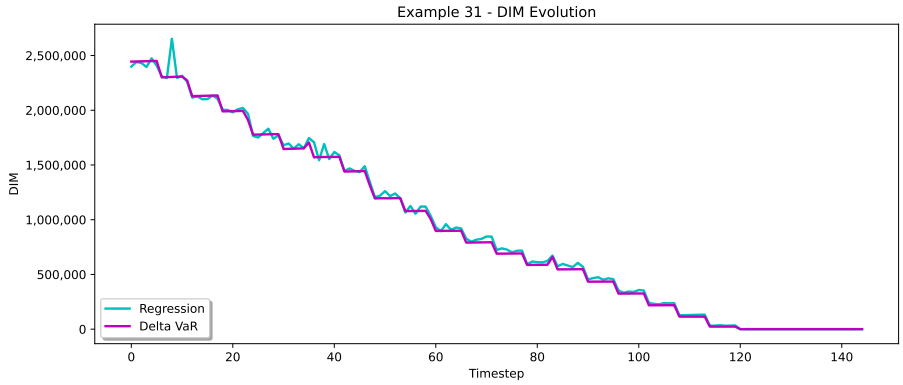

Figure 37: Evolution of expected IM from the regression model vs Delta VaR benchmark.

demonstrates a simple XVA calculation for a Receiver Swap (in netting set CPTY_A) and a Payer Swap (in netting set CPTY_B) with two initial VM balances each such that the netting set is fully collateralised resp. over/under-collateralised.

The new flag `firstMporCollateralAdjustment` in `input/vre_firstmpor.toml`'s `[analytics.xva]` block affects the first MPoR, 2 weeks in the example. If set to true, the difference between initial collateral balance and mtm is carried over during the first MPoR period, if the difference increases our exposure - a conservative measure.

9.12 Exposure Family: Methods

The method cases in `examples/economic/exposure/methods` demonstrate how to use American Monte Carlo Simulation (AMC) and related compiled-grid variants for exposure generation in VRE. This is the maintained location for the old American-Monte-Carlo example family; the legacy top-level directory name `examples/AmericanMonteCarlo` is retired.

- We start with benchmarking against "classic" exposure simulation, i.e. we run both AMC and classic simulation on a small IR/FX portfolio, almost vanilla, that consists of a Bermudan Swaption, Single and Cross Currency Swaps, FX Swap and FX Option, and we compare the resulting AMC vs. classic exposures.
  Run with `python run_benchmark.py`

- Scripted Bermudan Swaption and LPI Swap: This case shows that the scripted trade framework works with AMC too, demonstrated here with a scripted Bermudan Swaption and an LPI Swap.
  Run with: `python run_scriptedberm.py`
- FX TaRF: FxTaRF product is implemented using the scripted trade framework "under the hood" with the payoff script embedded into C++, so that it is neither explicit in the trade XML nor in the script library.
  Run with: `python run_fxtarf.py`
- Forward Bond with AMC, run with: `python run_forwardbond.py`
- Overlapping Close-Out Grid, run with: `python run_overlapping.py`
- Scenario Statistics, run with: `python run_scenariostatistics.py`

9.12.1 Benchmarking AMC vs Classic Simulation

This example demonstrates how to use American Monte Carlo simulation (AMC) to generate exposures in VRE. For a sketch of the methodology and comments on its implementation in VRE see [2]. Moreover we discuss the essential configuration changes for applying AMC.

Calling

```
python run_benchmark.py
```

performs two VRE runs, a 'classical' exposure simulation and an American Monte Carlo simulation, both on a quarterly simulation grid and for the same portfolio consisting of four trades:

- Bermudan swaption
- Single Currency Swap
- Cross Currency Swap
- FX Option

We use a 'flat' market here (yield curve and Swaption volatility surface). The number of simulation paths is 2k in the classic simulations. If not stated otherwise below, the number of training paths and simulation paths is 10k in the AMC simulations.

In the following we compare the AMC exposure profiles to those produced by the 'classic' valuation engine for each trade and the netting set.

Figure 38 shows the EPE and ENE for a Bermudan Swaption 10y into 10y in (base ccy) EUR with physical settlement. The classic run uses the LGM grid engine for valuation. We observe close agreement between the two runs. To achieve the observed agreement, it is essential to set the LGM model's mean reversion speed to zero in both

- the Bermudan Swaption pricing policy published from `input_native/vre_classic/runtime_policy/pricing_runtime_pricingengine.toml`, and
- the Cross Asset Model's IR model components in the published simulation contracts `input_native/vre_classic/runtime_policy/simulation_runtime.toml` and `input_native/vre_amc/runtime_policy/simulation_runtime.toml`

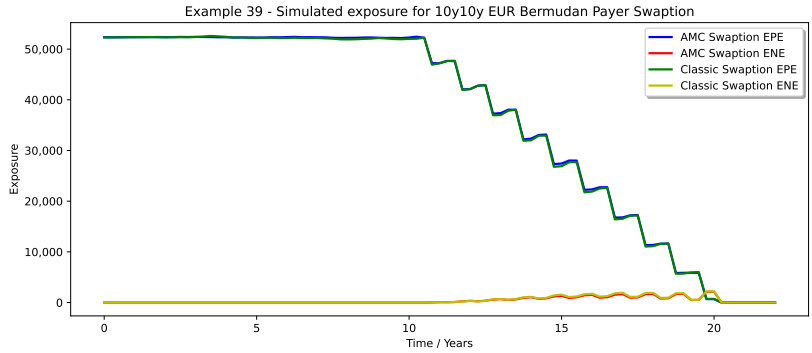

Figure 38: EPE of a EUR Bermudan Swaption computed with the classic and AMC valuation engines, using 50k training paths for the AMC simulation.

Figure 39: EPE of a USD swap computed with the classic and AMC valuation engines

and to use a high order 6 of the regression polynomials in the AMC pricing policy `input_native/vre_amc/runtime_policy/pricing_policy_pricingengine_amc.yaml`.

Figure 39 shows the EPE and ENE for a 20y vanilla Swap in USD. The currency of the amc calculator is USD in this case, i.e. it is different from the base ccy of the simulation (EUR). The consistency of the classic and amc runs in particular demonstrates the correct application of the currency conversion factor (see methodology guide). To get a better accuracy for purposes of the plot in this document we increased the number of training paths for this example to 50k and the order of the basis functions to 6.

Figure 40 shows the EPE and ENE for a 20y cross currency Swap EUR-USD.

Figure 41 shows the EPE and ENE for a vanilla FX Option EUR-USD with 10y1m expiry. For the classic run the FX volatility surface is not implied by the cross asset model but kept flat, which yields a slight hump in the profile. The AMC profile is flat on the other hand which demonstrates the consistency of the FX Option pricing with the risk factor evolution model.

Analytic Configuration

To use the AMC engine for an XVA simulation the maintained run bundle `input/vre_amc.toml` uses the following `[analytics.simulation]` block:

```
[analytics.simulation]
active = "Y"
amc = "Y"
```

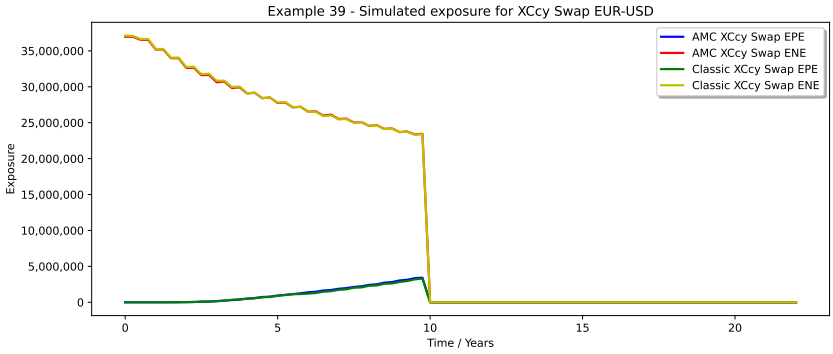

Figure 40: EPE of a EUR-USD cross currency swap computed with the classic and AMC valuation engines

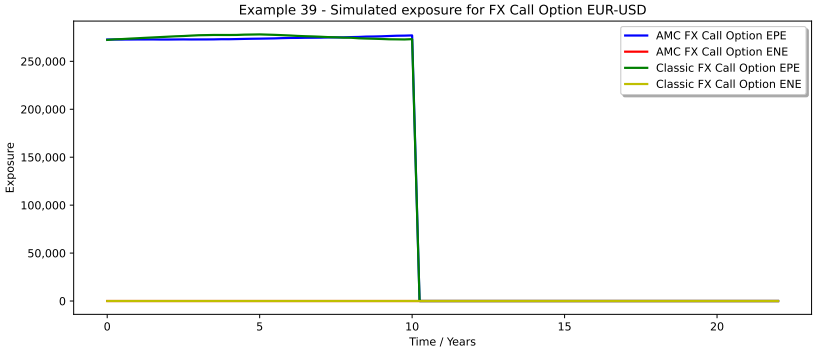

Figure 41: EPE of a EUR-USD FX option computed with the classic and AMC valuation engines

```
amcTradeTypes = "Swap,Swaption,FxOption,CompositeTrade"
simulationConfigFile = "simulation.xml"
pricingEnginesFile = "pricingengine.xml"
amcPricingEnginesFile = "pricingengine_amc.xml"
baseCurrency = "EUR"
```

The trades which have a trade type matching one of the types in the `amcTradeTypes` list, will be built against the pricing engine config provided and processed in the AMC engine. As a naming convention, pricing engines with engine type AMC provide the required functionality to be processed by the AMC engine, for technical details cf. [2].

All other trades are processed by the classic simulation engine in VRE. The resulting cubes from the classic and AMC simulation are joined and passed to the post processor in the usual way.

Note that since sometimes the AMC pricing engines have a different base ccy than the risk factor evolution model (see below), a horizon shift parameter in the simulation set up should be set for all currencies, so that the shift also applies to these reduced models.

Pricing Engine Configuration

At this point we assume that the reader is generally familiar with the configuration section 12, in particular pricing engine configuration in section in [1].

The pricing engine configuration is similar for all AMC enabled products, and is published from the native pricing-policy surface under `input_native/vre_amc/runtime_policy/pricing_policy_pricingengine_amc.yaml`. For Bermudan Swaptions the published block is:

```yaml
- productType: BermudanSwaption
  model:
    name: LGM
    parameters: {}
  engine:
    name: AMC
    parameters:
      Training.Sequence: MersenneTwisterAntithetic
      Training.Seed: "42"
      Training.Samples: "50000"
      Pricing.Sequence: SobolBrownianBridge
      Pricing.Seed: "17"
      Pricing.Samples: "0"
      Training.BasisFunction: Monomial
      Training.BasisFunctionOrder: "6"
      BrownianBridgeOrdering: Steps
      SobolDirectionIntegers: JoeKuoD7
      MinObsDate: "true"
      RegressionOnExerciseOnly: "false"
```

The `Model` differs by product type, table 17 summarises the supported product types and model and engine types. The engine parameters are the same for all products:

1. `Training.Sequence`: The sequence type for the training phase, can be `MersenneTwister`, `MersenneTwisterAntithetc`, `Sobol`, `Burley2020Sobol`, `SobolBrownianBridge`, `Burley2020SobolBrownianBridge`
2. `Training.Seed`: The seed for the random number generation in the training phase
3. `Training.Samples`: The number of samples to be used for the training phase

4. Pricing.Sequence: The sequence type for the pricing phase, same values allowed as for training
5. Training.BasisFunction: The type of basis function system to be used for the regression analysis, can be Monomial, Laguerre, Hermite, Hyperbolic, Legendre, Chbyshev, Chebyshev2nd
6. BasisFunctionOrder: The order of the basis function system to be used
7. Pricing.Seed: The seed for the random number generation in the pricing
8. Pricing.Samples: The number of samples to be used for the pricing phase. If this number is zero, no pricing run is performed, instead the (T0) NPV is estimated from the training phase (this result is used to fill the T0 slice of the NPV cube)
9. BrownianBridgeOrdering: variate ordering for Brownian bridges, can be Steps, Factors, Diagonal
10. SobolDirectionIntegers: direction integers for Sobol generator, can be Unit, Jaeckel, SobolLevitan, SobolLevitanLemieux, JoeKuoD5, JoeKuoD6, JoeKuoD7, Kuo, Kuo2, Kuo3
11. MinObsDate: if true the conditional expectation of each cashflow is taken from the minimum possible observation date (i.e. the latest exercise or simulation date before the cashflow's event date); recommended setting is true
12. RegressionOnExerciseOnly: if true, regression coefficients are computed only on exercise dates and extrapolated (flat) to earlier exercise dates; only for backwards compatibility to older versions of the AMC module, recommended setting is false

Product Type	Model	Engine
Swap	CrossAssetModel	AMC
CrossCurrencySwap	CrossAssetModel	AMC
FxOption	CrossAssetModel	AMC
BermudanSwaption	LGM	AMC
MultiLegOption	CrossAssetModel	AMC

Table 17: AMC enabled products with engine and model types

Additional Features

As a side product the AMC module provides plain MC pricing engines for Bermudan Swaptions and a new trade type MultiLegOption with a corresponding MC pricing engine.

MC pricing engine for Bermudan swaptions

The maintained native pricing-policy surfaces use the same decomposition for a dedicated MC Bermudan Swaption engine:

- the model remains LGM with the same calibration family as the grid setup (Calibration, CalibrationStrategy, reversion, volatility, shift-horizon and tolerance controls);

- the engine switches from grid valuation to MC valuation while keeping the same training/pricing sequence family documented in 9.12.1;
- the MC variant adds an explicit pricing-sample count for the forward pricing phase, whereas the AMC bundle typically uses `Pricing.Samples = 0` because the training phase fills the T0 cube slice.

In other words, the maintained native surface no longer relies on an XML product block here; the same pricing-policy shape is published through the resolved pricing policies and uses the same parameter families as the AMC surfaces already shown above.

Multi Leg Options / MC pricing engine

The following listing shows a sample MultiLegOption trade. It consists of

1. an option data block; this is optional, see below
2. a number of legs; in principle all leg types are supported, the number of legs is arbitrary and they can be in different currencies; if the payment currency of a leg is different from a floating index currency, this is interpreted as a quanto payoff

If the option block is given, the trade represents a Bermudan swaption on the underlying legs. If the option block is missing, the legs themselves represent the trade.

See [2] for limitations of the multileg option pricing engine.

```
<Trade id="Sample_MultiLegOption">
  <TradeType>MultiLegOption</TradeType>
  <Envelope>...</Envelope>
  <MultiLegOptionData>
    <OptionData>
      <LongShort>Long</LongShort>
      <OptionType>Call</OptionType>
      <Style>Bermudan</Style>
      <Settlement>Physical</Settlement>
      <PayOffAtExpiry>false</PayOffAtExpiry>
      <ExerciseDates>
        <ExerciseDate>2026-02-25</ExerciseDate>
        <ExerciseDate>2027-02-25</ExerciseDate>
        <ExerciseDate>2028-02-25</ExerciseDate>
      </ExerciseDates>
    </OptionData>
    <LegData>
      <LegType>Floating</LegType>
      <Payer>false</Payer>
      <Currency>USD</Currency>
      <Notionals>
        <Notional>100000000</Notional>
      </Notionals>
      ...
    </LegData>
    <LegData>
      <LegType>Floating</LegType>
      <Payer>true</Payer>
      <Currency>EUR</Currency>
      <Notionals>
        <Notional>100000000</Notional>
      </Notionals>
      ...
    </LegData>
  </MultiLegOptionData>
</Trade>
```

The pricing engine configuration is similar to that of the MC Bermudan swaption engine, cf. 9.12.1. In the maintained native policy surface this appears as:

```
- productType: MultiLegOption
  model:
    name: CrossAssetModel
    parameters: {}
  engine:
    name: AMC
    parameters:
      Training.Sequence: MersenneTwisterAntithetic
      Training.Seed: "42"
      Training.Samples: "10000"
      Pricing.Sequence: SobolBrownianBridge
      Pricing.Seed: "17"
      Pricing.Samples: "0"
      Training.BasisFunction: Monomial
      Training.BasisFunctionOrder: "4"
      BrownianBridgeOrdering: Steps
      SobolDirectionIntegers: JoeKuoD7
      MinObsDate: "true"
      RegressionOnExerciseOnly: "false"
```

Model Parameters special to that product are

1. `IrCalibrationStrategy` can be `None`, `CoterminalATM`, `UnderlyingATM`
2. `FXCalibration` can be `None` or `Bootstrap`
3. `ExtrapolateFxVolatility` can be `true` or `false`; if false, no calibration instruments are used that require extrapolation of the market fx volatility surface in option expiry direction
4. `Corr_Key1_Key2`: These entries describe the cross asset model correlations to be used; the syntax for `Key1` and `Key2` is the same as in the simulation configuration for the cross asset model

9.12.2 Scripted Bermudan

Calling

```
python run_scriptedberm.py
```

demonstrates exposure simulation using AMC for selected scripted trade types

- Bermudan Swaption
- LPI Swap

Both payoffs are defined in the `scriptlibrary.xml` which are referenced in `portfolio_scriptedberm.xml` by the `ScriptName` tag.
To enable the AMC processing the maintained run bundle `input/vre_scriptedberm.toml` uses the following highlighted settings.

```
[analytics.simulation]
active = "Y"
amc = "Y"
simulationConfigFile = "simulation.xml"
pricingEnginesFile = "pricingengine.xml"
amcPricingEnginesFile = "pricingengine_amc.xml"
amcTradeTypes = "ScriptedTrade"
baseCurrency = "EUR"
```

```
cubeFile = "cube.csv.gz"
aggregationScenarioDataFileName = "scenariodata.csv.gz"
aggregationScenarioDataDump = "scenariodata.csv"
```

Note that VRE can handle a mix of trades covered by AMC simulation and covered by "classic" simulation. The respective NPV cubes are combined before generating results such as exposures or XVAs.

9.12.3 Scripted TaRF

Calling

```
python run_fxtarf.py
```

demonstrates exposure simulation and XVA for another scripted product, an FX Target Redemption Forward (TaRF). In contrast to the cases presented above, you won't see the payoff script library in the Input folder, nor is the script embedded into the trade XML file. The trade type in this case is `FxTARF` which has its own implementation in `vre_data/vred/portfolio/tarf.xpp` and a separate trade schema. However, the scripted trade framework is used under the hood, and the payoff script is embedded into the C++ code in vre_data/vred/portfolio/tarf.cpp.

9.12.4 Forward Bond

Calling

```
python run_forwardbond.py
```

demonstrates AMC exposure simulation and XVA for a forward and a plain vanilla bond. The results can be compared to a classical exposure simulation by switching from `input/vre_forwardbond.toml` to a classic benchmark bundle, or equivalently by setting the run bundle's `amc` flag to `"N"` for a local comparison.

For both cases, forward and plain vanilla bond, the security spread is implied from given bond prices. This requires a proper security specification in the curve configuration. Compare the maintained *Securities* node within the published curve-set contract. Both fields, PriceQuote and SpreadQuote, are mandatory. In the market data file, if the price quote is given in the absence of a spread quote, the spread is implied. Otherwise the spread quote is used or none.

In case of a forward bond, the convention for security specification requires the format *securityId_FWDEXP_expiryDate*.

9.12.5 Overlapping Close-out Grids in AMC

Calling

```
python run_overlapping.py
```

demonstrates the case where the primary and close-out grid overlap, using a daily simulation grid up to 15 days. This is supported in the current VRE release line and is useful when close-out timing has to be resolved at a finer granularity than the main exposure grid.

9.12.6 Extract Scenario Statistics

Calling

```
python run_scenariostatistics.py
```

demonstrates a basic statistical analysis of the simulated raw scenario data (simulated yield curve discount factors, FX spot rates etc), generating a report with min, mean, max, standard deviation, skewness and kurtosis for each factor and at each simulated future date.

9.13 XVA Risk

The examples in `examples/economic/xva` are focused on **economic (non-regulatory)** XVA workflows: stress testing, explain and sensitivity analysis. They can be run individually as discussed below, or together with:

```
python run.py
```

The aggregate runner executes `run_stress.py`, `run_xvaexplain.py` and `run_sensi.py` with deterministic sample defaults for reproducible smoke runs.

9.13.1 XVA Stress Testing

Calling

```
python run_stress.py
```

demonstrates XVA stress testing with both classic and AMC XVA engines. The script runs `input/vre_stress_classic.toml` and `input/vre_stress_amc.toml` and writes results to `examples/economic/xva/output/stress/classic` and `examples/economic/xva/output/stress/amc`.

The analytic type `xvaStress` uses the published stress-pack surface in `input_native/vre_stress_classic/runtime_policy/stress_pack.yaml` and supports both zero- and par-domain market shocks.

Key outputs are `xva.csv`, trade/netting-set exposure reports and runtime logs per scenario set. Like the other XVA analytics, stress testing evaluates XVA on a simulation market rather than the static today market; enabling `useSpreadedTermStructures` in the stress pack is recommended for production-like par-domain stress behaviour.

9.13.2 XVA P&L Explain

Calling

```
python run_xvaexplain.py
```

demonstrates the XVA explain analytic with results in `examples/economic/xva/output/explain`.

VRE can compute the market implied XVA change between two evaluation dates. For each risk factor defined in the sensitivity config VRE computes the par rate change between t0 and t0 + mporDays. VRE derives for each risk factor a shift scenario $(ParRate(t_1) - ParRate(t_0))$ and computes the CVA change implied by those risk factor shifts at t0.

Key outputs include `xvaExplain.csv`, `xvaExplain_details.csv`, `xvaExplain_summary.csv`, and generated stress scenario diagnostics (`xvaExplain_parStressTest.xml`, `xvaExplain_zeroStressTest.xml`).

Like XVA stress, XVA explain replaces the today market with a simulation market during evaluation. It is recommended to activate `use_spreaded_term_structures` in `input_native/vre_explain/runtime_policy/sensitivity_runtime.toml` and include relevant volatility factors in the published scenario market profile.

9.13.3 XVA Sensitivities

Calling

```
python run_sensi.py
```

demonstrates the XVA sensitivity analytic with results in `examples/economic/xva/output/sensi/classic` and `examples/economic/xva/output/sensi/amc`.

The analytic computes XVA sensitivities by bump & revalue and generates the sensitivity scenarios from a sensitivity configuration similar to the one used in NPV sensitivity calculation. It utilises the existing XVA machinery "under the hood", supports XVA calculation using "classic" and AMC simulation. Sensitivities are generated in both the "raw" (e.g. zero rate) domain as well as in the par domain.

This analysis is run twice (classic and AMC) for a small two-swap uncollateralised reference portfolio. Key outputs include:

- `xva.csv`
- `xva_par_sensitivity_cva.csv`, `xva_zero_sensitivity_cva.csv`
- equivalent DVA/FBA/FCA sensitivity files
- Jacobian diagnostics (`xva_sensi_jacobi.csv`, `xva_sensi_jacobi_inverse.csv`)

For methodology summaries and the current scope of the implementations see [2].

Note:

- The XVA sensitivity analytic replaces the TodaysMarket with a SimulationMarket during the XVA calculation. This is a template for extending other VRE analytics (such as VaR) for exposing their evaluation under stressed markets.
- Realistic portfolios may require large scenario sets and simulation loads; AMC usually provides better runtime scaling than classic simulation in this context.
- For accelerated AAD/GPU workflows, see the Performance section 9.15 and the regulatory parity workflows in sections 9.7.11 and 9.8.4.

9.13.4 Regulatory CVA Capital Moved to CVA Risk

The SA-CVA and BA-CVA capital examples were moved out of `XvaRisk` and are now documented in section 9.8 (`examples/regulatory/pillar_1/credit/cva`). This preserves the economic vs regulatory split in the examples chapter:

- `XvaRisk`: economic XVA analytics
- `CvaRisk`: regulatory CVA capital (MAR50 / DIS51 evidence)

9.14 Credit Risk

The Credit Risk examples in `examples/regulatory/pillar_1/credit` focus on CRE and DIS-oriented counterparty credit workflows:

- SA-CCR capital and disclosure routes
- IMM exposure and IMM disclosure routes
- Credit Portfolio Model (CPM) loss-distribution examples

This directory is the current counterparty-credit companion to the CVA and reporting chapters. The authored bundles do not merely reproduce canonical benchmark numbers; they show how the same calculations fit into legal-entity routing, disclosure production, scope evidence and current runtime-publication controls.

Run the smoke suite with `python run.py` (which calls `run_cre_regression_smoke.py`). For the same full script list via explicit wrapper use `python run_cre_full.py`.

9.14.1 SA-CCR

Calling

```
python run_saccr.py
```

demonstrates SA-CCR coverage across BIS examples, CRE FAQ/margining variants and DIS routes. The runner executes a matrix of configurations including:

- BIS examples 1-5 (`vre_saccr_BIS_*.toml`)
- FAQ1/FAQ2, MA split/group, CCP overlay/client, multi-asset and outside-collateral variants
- IRB disclosure route (`vre_saccr_irb.toml`)
- CCR8 matrix route (`vre_saccr_CCR8_matrix.toml`)

The authored SA-CCR bundles are current native packs: each one binds a legal entity, market snapshot, resolved counterparty / netting / legal-entity publications, and the relevant SA-CCR overlays into one deterministic runtime bundle. The example portfolios cover the canonical BIS swaps-and-options examples, but they also extend into netting-set splits, margin-agreement grouping, CCP treatment, multi-asset allocation and outside-collateral cases. That broader estate is important because it shows how the same core SA-CCR analytic responds to structurally different collateral and legal-agreement arrangements rather than only to different trade mixes.

The runner matrix mirrors the main supervisory cases in the CRE text and FAQ estate: baseline BIS examples 1–5, FAQ1/FAQ2 exceptions, margin-agreement split/group cases, CCP overlay and CCP client pass-through, multi-asset netting, outside-collateral treatment, and the IRB / CCR8 disclosure routes. That makes the suite useful both for methodology onboarding and for regression control.

Committed reference outputs and disclosure artefacts are maintained under `examples/regulatory/pillar_1/credit/ccr/sa_ccr/expected_output` (with subfolders `IRB` and `CCR8_matrix` for route-specific runs). Key outputs include:

- summary and netting-set outputs: `saccr_output_*.csv`, `saccr_netting_set_results.csv`
- allocation/evidence outputs: `saccr_ccr_exposure_facts.csv`, `saccr_rwa_method_pairs.csv`
- DIS42 disclosure files: `dis42_ccr5_saccr.csv`, `dis42_ccr6_saccr.csv`, `dis42_ccr8_saccr.csv`
- IRB route output: `dis42_ccr4_saccr_irb.csv`

Additional maintained evidence includes `saccr_margin_agreement_links.csv`, `ccp_qualification_resolution_facts.csv`, `cva_capital_perimeter_resolution_facts.csv`, scope evidence JSON files, and resolved run-bundle publications. These are part of the example contract, not incidental diagnostics. See [2] for the current implementation scope.

9.14.2 IMM

Calling

```
python run_imm.py
```

demonstrates IMM exposure and disclosure workflows for the CRE53 benchmark set:

- canonical OTC swap baseline: `input/vre_imm_swap.toml`
- method matrix: classic (`vre_imm_otc_classic.toml`), AMC (`vre_imm_otc_amc.toml`), CG (`vre_imm_otc_cg.toml`), CG/AAD (`vre_imm_otc_cg_aad.toml`)
- IRB route variant: `vre_imm_otc_cg_irb.toml`

These IMM runs are especially valuable because they isolate method choice over a common regulatory portfolio and common runtime publications. The bundled portfolio focuses on OTC swaps and related scripted trade payloads so that the comparison between classic, AMC, CG and CG/AAD remains numerically meaningful. The authored bundles then vary the simulation mode, pricing-engine policy, compiled CE controls and disclosure route without changing the economic book. That makes the suite both a methodology example and a backend-validation example.

The CRE53 method matrix is the core maintained example. The canonical swap baseline anchors the exposure profile, while the OTC benchmark portfolio is reused across classic, AMC, CG and CG/AAD so that changes in EPE, EEPE, disclosure files or runtime diagnostics can be attributed to method choice rather than to changing economics. The IRB variant then extends the same book into the alternative disclosure route without changing the portfolio.

Committed reference outputs are maintained under `examples/regulatory/pillar_1/credit/ccr/imm/expected_output/[variant]`

(for example swap, otc_classic, otc_amc, otc_cg, otc_cg_aad, otc_cg_irb). Key outputs include:

- imm_exposure.csv
- DIS42 disclosure files: dis42_ccr1_imm.csv, dis42_ccr7_imm.csv, dis42_ccr8_imm.csv
- IRB route file: dis42_ccr4_imm_irb.csv
- route evidence files such as imm_ccr_exposure_facts.csv and imm_rwa_method_pairs.csv

For the compiled methods the output folders also include the artefacts needed to audit the exposure engine itself, including higher-of evidence, selection-evidence facts and compiled exposure diagnostics where applicable. The IMM suite should therefore be read both as a regulatory exposure example and as a controlled methodology-comparison harness.

Backend coverage in this suite is classic/AMC/CG/CG-AAD. No dedicated GPU IMM runner is packaged here.

9.14.3 Credit Portfolio Model

The purpose of the credit portfolio model in VRE is to generate an integrated portfolio gain/loss distribution at a given future horizon which is driven by

- credit defaults and rating migrations in Bonds and CDS, and
- the PnL of a portfolio of derivatives over the specified time horizon.

The model integrates Credit and Market Risk by jointly evolving systemic credit risk drivers alongside the usual risk factors in VRE's Cross Asset Model. See also the separate documentation in docs/user_guide/creditmodel.tex|pdf. As mentioned there, this example will only run if the Eigen installation has been done before building VRE as described in section 8.4.

By running

```
python run_cpm.py
```

this example demonstrates the model's outcome for seven demo portfolios

Case	Credit Mode	Exposure Mode	Evaluation
Single Bond	Migration	Value	Analytic
Bond and Swap	Migration	Value	Analytic
3 Bonds	Migration	Value	Analytic
10 Bonds	Migration	Value	Analytic
10 Bonds	Migration	Value	Terminal Simulation
Bonds and CDS	Migration	Notional	Analytic
100 Bonds	Default	Notional	Analytic

The last demo case in this table can be activated by uncommenting the corresponding section at the end of the run_cpm.py script.

9.15 Performance

9.15.1 Multi-threading

Calling

```
python run_multithreading.py
```

demonstrates the parallelisation of exposure simulation for a portfolio of eight Vanilla Swap copies, results in folder `examples/performance/output/multi`. The multi-threaded valuation engine in VRE splits the portfolio into n parts where n is the number of threads `nThreads` specified in the maintained multi-threading run bundle.

9.15.2 NPV Sensitivities with AAD and GPUs

Calling

```
python run_sensi.py
```

demonstrates alternative ways of speeding up sensitivity calculations - using AAD or an external compute device. The test portfolio consists of

- Vanilla Equity Option
- Equity Barrier Option
- Equity Accumulator
- Asian Basket Option
- FX TaRFs

The sensitivity analysis is then run in four ways, see `run_sensi.py`,

- with "classic" bump and revalue ((19 sec on Apple M2 Max))
- as above but using the Computation Graph, enabled in the maintained CG pricing policy, which is the basis for the following two approaches ((14 sec on Apple M2 Max))
- using AAD, via the maintained AAD pricing policy ((2.2 sec on Apple M2 Max))
- using the external device if available, via the maintained GPU pricing policy (2.3 sec on Apple M2 Max with "OpenCL/Apple/Apple M2 Max" device)

to compare sensitivities and performance. In the latter case we have set the external device in the GPU pricing policy to "BasicCpu/Default/Default" which mimics an external device on the CPU. On a macbook pro (2023) with M2 Max processor, we can also choose "OpenCL/Apple/Apple M2 Max" here (a 38 core GPU). The Jupyter notebook `vre_aadsensi.ipynb` in this `examples/performance` folder also kicks off these four runs, but adds further commentary and visualises results. To run this notebook you need the current Python bindings, either from a local source build or by "pip install" of VRE as discussed in section 9.16.

9.15.3 CVA Sensitivities with AAD

Calling

```
python run_cvasensi.py
```

demonstrates a prototype CVA sensitivity calculation applying AAD to a single Swap instrument with results in folder `examples/performance/output/cvasensi`.

This script executes four batches

- a "base" CVA calculation using AMC for reference (about 9 sec on Apple M2 Max)
- a bump & revalue CVA sensitivity calculation (about 48 sec on Apple M2 Max)
- CVA sensitivity using AAD (about 2 sec on Apple M2 Max)
- CVA sensitivities with GPU parallelisation, porting the conditional expectation calculations to the external device is work in progress, hence no noticeable speedup to be reported yet

Run the corresponding Juypter notebook with: `python -m jupyterlab vre_cvasensi.ipynb &`

9.16 VREPython

In the VRE 0.12.0 release line we provide easy access to VRE via a pre-compiled Python module. Some example scripts using this VRE module are provided in this example, so change to this directory first

```
cd examples/vre_python
```

The examples require Python 3. The VRE Python module is then installed with a one-liner, see step 3 below. However, to separate VRE from any other Python environments on your machine, we recommend creating a virtual environment first. In that case the steps are as follows.

1. To create a virtual environment: `python -m venv env1`
2. To activate this environment on Windows: `.\env1\Scripts\activate.bat` or on macOS/Linux: `source env1/bin/activate`
3. Then install the latest release of VRE: `pip install vannarho-risk-engine`
4. Try examples:
   - `python vre.py`
     This demonstrates the Python-wrapped version of the VRE application that is also used in the command line application `vre.exe`. We use it here to re-run the swap exposure from section 9.10.1.
   - `python ore2.py`
     This extends the previous example and shows how to access and post-process VRE in-memory results in the Python framework without reading files.
   - `python commodityforward.py`
     The VRE Python module also allows lower-level access to the QuantLib and QuantExt libraries, demonstrated here for a CommodityForward instrument defined in QuantExt. Note that the VRE Python module contains the entire QuantLib Python functionality.

More use cases of the VRE Python module, including lower-level QuantLib access and notebook-style workflows, can be found in the current repository under `vre_python/examples` and `examples/vre_python/notebooks`.

5. You can deactivate the environment with `deactivate`
   or even fully remove the environment again by removing the `env1` folder.

Finally, you can build the Python module and installable packages yourself following the instructions in sections 8.3 based on your local VRE code.

Jupyter Notebook Examples

The current VRE repository ships a maintained notebook set under `examples/vre_python/notebooks`, including `example_1` through `example_11`.

To try, change e.g. to `examples/vre_python/notebooks/example_1` and launch the notebook there by calling

```
python3 -m jupyterlab vre.ipynb &
```

If not installed yet, this might require a

```
python3 -m pip install jupyterlab
```

and some of the Jupyter notebooks require additional packages to be installed with pip:

- matplotlib
- pandas
- numpy
- scipy
- ipywidgets

10 Current Parameterisation And Runtime Contracts

The current parameterisation surface is best understood as a layered contract system rather than as a single master XML file. This is the most important conceptual change for users coming from older VRE releases and the basis for the parameterisation chapter that follows.

Current Layer Model

Contract layer	Format	Role
Authoring layer	`TOML`	Human-authored run bundles and runtime knobs
Curated specification layer	`YAML`	Slow-moving policy, registry and mapping documents
Reference / runtime machine layer	`JSON`	Resolved bundles, manifests, evidence, published machine-readable contracts
Runtime-ready package root	`input_native/` trees	Package assembled for deterministic engine loading
Compatibility layer	retained `XML`	Existing engine readers, selected payload families and migration bridges

Table 18: Current parameterisation layers.

Current User Workflow

The normal workflow is now increasingly:

1. author or select a run bundle,
2. resolve or publish any registry and policy material needed by that bundle,
3. assemble or consume the runtime-ready package under `input_native`,
4. execute the run and inspect both reports and machine-readable evidence.

The detailed XML family chapters are now preserved separately under `docs/user_guide/legacy_parameterisation/`. The maintained guide now uses native/current chapters for the active configuration surface and keeps the legacy material as archival reference.

Retained XML Exceptions

Not every XML family is being removed at once, and not every XML family should be removed at once.

In particular:

- portfolio and trade payloads remain XML,
- script-library payloads remain XML,
- example and regression continuity still requires some XML compatibility packs,
- some migrated bundles still carry XML compatibility artifacts while the runtime seam is being removed family by family.

The practical reading is therefore:

- current source-of-truth direction is ADR/native,
- XML remains supported where the runtime or payload boundary still requires it,

- XML detail is preserved below for reference, not because XML is still the design target for all touched families.

11 Legacy XML Reference Status

The XML-era parameterisation chapters that previously occupied this part of the guide have now been archived under `docs/user_guide/legacy_parameterisation/`. They are retained deliberately so that no engineering detail is lost during the ADR/native migration.

They should, however, be read with the following status in mind:

- they are a preserved *compatibility reference*,
- they are not the preferred authoring model for newly redesigned configuration families,
- where current ADR/native chapters describe an equivalent maintained surface, the current chapters take precedence for new work,
- for older XML-era example narratives and historical walkthroughs, users should consult the preserved older user-guide release line (for example the v0.8.x guide) rather than treat historical XML examples as the maintained corpus.

This split is intentional. VRE must remain understandable to existing users while still documenting the current engine as it actually exists today.

12 Parameterisation

The current VRE parameterisation surface is no longer organised around a single master XML file. The maintained surface is a layered contract system in which authored run bundles, curated registry seeds, published machine contracts and runtime-ready package roots each play a distinct role.

For the current guide, the correct mental model is:

1. an operator selects or authors a run bundle in `TOML`,
2. curated slow-moving policy and registry seeds are maintained in `YAML` or controlled `JSON`,
3. publication tooling resolves these sources into machine contracts under `input_native` and `published/`,
4. VRE consumes the runtime-ready package, with compatibility XML emitted only where a retained reader or payload boundary still requires it.

Layer	Primary format	Role
Run selection	`TOML`	Operator-authored run bundle, scope bundle, and runtime selector input
Curated policy and registry seeds	`YAML` / controlled `JSON`	Slow-moving source material for reporting scope, rulebooks, conventions, market manifests, SIMM packages and reference registries
Published machine contracts	`JSON`	Deterministic reference and runtime publications with lineage, hashes and resolved joins
Runtime package root	`input_native/` tree	File layout passed to the engine and accompanying tooling
Compatibility boundary	generated `XML`	Legacy readers and retained payload families only

Table 19: Current VRE parameterisation layers.

Archive policy

The XML-era parameterisation chapters have not been deleted. They are preserved verbatim under `docs/user_guide/legacy_parameterisation/` so that no engineering detail is lost during the rewrite. The current chapter replaces them as the maintained narrative and uses the new native surfaces wherever a current surface exists.

XML exception policy

In the maintained guide, XML snippets are no longer used to explain current engine-control surfaces. The explicit exceptions are:

- portfolio / trade payloads, which remain XML,
- scripted-trade payloads, which remain XML,
- archived legacy material, which remains available as historical reference.

Every other family in this chapter is documented using the current authored or published contract that operators and tooling should use now.

12.1 Analytics: `vre.toml` and resolved run bundles

The master input file is a native run-bundle surface authored in `TOML` and published into a machine-readable runtime bundle under `input_native` in `examples/`. The run bundle determines

- the as-of date and runtime package root,
- the market-data and fixing inputs,
- the resolved market, curve, pricing, netting, counterparty and reference contracts,
- the selected market configurations for pricing, simulation and calibration,
- the analytics that are active for the run.

The important change is that the master selector now points predominantly to published JSON contracts and resolved registries. The authoritative current surface is therefore a pair:

- a human-authored source bundle such as `examples/getting_started/00_minimal_setup/input/vre.toml`;
- a resolved machine bundle such as `examples/regulatory/pillar_1/credit/cva/migrated_examples/input_native/vre_sacva_cam_ext_amc_credit/runtime_policy/resolved_run_bundle.json`.

The source bundle remains the normal authoring surface. The resolved JSON bundle is the runtime publication used by validators, provenance checks and package assembly. Where a maintained pack still has a compatibility lineage, the resolved publication also carries provenance fields such as `sourceConfig` so that pack assembly and review remain auditable.

Setup

Listing 3 shows the current maintained authoring pattern.

Listing 3: Current run-bundle setup block

```
[setup]
asofDate = "2016-02-05"
executionMode = "Regulatory"
runLegalEntityId = "LE_CVA_IMM"
inputPath = "input_native/vre_sacva_cam_ext_amc_credit"
outputPath = "Output/sacva_cam_ext_amc"
logFile = "log_cam_ext_amc_credit.txt"
logMask = "65535"
marketDataFile = "market_20160205.txt"
fixingDataFile = "fixings_20160205.txt"
implyTodaysFixings = "N"
curveConfigFile = "runtime_policy/published/resolved_curve_configuration.json"
conventionsFile = "reference_registry/published/convention_manifest.json"
marketConfigFile = "runtime_policy/published/resolved_todays_market.json"
pricingEnginesFile = "runtime_policy/published/pricingengine/resolved_pricing_policy.json"
portfolioFile = "portfolio_fd_eqcom.xml"
counterpartyFile = "runtime_policy/published/resolved_counterparty_information.json"
nettingSetManagerFile = "runtime_policy/published/resolved_netting_set_information.json"
saccrCsaFile = "runtime_policy/published/resolved_netting_set_information.json"
legalEntityMasterFile = "runtime_policy/published/resolved_legal_entity_master.json"
scriptLibrary = "scriptlibrary_master.xml"
referenceDataFile = "reference_registry/published/resolved_reference_data.json"
observationModel = "Disable"
nThreads = "1"
```

The maintained setup keys fall into the following groups.

12.1.0.1 Run identity and packaging

- `asofDate`: the valuation or reporting date for the run.
- `inputPath`: the runtime package root. In the native model this is typically an `input_native/<bundle>` tree.
- `outputPath`: output root for reports, evidence and generated runtime artifacts.
- `executionMode`: current examples use *Regulatory* when legal-entity and scope enforcement is required. Diagnostic and compatibility-oriented runs keep the looser execution semantics.
- `runLegalEntityId`: the legal entity in scope for regulatory runs. This is the canonical run-scope selector and should agree with the legal-entity master, counterparty data and scope/reporting overlays.
- `useAnalytics`: optional run-level filtering switch where a launcher needs to restrict the active analytic subset.

12.1.0.2 Market and fixing data

- `marketDataFile`: market snapshot used for curve building and pricing.
- `fixingDataFile`: fixing history used by coupons, option settlement and path-dependent trades.

- `dividendDataFile`: optional dividend input when the run requires equity dividend curves.
- `marketDataMappingFile`, `fixingDataMappingFile`: optional mapping layers when source-system names must be normalised before loading.
- `implyTodaysFixings`: whether today's fixing should be inferred rather than read directly from the fixing file.
- `enrichIndexFixings`: whether sparse fixing series should be gap-filled according to the current fixing enrichment rules.
- `fixingCutoff`, `ignoreFixingLead`, `ignoreFixingLag`: advanced controls for fixing use in backfilling and operational cut-off handling.

12.1.0.3 Resolved runtime contracts

- `curveConfigFile`: published resolved curve configuration.
- `conventionsFile`: published convention manifest, normally under `reference_registry/published`.
- `marketConfigFile`: published today's-market contract used to resolve pricing and simulation market views.
- `pricingEnginesFile`: published pricing-policy contract for the selected engine set.
- `referenceDataFile`: resolved reference-data publication.
- `counterpartyFile`: resolved counterparty and regulatory classification contract.
- `nettingSetManagerFile`: resolved netting-set publication.
- `collateralBalancesFile`: current collateral-balance state when required at setup level.
- `saccrCsaFile`: SA-CCR-specific netting or CSA contract; in the current native examples this typically points at the same resolved netting-set publication as `nettingSetManagerFile`.
- `legalEntityMasterFile`: resolved legal-entity master used by scope filtering, reporting and regulatory capital analytics.
- `approvalFile`, `methodAssignmentFile`, `tradingDeskFile`: scope/reporting overlays used by Basel end-game market-risk and mixed-method capital runs.

12.1.0.4 Portfolio payloads and retained XML exceptions

- `portfolioFile`: trade payload. Portfolio and trade data remain XML in the maintained guide.
- `scriptLibrary`: scripted-trade payload. Script library content also remains XML in the maintained guide.
- `inputPathPortfolio`, `portfolioRecurseIntoSubDirectories`: optional portfolio-loading controls for large portfolio estates.

12.1.0.5 Calendar and currency extensions

- `calendarAdjustment`: additional calendar holidays and business days.
- `currencyConfiguration`: optional external currency extension surface for currencies not hard-coded in the QuantLib / QuantExt layer.
- `dateCalendar`, `dateConvention`, `accrualDate`: ancillary date controls used in specialised workflows and migration utilities.

12.1.0.6 Operational behaviour

- `observationModel`: notification and observer strategy used in simulation-heavy runs. The current examples continue to use *Disable* for the most aggressive notification suppression.
- `lazyMarketBuilding`: optional lazy construction of market objects.
- `continueOnError`: whether the run continues after recoverable errors.
- `buildFailedTrades`: whether failed trade builds should be replaced by dummy failed-trade placeholders.
- `defaultMappingToIdentity`: optional fallback mapping behaviour when an explicit mapping layer is absent.
- `nThreads`: CPU thread count used by the analytics that support parallel execution.
- `scratchDiagnostics`, `csvCommentReportHeader`: optional diagnostic and report-annotation controls.

12.1.1 Logging

The historical XML surface had a dedicated `Logging` section. In the current maintained run bundle, the commonly used logging controls remain part of `[setup]`:

- `logFile`: the main log file name.
- `logMask`: the logging bit mask controlling verbosity.

The current examples rely on these two keys. The resolved run-bundle publication preserves whatever logging controls were selected at authoring time, so provenance and replay remain deterministic even while the dedicated logging surface is still being rationalised.

Markets

The `[markets]` block selects the named market configurations to be used for pricing, simulation and model calibration. A representative block is:

Listing 4: Current market-selection block

```
[markets]
lgmcalibration = "default"
fxcalibration = "default"
pricing = "default"
simulation = "default"
sensitivity = "default"
```

The supported selectors seen in the maintained examples are:

- `pricing`: market view used for base pricing and report generation.
- `simulation`: market view used when simulation objects are needed for XVA, IMM, PFE, scenario and VaR style runs.
- `sensitivity`: market view used for sensitivities when sensitivity shocks require a distinct market mapping.
- `lgmcalibration`, `fxcalibration`, `eqcalibration`, `infcalibration`: optional distinct calibration views for the relevant model blocks.

Each selector points to a configuration identifier exposed by the current `resolved_todays_market.json` publication. The same concept as the old market-configuration switch is therefore retained, but the selected object sets are now machine-published and explicitly auditable.

Analytics

The `[analytics]` block contains one child block per analytic family. Each child block has at least an `active` flag and then the analytic-specific configuration. The current engine estate is substantially broader than the valuation-only scope of early VRE releases. The maintained examples and compliance material cover:

- counterparty credit risk: `SA-CCR`, `IMM`;
- credit risk: `SA-CR`, `IRB`;
- CVA capital: `BA-CVA` reduced and full, `SA-CVA`, including SACVA sensitivities sourced from SA-CCR or IMM;
- market risk: `SA-MR` and `IMA`;
- scope-based reporting and evidence flows for `EU`, `PR`, `US`, `MAS` and `AU`;
- exposure, XVA, initial margin, CRIF, P&L explain and VaR style analytics.

What follows mirrors the historical chapter structure, but using the current native block names and current file references.

12.1.2 Pricing, Cashflows, Curves

The baseline valuation analytics are `npv`, `cashflow`, `curves`, `creditcurves`, `additionalResults` and `portfolioDetails`.

12.1.2.1 NPV

The npv block is the standard entry point:

- active: activates the analytic.
- baseCurrency: report currency for the NPV output.
- outputFileName: report file name.
- extendedResults, additionalResults, additionalResultsReportPrecision: optional expansion of the valuation report payload.

12.1.2.2 Cashflows

The cashflow block keeps the current cashflow-report controls:

- active
- outputFileName

12.1.2.3 Curves and credit curves

The curves and creditcurves blocks provide curve snapshots or diagnostics:

- configuration: named market or curve configuration to dump.
- grid: reporting grid for the output.
- outputFileName: output destination.
- outputTodaysMarketCalibration: include calibration details in the curve dump where supported.

12.1.2.4 Portfolio details and auxiliary reports

- portfolioDetails: writes market-object and risk-factor reports via marketObjectFileName and riskFactorFileName.
- additionalResults: writes auxiliary trade-level result sets via outputFileName.

12.1.3 Simulation and Model Calibration

The current simulation estate spans calibration, simulation, scenario, scenarioStatistics and pfe.

12.1.3.1 Calibration

The calibration block is the narrowest form:

- active
- configFile: calibration contract
- outputFile: calibration output

12.1.3.2 Simulation

The simulation block is the main execution control for exposure and path-based analytics. The keys visible in maintained examples include:

- simulationConfigFile: resolved simulation publication.

- pricingEnginesFile, amcPricingEnginesFile, amcCgPricingEnginesFile, immCgAmcPricingEnginesFile, immCgAmcCgPricingEnginesFile: pricing-policy variants used for classic, AMC and compiled-AAD execution.
- active, baseCurrency, cubeFile, storeScenarios, computeCube, scenariodump, aggregationScenarioDataFileName, additionalScenarioDataFileName: output controls.
- amc, amcCg, immCgAmc, immCgAmcCg: execution-mode switches selecting classic AMC, compiled-AAD AMC or IMM-specific compiled variants.
- amcTradeTypes, immCgAmcTradeTypes: trade-family restriction for AMC passes.
- allowPartialScenarios, strictDimensionMatch, strictCorrelations, scenarioSupersetAllowed, maxGridSize, LockConfigFromXml: runtime strictness and compatibility guards.
- observationModel, curveSensiGrid, vegaSensiGrid: path and Greeks-related controls used in advanced simulation-driven risk runs.
- compiled host CE controls for the IMM / compiled simulation path, including cpuCompiledKernels, cpuSimdTarget, immCgCpuCompiledKernels, immCgCpuSimdTarget, immCgForwardThreads, immCgCubeThreads, immCgRegressionOrder, immCgRegressionVarianceCutoff, immCgCacheConditionalExpectation, immCgCeSimd, immCgCeSimdTarget.
- xvaCgUseExternalComputeDevice, xvaCgExternalComputeDevice, xvaCgKernelMaxArgs, xvaCgOpenClCompatibilityMode, xvaCgExternalDeviceCompatibilityMode, xvaCgUseDoublePrecisionForExternalCalculation, xvaCgUsePythonIntegration, xvaCgUseRedBlocks, xvaCgAadCheckpointPolicy, xvaCgAadFrontierPruning, xvaCgAdHybridFallback, xvaCgTradeLevelBreakdown, xvaCgForwardThreads, xvaCgCubeThreads, xvaCgRegressionOrder, xvaCgRegressionVarianceCutoff, xvaCgCacheConditionalExpectation, xvaCgCeSimd, xvaCgCeSimdTarget, xvaCgDynamicIM, xvaCgDynamicIMStepSize, xvaCgBumpSensis, xvaCgSensitivityConfigFile: advanced compiled-AAD and external-device controls for XVA CE and sensitivities.
- enableScenarioXva, flipViewXVA, storeFlows, storeSensis, storeCreditStateNPVs, storeSurvivalProbabilities: auxiliary path-data controls.

12.1.3.3 Compiled host CE controls The following controls govern the compiled host conditional-expectation path used in IMM and other simulation-heavy compiled workflows:

- cpuCompiledKernels: compatibility alias for the current host-kernel policy. Auto uses JIT kernels when the build and machine support them, Off forces interpreter fallback, and Strict raises if the requested compiled path cannot be attached.
- cpuSimdTarget: compatibility alias for the preferred SIMD target. Auto selects the best supported ISA, typically NEON on arm64 and AVX-family targets on x86_64.

- `immCgCpuCompiledKernels`: canonical host-kernel policy for the compiled IMM conditional-expectation path. It has the same operational meaning as `cpuCompiledKernels`, but should be preferred in current authored bundles.
- `immCgCpuSimdTarget`: canonical SIMD target selector for the IMM compiled host path.
- `immCgForwardThreads`: caps the worker pool used for the compiled forward sweep. Increase it for large path counts when the host has spare cores; reduce it when memory pressure or oversubscription dominates.
- `immCgCubeThreads`: thread budget for host-side cube assembly and write-out. This is the practical control for preventing the cube writer from becoming the serial bottleneck after a fast compiled sweep.
- `immCgRegressionOrder`: polynomial order of the least-squares regression basis used in compiled CE. Higher order increases expressive power and cost; lower order reduces solve time and stabilises large runs.
- `immCgRegressionVarianceCutoff`: optional PCA-style explained variance threshold used to drop weak principal components before the CE solve. This is a regularisation control for numerically noisy or over-parameterised regressions.
- `immCgCacheConditionalExpectation`: enables reuse of CE factorizations between compatible nodes and basis/filter combinations. It should normally remain enabled in production-style compiled runs.
- `immCgCeSimd`: policy for SIMD acceleration inside the CE solve itself. It mirrors the host-kernel policy but acts on the regression and dense-linear-algebra path.
- `immCgCeSimdTarget`: preferred SIMD target for the CE solve. Override it only when benchmarking or when the CE path must be pinned to a narrower ISA than the global host path.

12.1.3.4 Compiled-AAD and external-device controls The following controls govern the compiled-AAD, external-device and XVA conditional-expectation path:

- `xvaCgUseExternalComputeDevice`: enables an external device path for compiled XVA CE and sensitivity evaluation.
- `xvaCgExternalComputeDevice`: explicit device selector, for example an OpenCL, Metal or CUDA target recognised by the compute environment.
- `xvaCgKernelMaxArgs`: upper bound on arguments per generated device kernel before code generation splits the kernel. This is a stability and compatibility control for large generated kernels.
- `xvaCgOpenClCompatibilityMode`: guarded-math compatibility mode for OpenCL execution. `Auto` keeps the safest device-specific behaviour unless a run has a clear reason to force `On` or `Off`.
- `xvaCgExternalDeviceCompatibilityMode`: conservative execution mode for external devices. Use it when a device path is numerically or operationally fragile and determinism matters more than raw speed.

- `xvaCgUseDoublePrecisionForExternalCalculation`: requests double-precision device evaluation where the backend supports it.
- `xvaCgUsePythonIntegration`: enables the Python integration seam used in selected advanced workflows and device experiments.
- `xvaCgUseRedBlocks`: keeps the reduced block structure in the compiled graph to improve runtime and memory behaviour.
- `xvaCgAadCheckpointPolicy`: checkpoint strategy for the AAD graph. `Auto` is the default, while `Recompute` and `Aggressive` trade additional recomputation for lower peak memory.
- `xvaCgAadFrontierPruning`: removes unreachable nodes from the adjoint frontier to reduce memory footprint and reverse-sweep cost.
- `xvaCgAdHybridFallback`: enables guarded fallback when a fully compiled path is not yet available or not yet robust for part of the run.
- `xvaCgTradeLevelBreakdown`: requests trade-level decomposition in the compiled path where the analytic supports it.
- `xvaCgForwardThreads`: thread budget for compiled forward evaluation in the XVA CG path.
- `xvaCgCubeThreads`: thread budget for compiled cube generation and write-out in the XVA CG path.
- `xvaCgRegressionOrder`: regression order for compiled XVA CE. The default is tuned for broad production utility; override only when bias/cost trade-offs are understood for the portfolio.
- `xvaCgRegressionVarianceCutoff`: variance-based regressor filter for the compiled XVA CE path.
- `xvaCgCacheConditionalExpectation`: CE cache toggle for the compiled XVA path. It should generally remain enabled.
- `xvaCgCeSimd`: SIMD execution policy for compiled XVA CE.
- `xvaCgCeSimdTarget`: preferred SIMD target for compiled XVA CE.
- `xvaCgDynamicIM`: enables Dynamic IM support inside the compiled XVA workflow when the run is using the dynamic-initial-margin model.
- `xvaCgDynamicIMStepSize`: revaluation step size used by the Dynamic IM path.
- `xvaCgBumpSensis`: requests bump-style sensitivities from the compiled XVA path where the run needs explicit sensitivity production.
- `xvaCgSensitivityConfigFile`: the sensitivity publication used by the compiled XVA sensitivity path. In current packs this is normally a resolved JSON sensitivity contract under `runtime_policy`.

12.1.3.5 Scenario and scenario statistics

- scenario: writes scenario outputs via simulationConfigFile and scenarioOutputFile.
- scenarioStatistics: writes path distributions and statistics via outputDistributions, outputStatistics, amcPathDataOutput and scenarioOutputFile.

12.1.3.6 PFE

The pfe block is the classic exposure-profile surface:

- baseCurrency, cubeFile, scenarioFile
- csaFile: resolved netting/CSA contract
- quantile, exposureProfiles, exposureProfilesByTrade, hyperCube, allocationMethod, marginalAllocationLimit, rawCubeOutputFile, netCubeOutputFile, exerciseNextBreak, calculationType

12.1.4 Value Adjustments

The xva block remains the central value-adjustment surface. The maintained keys include:

- csaFile, collateralBalancesFile, cubeFile, scenarioFile, baseCurrency
- XVA switches: cva, dva, fva, colva, mva, dim, creditMigration, dynamicCredit
- funding and collateral curves: fvaBorrowingCurve, fvaFundingCurve, fvaInvestmentCurve, fvaLendingCurve, flipViewBorrowingCurvePostfix, flipViewLendingCurvePostfix, collateralSpread, colvaSpread, eoniaFloor, collateralFloor
- output and profile controls: quantile, exposureProfiles, exposureProfilesByTrade, rawCubeOutputFile, netCubeOutputFile, hyperCube, allocationMethod, marginalAllocationLimit, exerciseNextBreak, useXvaRunner
- DIM and credit-migration controls: dimModel, dimScaling, dimQuantile, dimOutputGridPoints, dimOutputNettingSet, dimRegressionOrder, dimLocalRegressionBandwidth, dimLocalRegressionEvaluations, dimDistributionGridSize, dimDistributionCoveredStdDevs, dimEvolutionFile, dimRegressionFiles, dimRegressors, dimHorizonCalendarDays, creditMigrationConfig, creditMigrationTimeSteps, creditMigrationDistributionGrid, creditMigrationOutputFiles

12.1.5 Sensitivity and Stress Testing

The generic market-risk style sensitivity and stress blocks are sensitivity, stress, sensitivityStress, zeroToParSensiConversion, zeroToParShift and parStressConversion.

12.1.5.1 Sensitivity

- sensitivityConfigFile: resolved sensitivity publication

- marketConfigFile, pricingEnginesFile
- sensitivityOutputFile, scenarioOutputFile, parSensitivityOutputFile, parRateSensitivityOutputFile, crossGammaOutputFile, jacobiOutputFile, jacobiInverseOutputFile
- parSensitivity, recalibrateModels, optimiseRiskFactors, outputJacobi, outputSensitivityThreshold

12.1.5.2 Stress and sensitivity stress

- stressConfigFile: resolved stress pack
- marketConfigFile, pricingEnginesFile, sensitivityConfigFile
- scenarioOutputFile, scenarioCashflowOutputFile, stressZeroScenarioDataFile
- generateCashflows, precision, outputThreshold, calcBaseScenario

12.1.5.3 Conversion analytics

- zeroToParSensiConversion: converts zero sensitivities to par sensitivities using the supplied Jacobian output controls.
- zeroToParShift: converts zero stress shifts to par shifts via parShiftsFile and scenarioOutputFile.
- parStressConversion: converts stress packs using the supplied sensitivity and market configuration.

12.1.6 Value at Risk

The current run-bundle surface supports both historical and parametric VaR:

- historicalSimulationVar: keys include historicalPeriod, historicalScenarioFile, includeExpectedShortfall, mporCalendar, mporDays, mporOverlappingPeriods, quantiles, tradePnl, outputFile, simulationConfigFile.
- parametricVar: keys include method, breakdown, covarianceInputFile, portfolioFilter, quantiles, sensitivityInputFile, outputFile.

12.1.7 P&L, P&L Explain, ZeroToParShift, Scenario

The historical P&L toolchain remains present with native run-bundle blocks:

- pnl: uses simulationConfigFile, portfolioMporFile, curveConfigMporFile, conventionsMporFile, mporCalendar, mporDays, outputFileName.
- pnlExplain: adds sensitivityConfigFile, parSensitivity and mporDate to the P&L explain workflow.
- scenario: writes base or shocked market scenarios through scenarioOutputFile.

- `zeroToParShift` and `parStressConversion` complete the current scenario-conversion chain where par-space outputs are needed.

12.1.8 Initial Margin: ISDA SIMM, IM Schedule, CRIF Generation

The initial-margin surface is now distributed across `initialMargin`, `simm`, `crif` and `imschedule`.

- `initialMargin`: high-level selector via `method`.
- `simm`: current keys include `version`, `simmCalibration`, `calculationCurrency`, `mporDays`, `crif`, `enforceIMRegulations`, `writeIntermediateReports`.
- `crif`: keys include `simmVersion`, `baseCurrency`, `marketConfigFile`, `sensitivityConfigFile`, `crifOutputFile`.
- `imschedule`: keys include `calculationCurrency` and `crif` when the schedule calculator is driven from CRIF output.

12.1.9 XVA Stress Testing

The `xvaStress` block uses the current XVA market and stress surface:

- `marketConfigFile`
- `stressConfigFile`
- `writeCubes`

12.1.10 XVA Sensitivity

The `xvaSensitivity` block is the current XVA-sensitivity launcher:

- `marketConfigFile`
- `sensitivityConfigFile`
- `sensiSource`: sensitivity source selector
- `parSensitivity`, `allowPartialScenarios`, `flipViewXVA`, `outputJacobi`, `outputSensitivityThreshold`, `jacobiOutputFile`, `jacobiInverseOutputFile`, `externalFile`

12.1.11 XVA Explain

The `xvaExplain` block provides current MPOR-based XVA explain controls:

- `marketConfigFile`
- `sensitivityConfigFile`
- `mporCalendar`, `mporDays`
- `shiftThreshold`
- `writeCubes`

12.1.12 CCR Capital: SA-CCR and IMM

Current counterparty-capital runs are configured through `saccr` and `imm`.

12.1.12.1 SA-CCR

- activation and scope: `active`, `simplifiedSACCR`, `OEMSACCR`, `ignoreMarginSACCR`, `flipViewXVA`
- inputs: `saccrPortfolioFile`, `saccrCollateralBalancesFile`, `saccrCsaFile`, `saccrCommodityClassificationFile`, `saccrCorrelationFile`, `saccrCrifFile`, `saccrCcr3PortfolioClassificationFile`, `saccrIrbExposureAllocationFile`, `saccrIrbObligorRiskComponentsFile`, `saccrMarginAgreementFile`, `saccrMarginAgreementLinksFile`, `saccrMarginAgreementCollateralFile`, `saccrQccpDefaultFundCapitalFile`
- behaviour: `saccrAllowOutsideNettingCollateral`, `saccrEnableCre54`, `saccrEnableMarginAgreementSplits`, `saccrEnableMultiAssetAllocation`, `saccrProtectionSellerPremiumCapEnabled`, `saccrSoldOptionZeroingEnabled`, `saccrMtmScalingFactor`
- outputs: `saccrOutputFileName`, `saccrTreeOutputFileName`, `saccrCrifOutputFileName`, `saccrWriteExtendedReports`, `saccrWriteMarginAgreementDiagnostics`, `saccrBaseCurrency`

12.1.12.2 IMM

- `baseCurrency`, `csaFile`, `cubeFile`, `scenarioFile`, `marketConfigFile`, `stressConfigFile`, `stressTestId`, `saccrCcr3PortfolioClassificationFile`
- `immApproved`, `immScopeProducts`, `immEnableSwwr`, `immEnableMarginAgreementGrouping`
- `immApplyMporFloors`, `immMporFloorClientDays`, `immMporFloorOtcDays`, `immDisable5000TradeFloor`, `immEffectiveMaturityEnabled`
- `exposureProfiles`, `exposureProfilesByTrade`

12.1.13 CVA Capital: SA-CVA and BA-CVA

The current CVA-capital block is `sacva`. The method selection is carried by its fields:

- `saCvaMethod`: the capital method selector. This is where the maintained surface distinguishes `SA-CVA` from `BA-CVA` reduced or full workflows.
- `saCvaExposureMethod`: exposure source selector, allowing the run to bind to SA-CCR, IMM or auto-selected exposure production.
- `saCvaExposureProfileMethod`, `saCvaImmSource`, `saCvaImmSensiFile`, `immEepeBindingRule`: IMM-linked CVA controls.
- `cvaExecutionMode`: current regulatory execution-mode selector.
- `csaFile`, `collateralBalancesFile`: collateral and CSA inputs where the CVA workflow needs them directly.

- useUnhedgedCvaSensis, sacvaUseTradePar, sacvaFxOrientation, allowFlatPfeOverride: SA-CVA treatment controls.
- xvaCgExternalComputeDevice: external compute selector where CVA sensitivities are run through compiled CE / AAD infrastructure.

12.1.14 Market Risk: SA-MR and IMA

The marketRisk block is the current launch surface for the Basel SA-MR and IMA stack.

- scope and method: method, simplifiedMode, enforceDeskApproach, imaEnforceEligibility
- market and sensitivity inputs: marketConfigFile, pricingEnginesFile, sensitivityConfigFile, scenarioFile, externalFile
- sensitivity behaviour: sensiSource, parSensitivity, recalibrateModels, optimiseRiskFactors, alignPillars, outputJacobi, outputSensitivityThreshold
- SA-MR option and commodity method controls: mar40CommodityMethod, mar40InterestRateMethod, mar40OptionMethod, mar40OptionGreekSource, mar40OptionGreekOrder, mar40OptionScenarioSource, mar40OptionScenarioHedgePolicy, mar40OptionScenarioPoints, mar40OptionScenarioVolPoints, mar40OptionScenarioVolShift, mar40OptionScenarioWriteGrid
- IMA inputs: imaBacktestInputFile, imaPlaInputFile, imaModellabilityInputFile, imaComputeSource, imaDevicePolicy, imaCeCompiledKernels, imaCpuCompiledKernels
- compiled-AAD stability controls: samrCgAadCheckpointPolicy, samrCgAadScenarioLoop, samrCgDisableNodeDeletion, xvaCgExternalComputeDevice

This block is the current user-guide surface for both headless and RaaS-driven Basel III end-game market-risk runs.

12.2 Market View And Today's Market Publication

The historical `todaysmarket.xml` file mixed:

- market-view selection,
- family-to-configuration mapping,
- market-object lookup rules,
- runtime-ready assignments.

The maintained native surface splits those concerns across:

- `market_view.toml`: the authored view selector,
- `market_manifest.yaml`: the view-to-family map,
- `market_object_map.json`: the market-object family lookup map,
- `resolved_todays_market.json`: the published runtime contract.

The published contract is the runtime input. A representative structure is shown in listing 5.

Listing 5: Current resolved today's-market publication

```
{
  "marketView": {
    "viewId": "default",
    "asOfDate": "2026-03-13"
  },
  "marketManifest": {
    "views": [
      {
        "viewId": "default",
        "families": {
          "discounting_curves": "discount_default",
          "fx_spots": "fx_default",
          "fx_volatilities": "fxvol_default",
          "index_forwarding_curves": "index_default",
          "yield_curves": "yield_default"
        }
      }
    ]
  },
  "marketObjectMap": {
    "families": [
      { "familyId": "discount_default", "marketObject": "DiscountCurve" },
      { "familyId": "fx_default", "marketObject": "FXSpot" }
    ]
  },
  "todaysMarketParameters": {
    "configurations": [
      {
        "configurationId": "default",
        "marketObjectIds": {
          "YieldCurve": "yield_default",
          "DiscountCurve": "discount_default",
          "FXSpot": "fx_default",
          "FXVol": "fxvol_default"
        }
      }
    ]
  }
}
```

Publication structure

The publication is deliberately layered:

- `marketView` identifies the requested view and as-of date.
- `marketManifest.views` maps each view identifier to market families such as `discounting_curves` or `fx_spots`.
- `marketObjectMap.families` binds family identifiers to runtime market-object classes such as `DiscountCurve` or `FXVol`.
- `marketObjectMap.entries` provides the detailed lookup records: `familyId`, `marketObject`, `objectKey`, `lookupKey`, `objectId`.
- `todaysMarketParameters.configurations` exposes the final runtime configuration identifiers used by the engine.
- `todaysMarketParameters.marketObjects` records the resolved assignments per market-object class and family.

The effect is the same as in the legacy XML file: the engine can still ask for the *pricing* or *simulation* market configuration and obtain the right set of curves, surfaces and spot values. The difference is that each stage of the mapping is now explicit and separately testable.

12.2.1 Discounting Curves

Discounting curves are published as the market-object class `DiscountCurve`. In the current examples:

- the manifest exposes a family such as `discounting_curves`;
- that family resolves to a family identifier such as `discount_default`;
- the object map binds `discount_default` to `DiscountCurve`;
- the detailed entries then map lookup keys such as `EUR` or `USD` to resolved object identifiers such as `Yield/EUR/EUR1D`.

The same pattern is used across larger examples with family identifiers such as `discount_curve__default`, `discount_curve__ois`, `discount_curve__inccy`, `discount_curve__xois_eur` and `discount_curve__xois_usd`.

12.2.2 Index Curves

Index forwarding curves are published under the market-object class `IndexCurve`. The lookup key remains the index identifier such as `EUR-EURIBOR-6M` or `USD-LIBOR-3M`. The object map resolves those keys to object identifiers such as `Yield/EUR/EUR6M`.

The maintained family naming makes the intended use explicit:

- `index_forwarding_curve__default`
- `index_forwarding_curve__inccy`
- `index_forwarding_curve__single`
- `index_forwarding_curve__xois_eur`

12.2.3 Yield Curves

Yield curves are published separately from discount curves under the market object `YieldCurve`. This keeps the old distinction between discounting and reporting / general yield access explicit in the native model. The resolved configuration therefore maps both:

- `DiscountCurve`
- `YieldCurve`

to potentially different family identifiers, even when they happen to resolve to the same underlying object identifiers in a minimal example.

12.2.4 Swap Index Curves

Swap-index curves are published under `SwapIndexCurve`. These are the native successors of the old swap-index configuration nodes and are selected in exactly the same way: by configuration identifier, family identifier and object-key lookup.

12.2.5 FX Spot

FX spot data is published under `FXSpot`. A record in the object map contains:

- `objectKey`: for example `fx_spot:EURUSD`
- `lookupKey`: for example `EURUSD`
- `objectId`: for example `FX/EUR/USD`

This is more explicit than the old XML representation because the human lookup key and the runtime object identifier are both preserved.

12.2.6 FX Volatilities

FX volatility structures are published under `FXVol`. The same view may expose more than one family, for example a default family and a stress family. In the common example:

- `fxvol_default` points to `FXVol/EUR/USD`
- `fxvol_stress` points to `FXVol/EUR/USD/STRESS`

This makes the stressed-surface selection explicit in the market publication rather than hiding it in analytic-side overrides.

12.2.7 Swaption Volatilities

Swaption volatility structures are published under `SwaptionVol`. Larger current examples use family identifiers such as `swaption_vol__default` and `swaption_vol__xois_eur`. The same configuration mechanism used for FX vols applies: a market configuration points to a family, and the family points to the resolved object identifiers.

12.2.8 Cap/Floor Volatilities

Interest-rate cap/floor volatility structures are published under `CapFloorVol`. The native publication preserves the same economic role as the legacy XML family while making the selected family and object lookup explicit in the runtime JSON.

12.2.9 Default Curves

Default curves are published under `DefaultCurve`. Current examples use family identifiers such as `default_curve__default` and `default_curve__xois_eur`. This class is consumed by credit-risk, CCR and CVA workflows alongside the counterparty and legal-entity publications.

12.2.10 Securities

Security-level market data is published under `Securities`. The native object map allows the security market family to be switched independently from the curve and volatility families, which is useful in market-risk and XVA runs where security spreads are sourced differently from rates or FX.

12.2.11 Equity Curves

Equity curves are published under `EquityCurves`. The family mapping works exactly as for yield and discount curves: a configuration chooses the family, and the family chooses the concrete object identifiers.

12.2.12 Equity Volatilities

Equity volatility structures are published under `EquityVols`. The native publication allows equity-spot and equity-vol families to move independently, which is important for stress and scenario runs.

12.2.13 Inflation Index Curves

The current market publication distinguishes:

- `ZeroInflationCurve`
- `YoYInflationCurve`

This is clearer than the legacy XML surface because the runtime market-object class already encodes whether the curve is zero or year-on-year inflation.

12.2.14 Inflation Cap/Floor Volatility Surfaces

Inflation volatility surfaces are published separately as:

- `ZeroInflationCapFloorVol`
- `YoYInflationCapFloorVol`

Again, the market-object class makes the intended economic use explicit.

12.2.15 CDS Volatility Structures

CDS option volatility surfaces are published under `CDSVol`. This family becomes relevant in the richer credit and CVA examples where the market publication spans credit-option inputs as well as default curves.

12.2.16 Base Correlation Structures

Base correlation structures are published under `BaseCorrelation`. These are selected in the same configuration mechanism as other market objects, which means they can be switched between pricing and stressed views without special analytic-side wiring.

12.2.17 Correlation Structures

Standalone market correlation structures are published under `Correlation`. These are market-data correlations, distinct from the simulation model correlation set documented in section 12.3.2.

12.2.18 Commodity Curves and Volatilities

The current native publication also supports:

- `CommodityCurves`
- `CommodityVolatilities`

This is important for the current SA-CVA and market-risk scope, where commodity factors form part of the maintained Basel III end-game analytic estate.

12.2.19 Market Configurations

The final runtime objects consumed by the engine are the `todaysMarketParameters.configurations` entries. Each configuration contains:

- `configurationId`: for example `default` or `stress`
- `marketObjectIds`: a map from runtime market-object classes to family identifiers

This is the object that the run bundle selects via `markets.pricing`, `markets.simulation`, `markets.sensitivity` and the calibration selectors. The native market-publication design therefore preserves the old user-facing market-view concept, but with a much clearer machine-readable contract.

12.3 Simulation Runtime And Credit Simulation

The XML-era `simulation.xml` file combined four distinct concerns:

- path-generation and grid controls,
- cross-asset model specification,
- cross-factor correlations,
- simulation-market definition.

The maintained native surface separates these concerns into published machine contracts and keeps the run-bundle analytic block as the execution selector. The canonical runtime publication is `examples/common/input/runtime_policy/published/resolved_simulation.json`.

Listing 6: Current resolved simulation publication

```
{
  "simulationRuntime": {
    "grid": "241,1M",
    "calendar": "EUR,USD,GBP,CHF",
    "dayCounter": "A365",
    "sequence": "SobolBrownianBridge",
    "scenarioGenerator": "Simple",
    "seed": 42,
    "samples": 1000,
    "ordering": "Steps",
    "directionIntegers": "JoeKuoD7",
    "closeOutLag": "2W",
    "mporMode": "StickyDate",
    "timeStepsPerYear": 52,
    "xva": {
      "regressionOrder": 4,
      "regressionVarianceCutoff": 14,
      "cacheConditionalExpectation": "Auto",
      "ceSimd": "Auto"
    }
  },
  "crossAssetModelProfile": {
    "domesticCurrency": "EUR",
    "factors": {
      "currencies": ["EUR", "USD", "GBP"],
      "equities": [],
      "inflationIndices": [],
      "creditNames": [],
      "commodities": []
    },
    "measure": "LGM"
  }
}
```

12.3.1 Parameters

The top-level publication envelope carries `schemaVersion`, `seedVersion`, `publishedAt`, `contentHash` and `sourceLineage`. The operational simulation parameters then live under `simulationRuntime`.

The current `simulationRuntime` keys are:

- `grid`: the simulation grid specification. In the current examples this is expressed as a count plus step, for example `241,1M`.

- `calendar`: the calendar set used when advancing the path grid.
- `dayCounter`: day-count basis for time conversion.
- `sequence`: low-discrepancy or pseudo-random sequence choice.
- `scenarioGenerator`: generator family, for example *Simple*.
- `seed`: RNG seed for deterministic replay.
- `samples`: path count.
- `ordering`: dimension ordering policy used by the sequence.
- `directionIntegers`: Sobol direction-integer set.
- `closeOutLag`: close-out lag used in exposure and capital workflows.
- `mporMode`: MPOR interpretation mode used when the exposure engine advances collateral or close-out dates.
- `timeStepsPerYear`: annual discretisation fallback for the selected grid mode.

The current runtime publication also groups execution-specific controls:

12.3.1.1 XVA runtime controls

- `xva.regressionOrder`: polynomial order of the CE regression basis.
- `xva.regressionVarianceCutoff`: truncation or screening control used by the regression engine.
- `xva.forwardThreads`, `xva.cubeThreads`: thread allocation for forward and cube work.
- `xva.cacheConditionalExpectation`: CE cache policy.
- `xva.ceSimd`, `xva.ceSimdTarget`: SIMD policy for the CE implementation.

12.3.1.2 CPU runtime controls

- `cpu.compiledKernels`: host compiled-kernel policy.
- `cpu.simdTarget`: CPU SIMD target.

These runtime keys are intentionally separated from the model profile. They are execution-policy selections rather than economic-model definitions.

12.3.2 Model

The model definition is carried by `crossAssetModelProfile` and `crossAssetModelCorrelations`.

12.3.2.1 Cross-asset model profile The maintained profile contains:

- `domesticCurrency`: the domestic numeraire currency.
- `factors`: the factor universe grouped by `currencies`, `equities`, `inflationIndices`, `creditNames`, `commodities`.

- `bootstrapTolerance`: tolerance used during model bootstrap.
- `measure`: the model measure, for example *LGM*.
- `integration.discretization`, `integration.salvagingAlgorithm`, `integration.piecewise`: discretisation and correlation-salvage policy.
- `modelBlocks`: the model-family payloads.

The `modelBlocks` object is grouped by factor family:

- `interestRate`
- `foreignExchange`
- `equity`
- `inflation`
- `credit`
- `commodity`

Each model-block record contains:

- `modelType`: the model implementation, for example *LGM* or *CrossCcyLGM*.
- `key`: the curve, currency or factor key for the block.
- `parameters`: the model-specific parameter payload.

In the maintained examples the interest-rate blocks expose a detailed parameter shape including:

- `CalibrationSwaptions.Expiries`, `CalibrationSwaptions.Terms`, `CalibrationSwaptions.Strikes`
- `CalibrationType`
- `ParameterTransformation.Scaling`, `ParameterTransformation.ShiftHorizon`
- `Reversion.Calibrate`, `Reversion.InitialValue`, `Reversion.ParamType`, `Reversion.ReversionType`, `Reversion.TimeGrid`
- `Volatility.Calibrate`, `Volatility.InitialValue`, `Volatility.ParamType`, `Volatility.TimeGrid`, `Volatility.VolatilityType`

The FX blocks similarly expose a native parameter payload:

- `CalibrationOptions.Expiries`, `CalibrationOptions.Strikes`
- `CalibrationType`
- `DomesticCcy`
- `Sigma.Calibrate`, `Sigma.InitialValue`, `Sigma.ParamType`, `Sigma.TimeGrid`

This is the core improvement over the monolithic XML file: the native surface retains the model detail, but makes its family structure explicit and machine-readable.

12.3.2.2 Cross-asset correlations

The correlation payload is published separately as `crossAssetModelCorrelations`. Each entry contains:

- `factor1.type`, `factor1.name`
- `factor2.type`, `factor2.name`
- `value`

This allows the correlation set to be validated independently from the model blocks and to be reused across multiple simulation runtime profiles.

12.3.3 Market

The simulation market carried by the old XML file is now represented by `scenarioMarketProfile` in the resolved publication and by the market configuration selectors in the run bundle.

The current resolved structure is intentionally light:

- `scenarioMarketProfile.marketDefinitionText`: free-form description or provenance text for the simulation market definition.

The heavy market-object mapping is not duplicated here. Instead:

- the run bundle chooses the `markets.simulation` configuration,
- the selected `resolved_todays_market.json` publication provides the market-object family assignments,
- the simulation profile consumes those assignments deterministically.

This split is important because the same model profile can be reused with different market views, and a single market publication can serve pricing, simulation and sensitivity contexts.

12.3.4 CAM extension controls and strict modes

The current guide also needs to document the simulation-related execution controls that live in the run-bundle analytic block rather than in the resolved simulation publication.

The maintained `analytics.simulation` surface includes:

- `allowPartialScenarios`: whether incomplete scenario coverage is tolerated.
- `strictDimensionMatch`: whether factor dimensions must match the configured model and shift surface exactly.
- `strictCorrelations`: whether missing or inconsistent correlation entries are treated as hard errors.
- `scenarioSupersetAllowed`: whether the scenario market may contain factors beyond the active model universe.
- `maxGridSize`: guardrail against path grids that exceed the intended runtime envelope.
- `LockConfigFromXml`: compatibility switch used while migration seams still exist.
- `amc`, `amcCg`, `immCgAmc`, `immCgAmcCg`: execution-mode selectors for classic and compiled CE/AAD workflows.

- `xvaCgUseExternalComputeDevice`, `xvaCgExternalComputeDevice`, `xvaCgKernelMaxArgs`, `xvaCgOpenClCompatibilityMode`: external-device execution controls used in the current GPU and compiled-AAD examples.

These controls are still part of the user-facing simulation surface even though they are not part of the economic model. They determine whether the selected model can be executed robustly on CPU, compiled host CE, or external compute devices.

12.3.4.1 Credit simulation Credit simulation is now a sibling native family rather than an appendix to the cross-asset simulation XML. The current resolved credit-simulation contract `resolved_credit_simulation.json` combines:

- the authored `credit_simulation_runtime.toml`,
- transition-profile and transition-matrix publications,
- scoped legal-entity and netting-set selections,
- credit-mode controls used by portfolio-credit and capital analytics.

The current native split is therefore:

- cross-asset Monte Carlo and CE in `resolved\_simulation.json`,
- credit migration and transition modelling in `resolved\_credit\_simulation.json`.

12.4 Sensitivity Runtime And Shift Tables

VRE continues to support sensitivities with respect to the major market object families used by pricing, XVA, CVA capital and market risk, including:

- discount curves,
- index curves,
- yield curves,
- FX spots,
- FX volatilities,
- swaption volatilities,
- cap / floor volatilities,
- default and credit curves,
- equity spots and equity volatilities,
- inflation curves,
- CDS volatilities,
- bond credit spreads,
- base correlations,
- standalone market correlations.

The maintained native surface is no longer a monolithic `sensitivity.xml`. It is split into:

- `sensitivity_runtime.toml`: scalar execution switches,
- `sensitivity_profile.yaml`: profile-level behaviour,
- `sensitivity_shift_tables.json`: keyed shift payloads,
- `resolved_sensitivity.json`: the published runtime contract.

Listing 7: Current resolved sensitivity publication

```
{
  "sensitivityRuntime": {
    "profileId": "baseline_sensitivity_profile",
    "parConversionEnabled": true,
    "parSensiRemoveFixing": "FX-.*",
    "computeGamma": true,
    "useSpreadedTermStructures": true,
    "creditShockPrimitive": "HazardRate",
    "clampRecoveryRates": true
  },
  "sensitivityProfile": {
    "parConversionExcludes": ["FXSpot"],
    "twoSidedDeltaKeyTypes": ["DiscountCurve", "FXSpot"],
    "crossGammaFilter": [
      { "lhs": "DiscountCurve/EUR", "rhs": "FXSpot/USDEUR" }
    ]
  }
}
```

The general shift vocabulary is still the same as in the historical guide:

- `shiftType`: for example *Absolute* or *Relative*
- `shiftSize`: magnitude of the shift
- `shiftScheme`: finite-difference scheme such as *Forward*, *Backward* or *Central*
- `shiftTenors`: tenor buckets for curve shifts
- `shiftExpiries`: option-expiry buckets for volatility surfaces
- `shiftStrikes`: strike buckets for option-volatility surfaces
- `shiftTerms`: underlying swap or loss-level terms for swaption- and base-correlation-style shocks

The key difference is that these attributes are now normalised in JSON rather than scattered across XML nodes.

Authored runtime selector

The authored runtime selector carries only the scalar switches that genuinely vary by execution profile:

- `profileId`: links the runtime selector to the profile and shift-table publication.
- `parConversionEnabled`: enables par conversion where the shift family supports it.
- `parSensiRemoveFixing`: regex-style fixing-removal policy used in par sensitivity conversion.
- `computeGamma`: whether second-order sensitivities are requested.
- `useSpreadedTermStructures`: use spreaded term structures where supported.
- `creditShockPrimitive`: credit sensitivity primitive, for example *HazardRate*.
- `clampRecoveryRates`: whether recovery-rate shocks are clamped to the legal range.

Resolved publication

The published `resolved_sensitivity.json` contract contains:

- `sensitivityRuntime`
- `sensitivityProfile`
- `sensitivityShiftTables`
- `sensitivityAnalysis`

The last object, `sensitivityAnalysis`, is the fully resolved runtime form that the engine consumes. It combines the scalar runtime switches, the profile behaviour and the shift-family tables into a single deterministic payload.

Profile behaviour

The current `sensitivityProfile` object carries:

- `profileId`
- `parConversionExcludes`: key types that are excluded from par conversion.
- `twoSidedDeltaKeyTypes`: key types that should use two-sided delta treatment.
- `crossGammaFilter`: list of `lhs`, `rhs` key-prefix pairs used to restrict cross-gamma computation.

This is the native successor of the old XML cross-gamma and product-template behaviour. Instead of embedding the logic deep in an XML tree, the resolved profile makes the intended execution rules explicit.

12.4.1 FactorSpec contract and sensitivity axis canonicalization

CAM-extension sensitivity runs canonicalise all configured axes into a stable factor specification before pricing. The canonical axis vocabulary remains:

- tenor axes,
- expiry axes,
- strike axes,
- term or loss-level axes where the shocked family requires them.

In the native publication this normalization is visible directly in the `sensitivityAnalysis` payload:

- `shiftTenors` is already a normalized array of tenor strings,
- `shiftExpiries` is already a normalized array of expiries,
- `shiftStrikes` is already a normalized numeric strike array,
- `shiftTerms` is already a normalized term array,
- `shiftLossLevels` is already a normalized numeric loss-level array for base correlation shocks.

The operational advantage is that the resolved sensitivity contract can be compared directly against the model and market contracts when `strictDimensionMatch` is enabled in the simulation analytic block.

Shift families

The current `sensitivityShiftTables.families` object groups shifts by market family. The common example publication already demonstrates:

- `discountCurves`
- `creditCurves`
- `fxSpots`

- `swaptionVolatilities`
- `baseCorrelations`

The same family model extends to the other supported market-object classes in the larger current examples.

12.4.1.1 Discount curves

Discount-curve shift rows use:

- `ccy`
- `shiftType`, `shiftSize`, `shiftScheme`
- `shiftTenors`
- optional `parConversion` metadata with `instruments`, `singleCurve` and `conventions`

12.4.1.2 Credit curves

Credit-curve shift rows use:

- `name`
- `currency`
- `shiftType`, `shiftSize`, `shiftScheme`
- `shiftTenors`
- optional `parConversion` metadata

12.4.1.3 FX spots

FX-spot rows use:

- `ccypair`
- `shiftType`, `shiftSize`, `shiftScheme`

12.4.1.4 Swaption volatilities

Swaption-volatility rows use:

- `key` for the currency or surface key
- `shiftType`, `shiftSize`, `shiftScheme`
- `shiftExpiries`, `shiftTerms`, `shiftStrikes`

12.4.1.5 Base correlations

Base-correlation rows use:

- `indexName`
- `shiftType`, `shiftSize`, `shiftScheme`
- `shiftTerms`
- `shiftLossLevels`

Cross-gamma filter

The native cross-gamma filter preserves the historical semantics: each pair in `crossGammaFilter` matches sensitivity-key prefixes. For example:

- `DiscountCurve/EUR, DiscountCurve/EUR` restricts cross gammas to pairs of EUR discount-curve axes,
- `DiscountCurve/EUR, FXSpot/USDEUR` restricts cross gammas to the selected cross-family pair.

This is still the correct operational model for limiting cross-gamma explosion in large market-risk and SA-CVA runs.

Par Sensitivity Analysis

Par sensitivity remains a first-class concern in the maintained native surface. It is governed through two layers:

- run-bundle analytic switches such as `parSensitivity`, `outputJacobi`, `jacobiOutputFile`, `jacobiInverseOutputFile`
- the sensitivity runtime/profile publication via `parConversionEnabled`, `parSensiRemoveFixing`, `parConversionExcludes` and family-level `parConversion` metadata

Family-level `parConversion` records document:

- the instrument templates to use,
- whether the shocked family is treated as a single-curve conversion,
- the conventions required to construct the par instruments.

This explicit native contract is a substantial improvement over the old XML surface because the par-conversion rules are now machine-readable and can be validated against the published convention registry.

Operational consequences

The maintained native split has four direct benefits:

- the runtime selector remains short and reviewable,
- shift families become stable machine data rather than hand-maintained XML fragments,
- Config Doctor and CI can validate family coverage against the active market and curve publications,
- SA-CVA, SA-MR, IMA, XVA and compiled-AAD workflows can share the same resolved contract surface.

12.5 Stress Packs And Scenario Inputs

Stress tests can be applied in VRE to the same market segments and with the same granularity as described in the sensitivity section 12.4. The maintained current surface is no longer a single `stressconfig.xml` file. Instead, stress configuration is split into:

- authored stress-pack metadata in `stress_pack.yaml`,
- keyed shift payloads in `stress_shift_tables.json`,
- published machine contracts `stress_pack.json`, `stress_shift_tables.json` and `resolved_stress_pack.json`,
- a generated XML compatibility file `stresstest_compat.xml` for legacy readers that still expect the old `StressTesting` root.

The canonical current files are:

- `examples/common/input/runtime_policy/stress_pack.yaml`
- `examples/common/input/runtime_policy/stress_shift_tables.json`
- `examples/common/input/runtime_policy/published/resolved_stress_pack.json`

Why the split exists

The old XML file mixed two different classes of information:

- scenario metadata: test identifiers, labels, par-shift flags and general runtime switches,
- the shift payloads themselves: tenor vectors, volatility grids and scenario magnitudes.

These evolve at different rates. Human review normally focuses on the scenario catalogue and the meaning of each scenario, while the heavy shift vectors are often generated or maintained separately. The current split therefore keeps:

- the human-authored scenario catalogue in YAML,
- the high-churn shift payloads in JSON,
- the runtime-facing joined contract in `resolved_stress_pack.json`.

Resolved stress publication

The resolved publication joins stress-pack metadata with keyed shift payloads into the runtime-facing object consumed by the engine. A representative extract is shown in listing 8.

Listing 8: Current resolved stress publication

```
{
  "schemaVersion": "1.0.0",
  "publishedAt": "2026-03-13T00:00:00Z",
  "stressPack": {
    "packId": "baseline_stress_pack",
    "useSpreadedTermStructures": true,
    "tests": [
      {
        "stressTestId": "eur_6m_up",
        "label": "EUR 6M up",
        "parShiftFlags": {
          "irCurves": false,
          "capFloorVolatilities": false,
          "survivalProbability": false
        },
        "shiftTableRefs": {
          "discountCurves": ["discount_all_flat"],
          "indexCurves": ["index_eur_6m_up"],
          "yieldCurves": ["yield_all_flat"]
        }
      }
    ]
  },
  "stressShiftTables": {
    "families": {
      "indexCurves": [
        {
          "shiftId": "index_eur_6m_up",
          "key": "EUR-EURIBOR-6M",
          "shiftType": "Absolute",
          "shiftTenors": ["6M", "2Y", "3Y"],
          "shifts": [0.02, 0.02, 0.02]
        }
      ]
    }
  },
  "stressTesting": {
    "useSpreadedTermStructures": true,
    "tests": [
      {
        "stressTestId": "eur_6m_up",
        "label": "EUR 6M up",
        "parShiftFlags": {
          "irCurves": false,
          "capFloorVolatilities": false,
          "survivalProbability": false
        },
        "indexCurves": [
          {
            "shiftId": "index_eur_6m_up",
            "key": "EUR-EURIBOR-6M",
            "shiftType": "Absolute",
            "shiftTenors": ["6M", "2Y", "3Y"],
            "shifts": [0.02, 0.02, 0.02]
          }
        ]
      }
    ]
  }
}
```

At the top level the resolved contract carries:

- `schemaVersion`: publication schema version,
- `publishedAt`: timestamp of the published artifact,
- `stressPack`: the resolved metadata catalogue,

- `stressShiftTables`: the keyed shift payloads by family,
- `stressTesting`: the fully joined runtime object read by the engine,
- `sourceLineage`: provenance back to the authored YAML / JSON,
- `contentHash`: deterministic content fingerprint.

Stress-pack metadata

The `stressPack` branch is the scenario catalogue. Its core fields are:

- `packId`: unique identifier for the pack,
- `useSpreadedTermStructures`: global runtime switch,
- `tests`: list of individual stress scenarios.

Each `tests[]` entry contains:

- `stressTestId`: unique scenario identifier,
- `label`: human-readable label,
- `parShiftFlags`: booleans indicating whether the scenario is to be interpreted in the par domain for selected families,
- `shiftTableRefs`: references to one or more keyed payloads in the companion shift-table file.

This is the current native equivalent of the old `<StressTest id="...">` node.

UseSpreadedTermStructures

`useSpreadedTermStructures` is the current native equivalent of the legacy `UseSpreadedTermStructures` flag. If set to true, spreaded term structures over the base `t0` market are used for the scenario calculation. This improves the alignment between:

- the static today market,
- the stressed market used for revaluation,
- the simulation market in workflows where shocked term structures are reused in risk or explain runs.

This flag is still operationally important in market-risk, XVA stress and stress-style explain workflows, especially when curve rebuilding and volatility selection would otherwise drift too far from the base-market setup.

Par-shift flags

Each stress test carries a `parShiftFlags` object with the fields:

- `irCurves`
- `capFloorVolatilities`
- `survivalProbability`

If a flag is `true`, the corresponding family is interpreted as a par-shift scenario rather than a zero / direct shift scenario. The same important restriction from the legacy configuration remains:

- par-shift interpretation is set at the family level inside the stress scenario,
- it is not possible within one scenario to mix par and non-par semantics for different objects in the same family.

Par stress scenarios must remain aligned with the par-instrument definitions in the sensitivity configuration. In practice:

- the shifted tenors for IR curves must match the tenors of the par instruments used for sensitivity conversion,
- cap/floor stress expiries and strikes must align with the cap/floor par instrument definition,
- survival-probability par shifts remain coupled to the credit-curve par setup used by the converter.

This is still the same fundamental rule as the XML-era <ParShifts> block. The current guide therefore expects the reader to read this chapter together with section 12.4.

Shift-table families

The companion `stress_shift_tables.json` file stores the actual shift payloads keyed by `shiftId`. The publication separates families by their market-object shape rather than only by business meaning.

12.5.0.1 Curve-like families.

The following families use a tenor-vector payload:

- `discountCurves`
- `indexCurves`
- `yieldCurves`
- `commodityCurves`
- `survivalProbabilities`

Each entry contains:

- `shiftId`: unique payload identifier,
- `key`: the target curve key or wildcard *,
- `shiftType`: usually `Absolute` or `Relative`,
- `shiftTenors`: the shifted tenor grid,
- `shifts`: the vector of shift magnitudes.

The number of tenors and shifts must match exactly.

12.5.0.2 Spot-like families.

The following families use a scalar shift:

- `fxSpots`
- `equitySpots`
- `securitySpreads`
- `recoveryRates`

Each entry contains:

- `shiftId`
- `key`
- `shiftType`
- `shiftSize`

This is the native equivalent of the old `ShiftSize` blocks in the XML file.

12.5.0.3 FX-volatility family.

`fxVolatilities` is special because the engine still supports the two legacy modes:

- explicit expiry-by-expiry shifts,
- weighted or unadjusted single-pillar shifts.

In explicit mode the payload contains:

- `shiftId`
- `key`: e.g. `USDEUR`
- `shiftType`
- `shiftExpiries`
- `shifts`

In weighted or unadjusted mode the payload contains:

- `shiftId`
- `key`
- `shiftType`
- `mode`: `Weighted` or `Unadjusted`
- exactly one shifted pillar in `shiftExpiries` and `shifts`
- for `Weighted` additionally: `weightTenors` and `weights`

The same semantics as the legacy XML chapter still apply:

- `Unadjusted`: apply the shift only to the specified tenor,
- `Weighted`: apply the shift to the specified tenor and scale the other tenors using the supplied weights.

The validation rules in the current contract are stricter and explicit:

- weighted FX-vol shifts must carry exactly one shifted pillar,
- `weightTenors` and `weights` must be aligned,
- explicit mode must align `shiftExpiries` and `shifts`.

12.5.0.4 Other volatility families. The engine also accepts the following volatility stress families when present:

- `equityVolatilities`
- `commodityVolatilities`
- `capFloorVolatilities`
- `swaptionVolatilities`

The payload conventions are:

- `equityVolatilities`: `shiftExpiries` plus `shifts`,
- `commodityVolatilities`: `shiftExpiries`, `shifts` and `shiftMoneyness`,
- `capFloorVolatilities`: `shiftExpiries`, optional `shiftStrikes`, and per-expiry `Shift` rows,
- `swaptionVolatilities`: `shiftExpiries`, `shiftTerms` and either a default parallel shift or expiry/term-specific overrides.

Two practical rules from the legacy chapter remain important:

- for swaption-volatility shifts, a default `Shift` without expiry / term qualifiers acts as the fallback value for expiry/term points that are not overridden explicitly,
- for commodity-volatility shifts, the common operational pattern is still a single shift per expiry, implying the same shift size for all moneyness levels unless a more detailed structure is published.

Resolved runtime object

The engine reads the joined `stressTesting` object. This is the part of the resolved publication that mirrors the old `StressTesting` XML tree most closely:

- `useSpreadedTermStructures`
- `tests[]`
- within each test, the resolved family arrays such as `discountCurves`, `fxSpots`, `fxVolatilities`, `swaptionVolatilities`, etc.

The publication step resolves every `shiftTableRefs` entry into the actual shift objects so that the runtime contract is self-contained.

Replay and historical scenario inputs

Not all scenario-driven workflows are authored as stress packs. VRE also uses row-oriented scenario inputs for replay, historical analysis and evidence workflows, for example CSV or

TXT scenario streams under `configurations/inputs/csv/` and `configurations/inputs/txt/`.

This distinction matters:

- stress packs describe curated deterministic shock definitions used by the stress-style analytics,
- row-oriented replay inputs describe scenario streams or evidence tables produced elsewhere.

Compatibility note

The old XML structure is still available as a generated compatibility surface for legacy readers and for comparison against older examples. However, the maintained source of truth is now:

- YAML for stress-pack metadata,
- JSON for keyed shift payloads,
- resolved JSON for the engine-facing publication.

This retains the full legacy semantics while making the current stress surface easier to validate, publish, diff and consume in Config Doctor, scope-based reporting workflows and cross-repo evidence flows.

12.6 Calendar Adjustments Publication

Calendar adjustments are used to overlay additional holidays and business days onto the built-in calendar set used by VRE. The maintained current surface is no longer a hand-authored `calendaradjustment.xml` file. Instead, the current workflow uses:

- an authored seed `examples/common/input/runtime_policy/calendar_adjustments.yaml`,
- a normalised publication `published/calendar_adjustments_manifest.json`,
- a resolved runtime publication `published/resolved_calendar_adjustments_manifest.json`,
- a generated compatibility file `published/calendaradjustment_compat.xml`.

The canonical runtime publication is `examples/common/input/runtime_policy/published/resolved_calendar_adjustments_manifest.json`.

Listing 9: Current resolved calendar-adjustment publication


```
{
  "schemaVersion": "1.0.0",
  "seedVersion": "1",
  "publishedAt": "2026-03-13T00:00:00Z",
  "contentHash": "b37a82686a04b8d764b101f2119c675eaedd9a63914d2ca2bf17f8f14f0837fa",
  "sourceLineage": {
    "seedPath": "examples/common/input/runtime_policy/calendar_adjustments.yaml",
    "generator": "tools/ci/publish_calendar_adjustment_contracts.py"
  },
  "calendars": [
    {
      "name": "AUD",
      "additionalHolidays": ["2026-01-27"],
      "additionalBusinessDays": ["2026-04-25"],
      "versionedId": "AUD::1"
    },
    {
      "name": "BRISBANE_BANK",
      "baseCalendar": "AUD",
      "additionalHolidays": ["2026-08-14"],
      "additionalBusinessDays": ["2026-01-03"],
      "versionedId": "BRISBANE_BANK::1"
    }
  ]
}
```


Current contract role

Each current calendar record expresses one of two things:

- an overlay on an existing calendar, by adding holidays or business days,
- or a derived calendar, by naming a `baseCalendar` and then applying additional overrides.

The publication makes calendar overlays deterministic and auditable while still supporting a generated XML view for remaining legacy readers.

12.7 Authored seed

The authored YAML seed has the same practical intent as the old XML file, but with explicit native publication and traceability:

Listing 10: Current authored calendar-adjustment seed

```yaml
version: "1"
traceabilityRefs:
  adr: "docs/architecture/adr/0001-serialization-and-storage-format-standards.md"
calendars:
  - name: "AUD"
    additionalHolidays:
      - "2026-01-27"
    additionalBusinessDays:
      - "2026-04-25"
  - name: "BRISBANE_BANK"
    baseCalendar: "AUD"
    additionalHolidays:
      - "2026-08-14"
    additionalBusinessDays:
      - "2026-01-03"
```

The publisher validates that:

- calendar names are unique,
- date strings use ISO `YYYY-MM-DD` format,
- a date does not appear in both `additionalHolidays` and `additionalBusinessDays` for the same calendar.

12.8 Calendar records

Each entry in `calendars[]` carries:

- `name`: calendar identifier.
- `baseCalendar` [Optional]: name of an existing calendar used as the base for a new derived calendar.
- `additionalHolidays` [Optional]: list of extra holiday dates.
- `additionalBusinessDays` [Optional]: list of extra business-day dates.
- `versionedId`: publisher-generated identifier of the form `name::seedVersion`.

name

`name` is the calendar being amended or created.

Two cases are important:

- if no `baseCalendar` is provided, the name should refer to an existing calendar already known to the calendar parser,
- if `baseCalendar` is provided, the name identifies a newly derived calendar that will be registered using the given base.

additionalHolidays

Each date in `additionalHolidays` is added as a holiday to the named calendar. In the runtime implementation this is applied through the calendar overlay and stored in a set of additional holiday dates.

Operationally:

- duplicate holiday dates inside one record are rejected by the publisher and loader,
- adding a date that is already a holiday in the underlying calendar is safe; it is effectively idempotent and does not raise an error.

This keeps the useful practical note from the old guide: if an operator is unsure whether a date is already present as a holiday, adding it again as an additional holiday is a safe way to make the intent explicit.

additionalBusinessDays

Each date in `additionalBusinessDays` is forced to be a business day for the named calendar. In the runtime implementation this is applied by removing the holiday classification for that date.

Operationally:

- duplicate business-day dates inside one record are rejected,
- a date may not be authored as both an additional holiday and an additional business day in the same calendar record,
- marking a date as an additional business day is also effectively idempotent if the date is already a business day.

baseCalendar

If `baseCalendar` is provided, the record defines a new derived calendar.

For example, the record:

- `name = BRISBANE_BANK`
- `baseCalendar = AUD`

means:

- start with the existing `AUD` calendar,
- create a new named calendar `BRISBANE_BANK`,
- inherit the base calendar behavior,
- then apply the additional holiday and business-day overlays defined in the record.

This is the native equivalent of the older “create `CUSTOM_Japan` from `Japan`” pattern documented in the XML guide.

12.9 Base-calendar derivation rules

The runtime loader applies calendar adjustments in two stages:

- first it amends existing calendars that do not declare a `baseCalendar`,
- then it creates and overlays the derived calendars that do declare a `baseCalendar`.

That sequencing is important because it means:

- derived calendars inherit the already-amended state of their base calendar,
- a derived calendar should be based on an existing known calendar, not on another new derived calendar authored in the same batch.

The XML-era comment that new calendars should not be defined in terms of other new calendars still applies conceptually in the maintained native model.

12.10 Compatibility XML and the `CUSTOM_` prefix

The generated compatibility file `calendaradjustment_compat.xml` preserves the historical `CalendarAdjustments` structure for non-migrated readers.

The important distinction is:

- in the maintained native YAML / JSON surface, a derived calendar can have any valid name such as `BRISBANE_BANK`,
- in the historical XML + XSD world, new derived calendars often needed a `CUSTOM\_` prefix in order to pass schema validation.

So the `CUSTOM\_` rule is now a compatibility concern, not a native authoring requirement. It remains worth documenting because users may still see the generated compatibility XML or older examples that rely on it.

12.11 Practical examples

12.11.0.1 Overlaying an existing calendar. To add one holiday and one business day to `AUD`, author a record with:

- `name = AUD`
- no `baseCalendar`
- the extra dates under `additionalHolidays` and `additionalBusinessDays`.

12.11.0.2 Creating a derived calendar. To create a Brisbane-specific banking calendar from `AUD`, author:

- `name = BRISBANE_BANK`
- `baseCalendar = AUD`
- any Brisbane-specific extra holiday or business-day adjustments.

This derived calendar then becomes available by its new name in the runtime bundle.

12.12 Operational interpretation

The maintained interpretation of the current calendar-adjustment surface is:

- use calendar overlays to patch isolated holidays or business days without forking full calendar definitions,
- use `baseCalendar` to create a named derived calendar when a local market calendar needs to inherit from a broader base,
- publish the resolved JSON artifact as the runtime and evidence contract,
- treat the generated XML only as a compatibility view.

This preserves the useful practical detail of the old `calendaradjustment.xml` chapter while documenting the current native surface precisely.

12.13 Curve Set And Runtime Curve Configuration

The current native curve-configuration surface separates:

- the selected curve set,
- the catalogue of curve families,
- the concrete build recipes,
- the dependency map used for validation and evidence.

The runtime publication is `resolved_curve_configuration.json`. It is assembled from:

- `curve_set.toml`
- `curve_family_registry.yaml`
- `curve_build_recipes.yaml`

A representative record is shown in listing 11.

Listing 11: Current resolved curve-configuration publication

```
{
  "curveSet": {
    "setId": "getting_started_minimal",
    "description": "Minimal curve configuration selection for the Getting Started baseline examples."
  },
  "curveFamilyRegistry": {
    "families": [
      {
        "familyId": "yield_curves",
        "curveType": "Yield",
        "xmlParent": "YieldCurves",
        "xmlTag": "YieldCurve"
      },
      {
        "familyId": "default_curves",
        "curveType": "Default",
        "xmlParent": "DefaultCurves",
        "xmlTag": "DefaultCurve"
      }
    ]
  },
  "curveBuildRecipes": {
    "records": [
      {
        "familyId": "yield_curves",
        "curveId": "EUR1D",
        "recipeVariant": "direct_zero",
        "dependencies": {
          "quotes": ["ZERO/RATE/EUR/EUR1D/A360/1Y"],
          "conventions": ["EUR-ZERO-CONVENTIONS-TENOR-BASED"]
        },
        "definition": {
          "Currency": "EUR",
          "InterpolationVariable": "Discount",
          "InterpolationMethod": "LogLinear"
        }
      }
    ]
  }
}
```

Publication structure

The resolved publication contains:

- `curveSet`: identifies the selected curve set and its description.
- `curveFamilyRegistry`: enumerates supported families with `familyId`, `curveType`, `xmlParent`, `xmlTag` and `description`.
- `curveBuildRecipes`: the concrete curve definitions grouped into records.
- `curveDependencyMap`: dependency summary per curve, listing `quotes`, `conventions` and `relatedCurveIds`.
- `reportConfiguration`: optional reporting-specific defaults.
- `records`: flattened access to the build-recipe records.

The `xmlParent` and `xmlTag` fields are carried only for compatibility reconstruction and provenance. The maintained guide should reason about the native fields `familyId`, `curveType`, `recipeVariant`, `dependencies` and `definition`.

12.13.1 Yield and Default Curve Structures

The common examples already publish yield and default curves through the native surface. The main definition keys seen in maintained records are:

12.13.1.1 Yield / YieldCurve

- `Currency`
- `DiscountCurve`
- `Segments`
- `InterpolationVariable`
- `InterpolationMethod`
- `YieldCurveDayCounter`
- `Tolerance`
- `BootstrapConfig`, `Extrapolation`, `MixedInterpolationCutoff`, `Report`

The curve record also carries a dependency map listing the quotes, conventions and any related curves needed for the build.

12.13.1.2 Default / DefaultCurve

- `Currency`
- `Type`
- `DiscountCurve`
- `DayCounter`
- `RecoveryRate`

- Quotes
- Conventions
- for the richer DefaultCurve family: AllowNegativeRates, BenchmarkCurve, Configurations, Extrapolation, InitialState, Pillars, SourceCurve, SpotLag, States

12.13.2 Swaption Volatility Structures

Swaption-volatility records are published with curveType = SwaptionVolatility. The normalized definition keys in current native records include:

- Dimension
- VolatilityType
- Extrapolation
- Interpolation
- DayCounter, Calendar, BusinessDayConvention
- OptionTenors, SwapTenors
- ShortSwapIndexBase, SwapIndexBase
- richer examples may also use ModelShift, OutputShift, OutputVolatilityType, SmileOptionTenors, SmileSwapTenors, SmileSpreads, ParametricSmileConfiguration, ProxyConfig, QuoteTag, Report

12.13.3 FX Volatility Structures

FX-volatility records are published with curveType = FXVolatility. The maintained normalized keys include:

- FXSpotID
- FXDomesticCurveID, FXForeignCurveID
- Dimension, VolatilityType, Extrapolation
- Calendar, BusinessDayConvention, DayCounter
- Expiries, Deltas, Quotes
- SmileDelta, SmileType, SmileInterpolation, TimeInterpolation, TimeWeighting, ButterflyErrorTolerance, Conventions, Report

12.13.4 Equity Curve Structures

Equity curves are published with curveType = EquityCurve. The current normalized definition keys include:

- Currency
- SpotQuote
- Quotes

- ForecastingCurve
- Type
- Calendar, DayCounter, Extrapolation
- DividendInterpolation, DividendExtrapolation
- ExerciseStyle

12.13.5 Equity Volatility Structures

Equity-volatility records are published with curveType = EquityVolatility. The current native keys include:

- EquityId
- Currency
- Dimension
- Expiries, Strikes, StrikeSurface
- Calendar, DayCounter
- PreferOutOfTheMoney
- VolatilityConfig, Report

12.13.6 Inflation Curves

Inflation-curve records are published with curveType = InflationCurve. The normalized definition keys include:

- Type
- BaseRate
- NominalTermStructure
- Quotes
- Conventions
- Calendar, DayCounter, Lag, Frequency, Extrapolation, Tolerance, UseLastFixingDate, Seasonality

12.13.7 Inflation Cap/Floor Volatility Surfaces

Inflation cap/floor volatility records are published as InflationCapFloorVolatility. The current normalized keys include:

- Type, QuoteType, VolatilityType
- Index, IndexCurve, IndexInterpolated
- YieldTermStructure
- Tenors, Strikes, CapStrikes, FloorStrikes

- SettlementDays, ObservationLag, Calendar, BusinessDayConvention, DayCounter
- BootstrapConfig, Conventions, Extrapolation, Report

12.13.8 CDS Volatilities

CDS-volatility records are published with curveType = CDSVolatility. The current normalized keys include:

- QuoteName
- Expiries, Terms
- StrikeFactor, StrikeSurface, StrikeType
- Calendar, DayCounter

12.13.9 Base Correlations

Base-correlation records are published with curveType = BaseCorrelation. The current normalized keys include:

- QuoteName
- DetachmentPoints, Terms
- Rule, Extrapolate, AdjustForLosses
- Calendar, BusinessDayConvention, DayCounter, SettlementDays, StartDate

12.13.10 Commodity Curves and Volatilities

The current native surface also carries explicit commodity families.

12.13.10.1 CommodityCurve

- Currency
- SpotQuote
- Quotes
- Conventions
- DayCounter, InterpolationMethod, Extrapolation

12.13.10.2 CommodityVolatility

- Currency
- Curve, PriceCurveId, YieldCurveId
- FutureConventions
- StrikeSurface, VolatilityConfig
- Calendar, DayCounter, OptionExpiryRollDays, Report

12.13.11 Securities

Security records are published with `curveType = Security`. The normalized keys include:

- `PriceQuote`
- `RecoveryRateQuote`
- `SpreadQuote`
- `CPRQuote`
- `ConversionFactor`

12.13.12 Correlations

Standalone market-correlation curve records are published with `curveType = Correlation`. The current native keys include:

- `CorrelationType`
- `Index1`, `Index2`
- `Dimension`, `OptionTenors`, `QuoteType`
- `Conventions`, `Currency`
- `Calendar`, `BusinessDayConvention`, `DayCounter`, `Extrapolation`

12.13.13 Bootstrap Configuration

Bootstrap and solver metadata is no longer hidden inside arbitrary XML subtrees. In the native surface it appears explicitly in the definition payload:

- `BootstrapConfig` on families such as `YieldCurve`, `CapFloorVolatility` and `InflationCapFloorVolatility`
- `Tolerance` and family-specific interpolation settings on yield and inflation curves
- dependency lists in `curveDependencyMap` for quotes, conventions and related curves

This separation makes it much easier for Config Doctor and CI to report missing quotes, unresolved conventions or inconsistent curve dependencies.

12.13.14 One Dimensional Solver Configuration

Solver-related settings are likewise expressed natively through explicit fields such as:

- `Tolerance`
- `InterpolationMethod`
- `InterpolationVariable`
- family-specific bootstrap or extrapolation controls

The important current design rule is that the solved curve remains a machine-readable recipe plus dependency map, not a monolithic free-form XML definition.

12.14 Reference Data Manifest And Registries

Reference data is still used to keep static or slowly changing security and entity attributes out of the trade payloads themselves. What has changed is the maintained contract surface. The current guide no longer treats reference data as a hand-authored `referencedata.xml` file. Instead, the maintained native surface is split into:

- an authored manifest `reference_data_manifest.yaml`,
- authored domain registries such as `credit_reference_registry.json`, `equity_reference_registry.json`, `commodity_reference_registry.json` and `portfolio_basket_reference.json`,
- published normalised registries under `published/`,
- a resolved runtime publication `published/resolved_reference_data.json`,
- a generated compatibility file `published/referencedata_compat.xml` for readers that still need the historical XML view.

The canonical machine publication is therefore `examples/common/input/reference_-registry/published/resolved_reference_data.json`.

Current top-level publication

A representative manifest extract is shown in listing 12.

Listing 12: Current reference-data manifest publication

```
{
  "schemaVersion": "1.0.0",
  "seedVersion": "1.0.0",
  "entries": [
    {
      "referenceId": "EQ_SPX",
      "validFrom": "2026-03-13",
      "type": "Equity",
      "registry": "equity_reference_registry.json"
    },
    {
      "referenceId": "BOND_US912828X703",
      "validFrom": "2026-03-13",
      "type": "Bond",
      "registry": "credit_reference_registry.json"
    },
    {
      "referenceId": "PB_SPX_NDX",
      "validFrom": "2026-03-13",
      "type": "PortfolioBasket",
      "registry": "portfolio_basket_reference.json"
    }
  ]
}
```

At runtime the resolved publication joins the manifest with the domain registries and exposes three useful views:

- `referenceDataManifest`: the published and normalised manifest,
- `domainRegistries`: the published per-domain registries,

- `referenceData`: the flattened runtime-facing list of resolved reference-data entries.

The resolved file also carries:

- `schemaVersion`,
- `publishedAt`,
- `sourceLineage.manifestPath`,
- `contentHash`.

This structure is materially better than the old monolithic XML file for publication, auditing, Config Doctor retrieval and cross-repo traceability.

Supported current families

The common current manifest demonstrates the maintained reference-data families:

- `Equity`
- `EquityIndex`
- `CurrencyHedgedEquityIndex`
- `Credit`
- `Bond`
- `CreditIndex`
- `Commodity`
- `PortfolioBasket`

This is broader than the old user-guide chapter, which mainly elaborated the bond case. The bond case remains important and is still documented in detail below because it is the reference-data family most directly tied to trade pricing and fitted bond curve construction.

12.15 Manifest entries

Each entry in `referenceDataManifest.entries[]` and in the authored manifest identifies exactly one reference-data record and the registry that owns it.

- `referenceId`: unique reference-data identifier. This is the native equivalent of the historical `ReferenceDatum id` attribute.
- `validFrom`: effective-from date for the record. The current surface uses ISO dates.
- `type`: reference-data family such as `Bond`, `Credit`, `Equity` or `PortfolioBasket`.
- `registry`: target domain registry that owns the payload.

The manifest is the navigation layer. It says what reference data exists, starting when it is valid, what type it is, and which domain registry contains the payload.

12.16 Domain registry records

Each domain registry stores a list of typed records. The current normalised records carry:

- `referenceId`
- `validFrom`
- `type`
- `metadata`
- `payloadXml`

After resolution, the flattened runtime list under `referenceData` adds:

- `registry`: the owning registry name,
- `payloadRoot`: the original payload root, for example `BondReferenceData` or `EquityReferenceData`.

The current surface therefore separates three concerns that used to be mixed in the XML file:

- identity and effective dating in the manifest,
- payload ownership in the domain registry,
- runtime joining and hashing in the resolved publication.

12.17 Bond reference data

Bond reference data remains the most important detailed case because it is used to keep bond static information out of individual trades and because it feeds curve and analytics workflows beyond plain trade loading.

Current uses include:

- populating bond trades, forward bonds, bond options, bond repos and convertible bonds where the trade carries only a security identifier or a partially specified bond block,
- building fitted bond curves and shifted bond curves, where bond static information and schedule detail are required by the helper instruments,
- market-risk and CRIF qualification logic that needs bond issuer, credit quality or securitisation attributes.

The resolved record for a bond keeps the manifest identity fields plus:

- `payloadRoot = BondReferenceData`,
- `metadata` with a small summary such as issuer and currency,
- `payloadXml` containing the detailed bond payload.

The maintained guide is native-first, so the detail below is described as a field catalogue rather than by reproducing the old XML snippet verbatim.

Bond identity fields

- `referenceId`: the security identifier, often following an external bond naming scheme such as ISIN.
- `validFrom`: the effective-from date of this reference-data version.
- `type`: must be `Bond` for a plain bond reference datum.

Bond payload fields

The bond payload itself supports the following fields. These remain the practically important bond static attributes, even though they are now carried through the registry publication rather than authored directly in one XML file.

- `SubType` [Optional]: reporting and product-taxonomy subtype. Legacy allowable values such as `ABS`, `Corp`, `Loans`, `Muni`, `Sovereign`, `ABX`, `CMBX`, `MBX`, `PrimeX`, `TRX` and `iBoxx` remain the useful classification vocabulary where applicable.
- `IssuerId`: issuer or obligor identifier for the bond. This is central to pricing, credit-risk and market-risk classification.
- `CreditCurveId` [Optional]: credit-curve identifier used where the bond should be linked to a named credit curve.
- `CreditGroup` [Optional]: optional grouping identifier for related credit names.
- `ReferenceCurveId`: reference discounting / benchmark curve for the bond.
- `IncomeCurveId` [Optional]: optional separate income curve identifier when the workflow distinguishes carry and reference discounting.
- `VolatilityCurveId` [Optional]: optional volatility curve identifier for workflows that require one.
- `RecoveryRate` [Optional]: fixed recovery used in the bond reference datum when a workflow requires an explicit recovery input.
- `SettlementDays`: settlement lag for the bond.
- `Calendar`: settlement calendar.
- `IssueDate`: issue date of the bond.
- `PriceQuoteMethod` [Optional]: price-quote method, when needed by the downstream pricing or reporting flow.
- `PriceQuoteBaseValue` [Optional]: base value used with the price-quote method.
- `quotedDirtyPrices` [Optional]: internal clean / dirty price interpretation derived from the historical `PriceType` field. The engine maps `Clean` and `Dirty` and falls back to clean if the provided value is invalid.
- `CreditQuality` [Optional]: credit-quality qualifier carried with the bond static data, for example for market-risk CSR classification.
- `Sector` [Optional]: sector classification.

- `IsSecuritisation` [Optional]: boolean flag indicating whether the bond is treated as a securitisation.
- `SecuritisationType` [Optional]: securitisation subtype such as RMBS or CMBS where the workflow needs one.
- `LegData`: one or more bond leg definitions carrying coupon, schedule, currency, day-count and notional structure.

LegData inside bond reference data

The `LegData` block remains one of the most important parts of the bond reference datum. It is what allows VRE to reconstruct a priced bond from a reference-data identifier rather than requiring each trade to inline all of the coupon and schedule mechanics.

For the common fixed-rate case, the bond leg specifies:

- leg type,
- payer flag,
- currency,
- payment convention,
- day-count convention,
- notionals and exchange flags,
- schedule rules,
- fixed coupon rates.

This is the field family that the legacy guide expressed as inline XML under `BondReferenceData/LegData`. In the current guide the same content is still normative, but it is owned by the reference-data payload in the registry publication rather than by a top-level hand-edited XML file.

Operational consequences

The practical effect of bond reference data has not changed:

- when a bond trade omits empty static fields, VRE can populate them from the bond reference datum,
- fitted bond curve builders require reference data in order to create the bond helpers consistently,
- shifted bond curves and market-risk analytics can use the same bond reference information without duplicating static attributes inside trade payloads.

This is why the bond field catalogue still matters at the same level of detail as in the old guide.

12.18 Resolved runtime view

The resolved runtime package flattens the manifest and registries into `referenceData.entries[]`. Each resolved entry contains:

- referenceId
- validFrom
- type
- registry
- payloadRoot
- metadata
- payloadXml

The important runtime rule is that every manifest entry must resolve to an actual record in the named registry. The publisher rejects unresolved entries, and resolved ordering is deterministic.

The metadata block is intentionally lightweight. It is there to make the resolved publication easier to search and reason about without having to parse the full payload every time. The full payload remains preserved in payloadXml, which is why the resolved package can still emit the compatibility XML view losslessly.

12.19 Compatibility XML

The publisher emits published/referencedata_compat.xml. This file is generated from resolved_reference_data.json; it is not the source of truth. The current guide therefore treats the compatibility XML as a downstream view only.

That distinction matters operationally:

- new authoring and review work should happen in the manifest and domain registries,
- runtime consumers should preferentially use the resolved JSON package,
- the compatibility XML exists only for readers that have not yet migrated away from the old surface.

12.20 Operational interpretation

The maintained interpretation of the current reference-data surface is:

- define the inventory of reference objects in the manifest,
- keep each family in its own domain registry,
- publish a deterministic resolved package for runtime consumption,
- preserve the detailed bond, credit, equity, commodity and basket payloads inside the resolved registry entries,
- treat XML as generated compatibility only.

This preserves the original purpose of the old referencedata.xml chapter while documenting the current native contract at the same practical level.

12.21 IBOR Fallback Policy

The IBOR fallback configuration represents the rules for replacing legacy IBOR reference rates with risk-free-rate based fallbacks. The maintained current surface is no longer a hand-authored `iborFallbackConfig.xml` file. Instead, the current workflow uses:

- an authored seed `examples/common/input/runtime_policy/ibor_fallback_policy.yaml`,
- a normalised publication `published/ibor_fallback_policy.json`,
- a resolved runtime contract `published/resolved_ibor_fallback_policy.json`,
- a generated compatibility file `published/iborfallbackconfig_compat.xml`.

The canonical runtime publication is `examples/common/input/runtime_policy/published/resolved_ibor_fallback_policy.json`.

Listing 13: Current resolved IBOR fallback publication


```
{
  "schemaVersion": "1.0.0",
  "seedVersion": "1",
  "publishedAt": "2026-03-13T00:00:00Z",
  "contentHash": "a66458109b135f3f7fdaffd14dfe473ed9c77f34af4eb2e29f899641c0352d7a",
  "sourceLineage": {
    "seedPath": "examples/common/input/runtime_policy/ibor_fallback_policy.yaml",
    "generator": "tools/ci/publish_ibor_fallback_contracts.py"
  },
  "globalSettings": {
    "enableFallbacks": true,
    "useRfrCurveInTodaysMarket": true,
    "useRfrCurveInSimulationMarket": false
  },
  "fallbacks": [
    {
      "iborIndex": "GBP-LIBOR-3M",
      "rfrIndex": "GBP-SONIA",
      "spread": 0.001193,
      "switchDate": "2022-01-01",
      "versionedId": "GBP-LIBOR-3M::1"
    },
    {
      "iborIndex": "USD-LIBOR-3M",
      "rfrIndex": "USD-SOFR",
      "spread": 0.0026161,
      "switchDate": "2023-07-01",
      "versionedId": "USD-LIBOR-3M::1"
    }
  ]
}
```


Current contract role

The current publication separates:

- global runtime switches controlling whether fallback replacement is active and where it is applied,
- per-index fallback rules,
- publication metadata such as schema version, publication timestamp and content hash,
- source lineage back to the curated native policy seed.

This is operationally better than the historical XML-only form because a run can now prove exactly which fallback policy was published, when it was published, and which native seed produced it.

12.22 Authored policy seed

The authored seed is a compact YAML policy document:

Listing 14: Current authored IBOR fallback seed

```yaml
version: "1"
traceabilityRefs:
  adr_0001: docs/architecture/adr/0001-serialization-and-storage-format-standards.md
global:
  enableFallbacks: true
  useRfrCurveInTodaysMarket: true
  useRfrCurveInSimulationMarket: false
fallbacks:
  - iborIndex: GBP-LIBOR-3M
    rfrIndex: GBP-SONIA
    spread: 0.001193
    switchDate: "2022-01-01"
  - iborIndex: USD-LIBOR-3M
    rfrIndex: USD-SOFR
    spread: 0.0026161
    switchDate: "2023-07-01"
```

The publisher validates that:

- the top-level `version`, `traceabilityRefs`, `global` and `fallbacks` blocks are present,
- every fallback rule has a unique `iborIndex`,
- every `switchDate` is a valid ISO date,
- every fallback rule has a non-empty replacement `rfrIndex`.

12.23 Global settings

The resolved JSON loader and the historical XML loader both recognise the same three global controls. Their current native names are:

- `enableFallbacks`
- `useRfrCurveInTodaysMarket`
- `useRfrCurveInSimulationMarket`

enableFallbacks

`enableFallbacks` is the master on/off switch. If it is `false`, IBOR fallback replacement is disabled even if individual fallback rules are present.

Operationally this means:

- the replacement map is retained,
- but `isIndexReplaced()` will return false for all indices,

- and no IBOR index is switched to its fallback RFR curve.

This is the cleanest way to disable fallback replacement in controlled testing.

useRfrCurveInTodaysMarket

If `useRfrCurveInTodaysMarket` is `true`, the today-market IBOR forwarding curve for a replaced index is built using:

- the replacement `rfrIndex` OIS curve,
- plus the configured fallback `spread`.

If it is `false`, today-market construction keeps the original IBOR curve logic even if the index is otherwise marked as replaced.

This switch therefore controls whether fallback replacement affects deterministic pricing and current-state risk calculations that are driven from the today market.

useRfrCurveInSimulationMarket

If `useRfrCurveInSimulationMarket` is `true`, the simulation-market forwarding curve for a replaced IBOR index is built from the replacement RFR OIS curve plus the spread.

If it is `false`, simulation continues to use the pre-fallback IBOR curve setup even though the today market may already have pivoted.

This distinction matters when:

- today-market valuation should reflect the post-cessation fallback,
- but simulation or legacy exposure testing is still being run under an older forward-curve assumption.

Loader defaults

The runtime configuration object uses the following defaults when no custom policy is loaded:

- `enableFallbacks = true`
- `useRfrCurveInTodaysMarket = true`
- `useRfrCurveInSimulationMarket = false`

These are the same values assigned in `IborFallbackConfig::clear()` and in the built-in default configuration.

12.24 Fallback rules

Each entry in `fallbacks[]` defines the replacement of one IBOR index. The current native rule fields are:

- `iborIndex`: canonical IBOR index name to be replaced.
- `rfrIndex`: replacement RFR index name.
- `spread`: additive spread applied to the replacement RFR rate.

- `switchDate`: date from which the replacement becomes active.
- `versionedId`: publisher-generated stable identifier combining the IBOR index and the seed version.

iborIndex

`iborIndex` identifies the legacy rate to be replaced, for example `GBP-LIBOR-3M` or `USD-LIBOR-3M`. The publisher requires these identifiers to be unique inside the policy.

rfrIndex

`rfrIndex` identifies the OIS / RFR curve that will replace the IBOR curve once the switch date is reached. Typical current examples are:

- `GBP-SONIA` for GBP LIBOR,
- `USD-SOFR` for USD LIBOR,
- `EUR-ESTER` for EUR fallbacks,
- `JPY-TONAR` for JPY fallback families.

spread

`spread` is the additive fallback spread applied to the replacement RFR rate. It is a required numeric field and may be positive or negative.

The current examples show both cases:

- positive fallback spreads such as `0.0026161` for `USD-LIBOR-3M`,
- negative fallback spreads for some short-tenor historical rules in the built-in default policy.

switchDate

`switchDate` is the decisive activation date. An index is treated as replaced if and only if:

- fallbacks are globally enabled,
- a rule exists for the index,
- the as-of date is on or after the configured switch date.

This is the exact runtime rule implemented by `IborFallbackConfig::isIndexReplaced()`.

versionedId

`versionedId` is generated during publication. It is not authored by hand. Its purpose is to provide a stable identity for evidence, publication diffs and future control-plane workflows. The current form is:

- `iborIndex::seedVersion`

for example `USD-LIBOR-3M::1`.

12.25 Built-in default configuration

If no custom configuration is specified, VRE uses a built-in standard fallback configuration from `IborFallbackConfig::defaultConfig()`.

That built-in policy contains a much broader fallback universe than the compact example seed, including:

- CHF LIBOR tenors with SARON replacement,
- EUR EONIA / EURIBOR / EUR LIBOR families with ESTER replacement,
- JPY LIBOR, TIBOR and Euroyen TIBOR families with TONAR replacement,
- AUD BBSW with AONIA replacement,
- HKD HIBOR with HONIA replacement,
- CAD CDOR with CORRA replacement,
- GBP LIBOR with SONIA replacement,
- USD LIBOR with SOFR replacement,
- TRY TRLIBOR with TLREF replacement.

The standard configuration is the production default. A custom published policy is therefore mainly used when:

- a jurisdictional or client-specific override is needed,
- regression or migration testing needs deterministic control of switch dates or spreads,
- release evidence must point to an explicit published policy artifact.

12.26 Testing utilities and switch-date overrides

The runtime configuration object also provides `updateSwitchDate(targetSwitchDate, indexName)`. This allows switch dates to be moved earlier for testing without having to author a completely separate fallback seed.

The important behavior is:

- only rules whose switch date is after the target date are updated,
- the override may apply to one index or to all indices,
- the change is logged as a structured configuration warning.

This utility exists for controlled testing. It is not a substitute for maintaining the published fallback policy.

12.27 Compatibility XML

The publisher emits `iborfallbackconfig_compat.xml`. This file is a generated compatibility view of the native policy. It preserves the historical global settings and fallback-rule structure for readers that still expect the old XML root.

The maintained rule is:

- author and review the YAML seed,
- publish JSON and resolved JSON,
- treat XML only as a generated bridge for non-migrated readers.

12.28 Operational interpretation

The current maintained interpretation of the fallback policy is therefore:

- choose whether fallback replacement is globally enabled,
- decide independently whether today-market and simulation-market curve construction should pivot to the replacement RFR curve,
- define one replacement rule per IBOR index with an RFR index, spread and switch date,
- rely on the built-in default policy when no custom policy is needed,
- use the published resolved artifact as the runtime and evidence contract.

This preserves the full operational meaning of the old `iborFallbackConfig.xml` chapter while documenting the current native surface precisely.

12.29 SIMM Registry And Calibration

The maintained SIMM calibration surface is no longer a monolithic `simmcalibration.xml`. The current model publishes:

- package metadata in `simm_calibration_package.yaml`,
- tabular calibration data in Parquet tables,
- a resolved JSON publication in `resolved_simm_calibration.json`.

The publication is designed for two distinct consumers:

- the engine runtime, which needs deterministic calibration lookup by SIMM version name;
- validation and audit tooling, which needs explicit row counts, schemas, provenance and exact compatibility fragments.

See the initial-margin example packs and the common SIMM registry under `examples/common/input/simm_registry/`. The maintained publication is shown in listing 15.

Listing 15: Current SIMM calibration publication

```
{
  "sourceLineage": {
    "packagePath": "examples/common/input/simm_registry/simm_calibration_package.yaml"
  },
  "simmCalibrationPackage": {
    "tableFiles": {
      "riskWeights": "tables/risk_weights.parquet",
      "correlations": "tables/correlations.parquet",
      "concentrationThresholds": "tables/concentration_thresholds.parquet",
      "riskClassCorrelations": "tables/risk_class_correlations.parquet"
    },
    "calibrations": [
      {
        "calibrationId": "2.5A official",
        "versionNames": ["2.5A", "2.5.a"]
      }
    ]
  }
}
```

12.29.1 SIMM Calibration

The authored package declares the calibration identities and the table files to be read. A package entry contains:

- `calibrationId`: stable calibration identifier.
- `versionNames`: one or more aliases that may be supplied in the run bundle `analytics.simm.version` field.
- `additionalFieldsXml`: compatibility and descriptive fields, for example `SIMM_EffectiveDate` and stress-period metadata.

The package also lists the four published table families:

- `riskWeights`

- correlations
- concentrationThresholds
- riskClassCorrelations

This replaces the old nested XML tree with a calibration registry plus typed table files.

12.29.2 General Structure

The resolved publication contains:

- schemaVersion, publishedAt, contentHash
- sourceLineage.packagePath
- simmCalibrationPackage
- tables: metadata about each Parquet table
- tableRows: the resolved rows exposed in JSON form

Each entry under tables carries:

- path: absolute logical path of the source table
- defaultName: the expected local file name
- rowCount: number of rows in the table
- schema: column schema

For example, the riskWeights table schema contains:

- calibration_id
- risk_class
- entry_name
- order_index
- mpor_days
- xml_fragment

The explicit table schema is a significant improvement over the historical XML surface because it makes it possible to validate coverage, ordering and type consistency without parsing a large free-form document.

12.29.3 Interest Rate

Interest-rate SIMM calibration rows are represented in the same native table model as the other risk classes. The riskWeights table contains rows for Delta, Vega and, where relevant, HistoricalVolatilityRatio, keyed by:

- calibration_id
- risk_class = InterestRate
- entry_name

- `mpor_days`
- `order_index`

The XML fragment is preserved verbatim in `xml_fragment` for deterministic compatibility reconstruction, but the maintained native guide should reason about the row keys, MPOR partition and risk-class grouping.

Interest-rate calibrations also carry the usual SIMM family concepts:

- delta and vega risk weights,
- intra-bucket and inter-bucket correlations,
- concentration thresholds,
- the IR-specific currency-grouping logic that still matters for SIMM aggregation.

12.29.4 Credit Qualifying

Credit-qualifying rows are published under `risk_class = CreditQualifying`. The native row model captures:

- bucketed delta risk weights,
- vega risk weights,
- the optional base-correlation weight used by the credit-qualifying class,
- concentration thresholds,
- qualifying-credit correlation rows.

In the current resolved publication, this appears as separate rows with `entry_name` values such as `Delta`, `Vega` and `BaseCorrelation`, partitioned by `mpor_days`.

12.29.5 Credit Non-Qualifying

Credit-non-qualifying rows are published under `risk_class = CreditNonQualifying`. The same native row model applies:

- bucketed delta weights,
- vega weights,
- correlation rows,
- concentration thresholds.

The current row model makes the residual bucket explicit in the underlying XML fragment and retains it as an auditable compatibility artifact while still grouping the rows natively by calibration id, risk class and MPOR.

12.29.6 Equity

Equity SIMM calibration rows are published under `risk_class = Equity`. The resolved publication shows:

- bucketed delta risk weights,

- bucketed vega risk weights,
- `HistoricalVolatilityRatio` rows,
- the corresponding correlation and concentration-threshold tables.

This mirrors the historical XML content, but is much easier to diff and audit because each entry is a row with explicit calibration id, MPOR and order.

12.29.7 Commodity

Commodity SIMM calibration rows are published under `risk_class = Commodity`. The current publication exposes:

- delta rows for multiple buckets,
- vega rows,
- historical-volatility-ratio rows,
- commodity correlation and concentration-threshold tables.

The resolved JSON is especially helpful here because the commodity class often has a large number of buckets, and the table representation makes the bucketed weights much easier to validate than in the historical XML layout.

12.29.8 FX

FX SIMM calibration rows are published under `risk_class = FX`. The native publication model preserves the historical distinctions between:

- FX delta risk weights,
- FX vega risk weights,
- historical-volatility ratios,
- FX concentration thresholds,
- the currency-group semantics that remain relevant for FX SIMM.

As with the interest-rate class, the current guide should think of these as typed rows with calibration identity and MPOR partition, not as nested XML subtrees.

12.29.9 Risk-class correlations

Risk-class correlations are separated from the risk-class-specific tables and published under `tableRows.riskClassCorrelations`. Each row carries:

- `calibration_id`
- `label1`
- `label2`
- `order_index`
- `value`

- `xml_fragment`

This is the native successor of the historical <RiskClassCorrelations> section. The explicit row form makes it easy to validate symmetry, ordering and completeness across risk classes.

12.29.10 Operational rule

The run bundle should reference the published resolved calibration, for example through `analytics.simm.simmCalibration` or the corresponding initial-margin runtime fields. Users should not hand-edit ad hoc XML calibration files for maintained examples. The native package-plus-table model is now the source of truth.

12.30 Convention Manifest And Registry

The maintained conventions surface is a published registry rather than a monolithic hand-authored `conventions.xml`. The current runtime-facing artifacts are:

- `reference_registry/published/convention_manifest.json`
- `reference_registry/published/convention_registry.json`

The manifest provides publication metadata and provenance. The registry carries the actual convention families grouped by business purpose. A representative extract is shown in listing 16.

Listing 16: Current convention-registry publication

```
{
  "swap": [
    {
      "id": "EUR-6M-SWAP-CONVENTIONS",
      "terms": {
        "FixedCalendar": "TARGET",
        "FixedConvention": "MF",
        "FixedDayCounter": "A365",
        "FixedFrequency": "Annual",
        "Index": "EUR-EURIBOR-6M"
      }
    }
  ],
  "fx": [
    {
      "id": "EUR-GBP-FX-CONVENTIONS",
      "terms": {
        "AdvanceCalendar": "TARGET,UK",
        "PointsFactor": 10000,
        "SourceCurrency": "EUR",
        "SpotDays": 2,
        "SpotRelative": true,
        "TargetCurrency": "GBP"
      }
    }
  ]
}
```

The native model preserves the breadth of the historical convention families, but each family is now a separately addressable list of records with:

- `id`: the convention identifier,
- `terms`: the normalized parameter payload.

12.30.1 Zero Conventions

Zero conventions are published under the `zero` family. The normalized term keys supported by the current publication model are:

- `Compounding`
- `CompoundingFrequency`
- `DayCounter`
- `EOM`

- RollConvention
- SpotCalendar
- SpotLag
- TenorBased
- TenorCalendar

12.30.2 Deposit Conventions

Deposit conventions are published under deposit with the current native keys:

- Index
- IndexBased

12.30.3 Future Conventions

Future conventions are published under future. The current term keys are:

- DateGenerationRule
- Index
- OvernightIndexFutureNettingType

12.30.4 FRA Conventions

FRA conventions are published under fra with the current normalized key:

- Index

12.30.5 OIS Conventions

OIS conventions are published under ois. The current term keys are:

- Index
- SpotLag
- Rule
- EOM
- FixedFrequency
- FixedConvention
- FixedPaymentConvention
- FixedDayCounter
- PaymentLag
- RateCutoff

12.30.6 Swap Conventions

Swap conventions are published under `swap`. The current term keys are:

- `Index`
- `FixedCalendar`
- `FixedConvention`
- `FixedDayCounter`
- `FixedFrequency`
- `FloatFrequency`
- `SubPeriodsCouponType`

12.30.7 Average OIS Conventions

Average-OIS conventions are published under `average\_ois`. The current term keys are:

- `Index`
- `OnTenor`
- `SpotLag`
- `RateCutoff`
- `FixedCalendar`
- `FixedConvention`
- `FixedPaymentConvention`
- `FixedDayCounter`
- `FixedTenor`

12.30.8 Tenor Basis Swap Conventions

Tenor-basis conventions are published under `tenor\_basis\_swap` with the current term keys:

- `PayIndex`, `ReceiveIndex`
- `PayFrequency`, `ReceiveFrequency`
- `SpreadOnRec`
- `IncludeSpread`
- `SubPeriodsCouponType`

12.30.9 Tenor Basis Two Swap Conventions

The second tenor-basis family is published under `tenor\_basis\_two\_swap`. The current keys are:

- `Calendar`

- LongIndex, ShortIndex
- LongFixedFrequency, ShortFixedFrequency
- LongFixedConvention, ShortFixedConvention
- LongFixedDayCounter, ShortFixedDayCounter
- LongMinusShort

12.30.10 BMA / SIFMA Basis Swap Conventions

BMA/SIFMA basis conventions are published under bma_basis_swap:

- BMAIndex
- Index
- LiborIndex

12.30.11 FX Conventions

FX conventions are published under fx with the current term keys:

- SourceCurrency
- TargetCurrency
- AdvanceCalendar
- Convention
- SpotDays
- SpotRelative
- PointsFactor

12.30.12 Cross Currency Basis Swap Conventions

Cross-currency basis conventions are published under cross_currency_basis:

- SettlementCalendar
- SettlementDays
- SpreadIndex, SpreadTenor
- FlatIndex, FlatTenor
- IsResettable, FlatIndexIsResettable
- RollConvention

12.30.13 Cross Currency Fix-Float Conventions

The native registry also has cross_currency_fix_float for fixed-versus-floating cross-currency swaps. The current term keys are:

- FixedCurrency

- FixedFrequency
- FixedConvention
- FixedDayCounter
- Index
- SettlementCalendar
- SettlementConvention
- SettlementDays

12.30.14 Inflation Swap Conventions

Inflation swap conventions are published under inflation_swap with the current term keys:

- Index
- ObservationLag
- Interpolated
- AdjustInflationObservationDates
- FixCalendar, FixConvention
- InflationCalendar, InflationConvention
- PublicationRoll, PublicationSchedule
- DayCounter

12.30.15 CMS Spread Option Conventions

CMS spread-option conventions are published under cms_spread_option. The current term keys are:

- Calendar
- DayCounter
- FixingDays
- ForwardStart
- RollConvention
- SpotDays
- SwapTenor

12.30.16 Ibor Index Conventions

Ibor-index conventions are published under ibor_index. The current term keys are:

- FixingCalendar
- BusinessDayConvention

- SettlementDays
- DayCounter
- EndOfMonth

12.30.17 Overnight Index Conventions

Overnight-index conventions are published under overnight_index:

- FixingCalendar
- SettlementDays
- DayCounter

12.30.18 Inflation Index Conventions

Zero-inflation-index conventions are published under zero_inflation_index with the current term keys:

- RegionName, RegionCode
- Currency
- Frequency
- AvailabilityLag
- Revised

12.30.19 Swap Index Conventions

Swap-index conventions are published under swap_index:

- Conventions
- FixingCalendar

12.30.20 FX Option Conventions

FX-option conventions are published under fx_option. The current term keys are:

- FXConventionID
- AtmType, LongTermAtmType
- DeltaType, LongTermDeltaType
- RiskReversalInFavorOf
- ButterflyStyle
- SwitchTenor

12.30.21 FX Option Time Weighting Scheme

The registry keeps a distinct fx_option_time_weighting family. This preserves the historical separation between generic FX-option conventions and the time-weighting scheme used by surface interpolation, even when the active registry slice is empty in a minimal example pack.

12.30.22 Commodity Forward Conventions

Commodity-forward conventions are published under commodity_forward with the current term keys:

- AdvanceCalendar
- BusinessDayConvention
- Outright
- PointsFactor
- SpotDays
- SpotRelative

12.30.23 Commodity Future Conventions

Commodity-future conventions are published under commodity_future. The current term keys are:

- Calendar, ExpiryCalendar
- BusinessDayConvention
- ContractFrequency
- AnchorDay, OffsetDays, ExpiryMonthLag
- OptionExpiryMonthLag, OptionExpiryOffset, OptionNthWeekday
- AdjustBeforeOffset
- IsAveraging
- ProhibitedExpiries
- SavingsTime

12.30.24 Credit Default Swap Conventions

CDS conventions are published under cds. The current term keys are:

- Calendar
- DayCounter
- Frequency
- PaymentConvention
- PaysAtDefaultTime

- `Rule`
- `SettlementDays`
- `SettlesAccrual`

12.30.25 Bond Yield Conventions

Bond-yield conventions remain a distinct family under `bond\_yield`. Even where a minimal registry slice carries no active records, the family is preserved as part of the maintained publication model so that the user guide continues to document the full intended convention estate rather than only the currently populated example subset.

12.31 Historical Return Policy

Historical-return configuration determines how VRE computes returns from historical observations for scenario generation, historical VaR, backtesting and related replay workflows. The maintained current surface is no longer a hand-authored `historicalreturnconfig.xml` file. Instead, the current workflow uses:

- an authored seed `examples/common/input/runtime_policy/historical_return_policy.yaml`,
- a published manifest `published/historical_return_manifest.json`,
- a resolved runtime publication `published/resolved_historical_return_manifest.json`,
- a generated compatibility file `published/historicalreturnconfig_compat.xml`.

The canonical current runtime artifact is `examples/common/input/runtime_policy/published/resolved_historical_return_manifest.json`.

Listing 17: Current resolved historical-return publication


```
{
  "schemaVersion": "1.0.0",
  "seedVersion": "1",
  "publishedAt": "2026-03-13T00:00:00Z",
  "contentHash": "8159b7f75c0726dc8789d1001a38b39fa3349e906fb005c61862680a54c53cd8",
  "sourceLineage": {
    "seedPath": "examples/common/input/runtime_policy/historical_return_policy.yaml",
    "generator": "tools/ci/publish_historical_return_contracts.py"
  },
  "returns": [
    {
      "key": "CommodityCurve",
      "type": "Relative",
      "displacement": 0.0,
      "versionedId": "CommodityCurve::1"
    },
    {
      "key": "CommodityCurve/WTI",
      "type": "Relative",
      "displacement": 0.7,
      "versionedId": "CommodityCurve/WTI::1"
    },
    {
      "key": "DiscountCurve",
      "type": "Absolute",
      "displacement": 0.0,
      "versionedId": "DiscountCurve::1"
    }
  ]
}
```


Current contract role

The historical-return policy now captures:

- the return calculation type for each risk-factor family or specific name,
- an optional displacement used in the return calculation,
- deterministic publication metadata such as `versionedId`, content hash and source lineage,

- a native publication path that can be validated and diffed outside the analytics runtime.

This is a better operational fit for evidence-bearing scenario workflows than the historical XML-only tree.

12.32 Authored seed

The authored seed is a compact YAML mapping:

Listing 18: Current authored historical-return seed

```yaml
version: "1"
traceabilityRefs:
  adr: "docs/architecture/adr/0001-serialization-and-storage-format-standards.md"
returns:
  - key: "CommodityCurve"
    type: "Relative"
    displacement: 0.0
  - key: "CommodityCurve/WTI"
    type: "Relative"
    displacement: 0.7
  - key: "DiscountCurve"
    type: "Absolute"
    displacement: 0.0
```

The publisher validates that:

- each `returns[].key` is unique,
- `type` is one of `Absolute`, `Relative` or `Log`,
- `displacement` is optional and defaults to `0.0`.

12.33 Return entries

Each entry under `returns[]` defines the return treatment for either:

- a whole risk-factor family, for example `DiscountCurve`, `CommodityCurve` or `FXSpot`,
- or a specific named override, for example `CommodityCurve/WTI`.

The fields are:

- `key`: either `RiskFactorKeyType` or `RiskFactorKeyType/Name`.
- `type`: return-calculation mode.
- `displacement` [Optional]: displacement applied in the return formula. If omitted, defaults to `0.0`.
- `versionedId`: publisher-generated stable identifier of the form `key::seedVersion`.

Key syntax

The key syntax is important and is enforced by the runtime loader:

- `DiscountCurve` means "use this rule for all discount-curve names",
- `CommodityCurve/WTI` means "use this rule only for the WTI commodity curve",

- no deeper path nesting is allowed: the key must be either one token or a family plus one name token.

The runtime loader rejects keys that are not of the form `RiskFactorKeyType` or `RiskFactorKeyType/Name`.

Specific override versus default family rule

This is the most important behavioral rule in the chapter.

When VRE looks up the return configuration for a risk factor:

- it first searches for a specific override `(keytype, name)`,
- if no specific override exists, it falls back to the default family rule `(keytype, "")`.

So in the example above:

- `CommodityCurve/WTI` uses displacement `0.7`,
- `CommodityCurve/Brent` falls back to the family default `CommodityCurve` with displacement `0.0`.

This precedence rule is implemented directly in `ReturnConfiguration::returnType()` and should be treated as normative.

12.34 Return types

The maintained allowed return types are:

- `Absolute`
- `Relative`
- `Log`

Absolute

Absolute return is computed as:

- `v2 - v1`

and is applied back to a base value as:

- `baseValue + returnValue`

Absolute returns are typically appropriate where the quantity is already a bounded or additive level rather than a multiplicative price-like process.

Relative

Relative return is computed as:

- `(v2 + displacement) / (v1 + displacement) - 1.0`

If `v1 + displacement` is effectively zero, the engine logs a warning and returns zero rather than throwing or dividing by zero.

Relative return is applied back as:

- `baseValue * (1.0 + returnValue)`

Log

Log return is computed as:

- `log((v2 + displacement) / (v1 + displacement))`

This requires the shifted ratio to be positive. If that condition fails, the engine logs a warning and returns zero.

Log return is applied back as:

- `baseValue * exp(returnValue)`

12.35 Displacement

`displacement` is optional and defaults to `0.0`.

It exists to stabilise relative and log returns in cases where the raw risk factor can approach zero or cross sign boundaries that would make the return undefined or numerically unstable.

Practical guidance:

- for strictly positive quantities far from zero, a zero displacement is usually acceptable,
- for quantities that may approach zero, a positive displacement can avoid division-by-zero and invalid log-ratio problems,
- specific-name overrides are the correct place to use a non-zero displacement when only one subset of a family needs it.

The current common example demonstrates exactly that pattern:

- all commodity curves use `Relative` returns by default,
- `CommodityCurve/WTI` overrides the family default with a displacement of `0.7`.

12.36 Built-in default configuration

Even with no authored file at all, the analytics runtime still has a built-in default configuration. The default constructor for `ReturnConfiguration` installs family defaults including:

- `DiscountCurve`, `YieldCurve`, `IndexCurve`, `DividendYield` and `SurvivalProbability` → `Log`
- `SwaptionVolatility`, `YieldVolatility`, `OptionletVolatility`, `FXSpot`, `FXVolatility`, `EquitySpot`, `EquityVolatility`, `CDSVolatility`, `CPIIndex`, `ZeroInflationCapFloorVolatility`, `YoYInflationCapFloorVolatility`, `CommodityCurve` and `CommodityVolatility` → `Relative`
- `RecoveryRate`, `BaseCorrelation`, `ZeroInflationCurve`, `YoYInflationCurve`, `SecuritySpread` and `Correlation` → `Absolute`

All of these built-in defaults carry zero displacement.

This matters operationally:

- a published policy is an explicit override of the runtime defaults,
- a thin authored file does not need to restate every family if only a few families or names require custom handling,
- but the maintained guide should still document the defaults so the user knows what happens when a family is omitted.

12.37 Admissibility caps and floors

After applying a return, VRE caps certain rebuilt values into admissible ranges:

- `BaseCorrelation` and `Correlation` are capped to `[-1.0, 1.0]`,
- `RecoveryRate` and `SurvivalProbability` are capped to `[0.0, 1.0]`.

This behavior is part of the runtime semantics of the return configuration and is relevant when choosing between absolute, relative and log treatments.

12.38 Compatibility XML

The publisher emits `historicalreturnconfig_compat.xml`. This file is a generated compatibility view of the native seed and resolved manifest. It is not the maintained authoring surface.

The maintained rule is therefore:

- author the YAML seed,
- publish JSON and resolved JSON,
- treat the compatibility XML only as a bridge for non-migrated readers.

12.39 Operational interpretation

The current maintained interpretation of the historical-return policy is:

- define family-level defaults by risk-factor key type,
- add specific-name overrides only where needed,
- choose from `Absolute`, `Relative` and `Log`,
- use a positive displacement only where numerical stability requires it,
- rely on the resolved JSON publication as the runtime and evidence contract.

This preserves the detailed meaning of the historical `ReturnConfiguration` chapter while documenting the current native surface and the true runtime override behavior.

13 Collateral Balances

Collateral balances are now treated as a separate row-oriented publication surface rather than as an inline branch of a netting XML tree. The maintained publication is:

- `examples/common/input/runtime_policy/published/collateral_balance.json`

A representative populated example is shown in listing 19.

Listing 19: Current collateral-balance publication

```
{
  "version": "1.0.0",
  "traceabilityRefs": {
    "sourceXml": "examples/regulatory/pillar_1/credit/input/SA-CCR/collateralbalances_outside.xml"
  },
  "rows": [
    {
      "nettingSetId": "NS1",
      "currency": "USD",
      "initialMargin": -50000.0,
      "variationMargin": 0.0,
      "allocationRule": "DIRECT",
      "outsideNettingSet": true
    }
  ]
}
```

13.1 Current contract role

The collateral-balance publication exists alongside, not inside, the resolved netting-set publication. This split is deliberate:

- netting-set and CSA structure are slow-moving agreement state,
- collateral balances are row-oriented operational state,
- the balance file is the natural place for frequently updated margin positions,
- the netting-set publication remains the source of truth for agreement and CSA definitions.

The current engine and publication tooling therefore consume:

- `resolved_netting_set_information.json` for legal agreement, CSA and independent-amount structure,
- `collateral_balance.json` for current balance rows.

This is cleaner than the XML-era embedding and more suitable for deterministic replay, evidence publication and future higher-volume balance feeds.

13.2 Top-level structure

The current publication carries:

- `version`: publication version,

- `traceabilityRefs`: provenance back to the authored source and supporting architecture / specification documents,
- `rows`: the list of collateral-balance rows.

Each row is the current native equivalent of one legacy `CollateralBalance` node.

13.3 Required row fields

- `nettingSetId`: the unique identifier of the legal netting set or balance bucket. Allowable values: any string. This is the minimum required identity field.
- `currency`: the currency in which the collateral balance amounts are denominated. Allowable values: see Table 22.

The engine will reject a row that does not provide a valid `nettingSetId` and `currency`.

13.4 Balance amount fields

The balance row can carry:

- `initialMargin` [Optional]
- `variationMargin` [Optional]

Allowable values: any numeric value.

The sign convention is the same as in the legacy XML guide:

- positive amounts indicate collateral received / held,
- negative amounts indicate collateral held by the counterparty or client, i.e. posted from the VRE reporting perspective.

In the common current examples, a negative `initialMargin` therefore means that the reporting entity has posted initial margin rather than received it.

13.5 Relationship to independent amount

The legacy chapter documented `IndependentAmountHeld` inside the balance file. In the maintained current surface that field no longer belongs in the collateral-balance row.

Current rule:

- independent amount remains part of the netting / CSA agreement state and is documented with the resolved netting-set information,
- collateral-balance rows only describe the current operational balance amounts such as initial margin and variation margin.

This is an intentional change in the current native model. The balance file is no longer used to override independent amount on a row-by-row basis.

13.6 Optional row qualifiers

The current JSON publication supports additional row qualifiers that were not spelled out in the earlier user-guide draft:

- `collateralType` [Optional]: free-form collateral type qualifier.
- `collateralHaircut` [Optional]: numeric haircut applied to the collateral.
- `outsideNettingSet` [Optional]: boolean, default `false`. If true, the balance is marked as being outside the legal netting-set perimeter.
- `allocationRule` [Optional]: allocation mode used when the balance is outside the netting set. If `outsideNettingSet = true` and no rule is given, the current engine defaults this to `DIRECT`.
- `allocationGroupId` [Optional]: identifier used to group allocated balances.
- `passThroughToCcp` [Optional]: boolean, default `false`. Indicates pass-through treatment to a CCP.
- `postedByClient` [Optional]: boolean, default `false`. Indicates client-posted collateral.
- `bankruptcyRemote` [Optional]: boolean, default `true`. Indicates whether the balance is treated as bankruptcy remote.

13.7 Extended netting-set identity

The current JSON contract can carry more than a bare `nettingSetId`. When the publication needs richer identity to stay aligned with current CCR, CVA and RWA flows, a row may additionally carry fields that map to `NettingSetDetails`, for example:

- `agreementType`
- `callType`
- `initialMarginType`
- `legalEntityId`
- `bookingEntityId`
- `nettingAgreementId`
- `marginAgreementId`
- `ccpId`
- `ccpRole`
- `ccpClearingMemberId`
- `ccpClientProtectionLevel`
- `immScopeProducts`
- `ccrMethod`
- `ccrMethodScope`

- `ccrExposureType`
- `ccpCleared`
- `qccpFlag`
- `clientClearingFlag`
- `immApproved`

In the simplest case only `nettingSetId` is needed. In the richer regulatory flows, however, these extended identity fields allow the balance publication to stay aligned with the same legal and clearing semantics used by the resolved netting-set and reporting layers.

13.8 Operational interpretation

The maintained interpretation of the balance row is therefore:

- identify the netting set or margin bucket,
- state the currency of the current collateral balance,
- provide the current initial and/or variation margin amounts,
- optionally qualify whether the balance sits outside the legal netting perimeter and how it should be allocated,
- optionally attach clearing / legal-entity qualifiers needed by the current capital and reporting workflows.

This is materially more explicit than the previous short native draft and keeps the practical meaning of the legacy XML chapter while using the current native publication split.

14 Counterparty Information

Counterparty information is no longer maintained as a single `counterparty.xml` authoring surface. The current maintained contract is a native publication pipeline that separates:

- counterparty identity and legal-entity binding in `counterparty_master.yaml`,
- regulatory and capital attributes in `counterparty_regulatory_profile.json`,
- pairwise counterparty correlation inputs in `counterparty_correlation_set.json`,
- the resolved runtime publication `published/resolved_counterparty_information.json`,
- a generated compatibility file `published/counterparty_compat.xml` for legacy readers that still expect the historical XML root.

The canonical current publication used by the runtime setup is `runtime_policy/published/resolved_counterparty_information.json`. A representative populated example is shown in listing 20.

Listing 20: Current resolved counterparty publication


```
{
  "schemaVersion": "1.0.0",
  "seedVersion": "1.0.0",
  "publishedAt": "2026-03-13T00:00:00Z",
  "contentHash": "14a7f2f635abe14dd3facc2bab5ec57862c11f3c53c8c2dbdb4f4c1cf16323f5",
  "sourceLineage": {
    "masterPath": "examples/regulatory/pillar_1/credit/input_native/vre_saccr_outside_collateral/runtime_policy/counterparty_
    "profilePath": "examples/regulatory/pillar_1/credit/input_native/vre_saccr_outside_collateral/runtime_policy/counterparty
    "correlationsPath": "examples/regulatory/pillar_1/credit/input_native/vre_saccr_outside_collateral/runtime_policy/counter
    "generator": "tools/ci/publish_counterparty_contracts.py"
  },
  "counterparties": [
    {
      "counterpartyId": "CPTY_A",
      "legalEntityId": "LE_CREDITRISK",
      "creditQuality": "IG",
      "saCvaRiskBucket": "2",
      "saCrRuleVersion": "CRE20_2023",
      "saCrRatingPolicy": "UNRATED_ONLY",
      "saCrRatingGrade": "UNRATED",
      "saCrCrmPolicyVersion": "CRE22_2023",
      "baCvaRiskWeight": 0.05,
      "saCcrRiskWeight": 1.0,
      "clearingCounterparty": false,
      "cvaExempt": false,
      "marketRiskCapitalCharge": true,
      "saCrMdbZeroRiskWeightEligible": false
    }
  ],
  "correlations": []
}
```


14.1 Current contract role

The resolved counterparty publication is the runtime join of three authored surfaces:

- `counterparty_master.yaml`: the master list of counterparty identifiers and legal-entity bindings,

- `counterparty_regulatory_profile.json`: the regulatory, capital, clearing and qualification attributes,
- `counterparty_correlation_set.json`: pairwise counterparty correlation values.

This split is deliberate. The master counterparty perimeter changes at a different rate from the regulatory profile, and both change at a different rate from pairwise correlation assumptions. The current publication model therefore keeps:

- identity and legal ownership in a small YAML master file,
- capital and reporting policy in a dedicated JSON regulatory profile,
- correlation assumptions in a separate JSON set,
- a deterministic runtime join that is versioned, hashed and traceable.

This is materially better suited to the current Basel III end-game estate than the historical XML-only surface, because the same published counterparty object is now consumed across SA-CCR, IMM, BA-CVA, SA-CVA, credit-risk capital, reporting and scope-driven runs.

14.2 Authored sources

14.2.0.1 Counterparty master. The authored master file provides the counterparty perimeter and the optional legal-entity binding. A representative authored seed is:

Listing 21: Counterparty master authoring seed

```yaml
version: 1.0.0
traceabilityRefs:
  adr: docs/architecture/adr/0001-serialization-and-storage-format-standards.md
counterparties:
- counterpartyId: CPTY_A
  legalEntityId: LE_CREDITRISK
- counterpartyId: CPTY_B
  legalEntityId: LE_CREDITRISK
```

14.2.0.2 Regulatory profile. The regulatory profile adds the risk and policy attributes that the old XML file mixed into a single `Counterparty` node:

Listing 22: Counterparty regulatory profile seed

```json
{
  "version": "1.0.0",
  "counterparties": [
    {
      "counterpartyId": "CPTY_A",
      "creditQuality": "IG",
      "baCvaRiskWeight": 0.05,
      "saCcrRiskWeight": 1.0,
      "saCvaRiskBucket": "2",
      "saCrRuleVersion": "CRE20_2023",
      "saCrRatingPolicy": "UNRATED_ONLY",
      "saCrRatingGrade": "UNRATED",
      "saCrCrmPolicyVersion": "CRE22_2023"
    }
  ]
}
```

14.2.0.3 Correlation set.

The correlation set is authored independently:

Listing 23: Counterparty correlation authoring seed

```
{
  "version": "1",
  "correlations": [
    {
      "counterparty1": "CPTY_A",
      "counterparty2": "CPTY_B",
      "correlation": 0.25
    }
  ]
}
```

The publication step is implemented in `tools/ci/publish_counterparty_contracts.py`. That publisher:

- validates the three authored inputs,
- normalises ordering and schema version metadata,
- rejects duplicate counterparty identifiers,
- rejects unknown profile / correlation references,
- rejects duplicate unordered correlation pairs,
- emits the resolved runtime JSON and the XML compatibility view.

14.3 Top-level resolved structure

The top-level resolved publication contains:

- `schemaVersion`: publication schema version. Current value: `1.0.0`.
- `seedVersion`: version inherited from the authored master seed.
- `publishedAt`: ISO-8601 publication timestamp.
- `contentHash`: deterministic content fingerprint of the resolved counterparties and correlations.
- `sourceLineage`: explicit provenance for the master, profile and correlation source files and the publishing tool.
- `counterparties`: resolved list of counterparty objects.
- `correlations`: resolved list of pairwise counterparty correlations.

The source-lineage block is operationally important. It gives Config Doctor, release evidence and maintainers a direct route back to the authored native surfaces that produced the runtime object.

14.4 Counterparty records

Each entry in `counterparties[]` is the current native equivalent of one historical `Counterparty` node. The fields are grouped naturally into identity, regulatory profile, SA-CR enrichment and CCP qualification state.

Identity and legal-entity fields

- `counterpartyId`: unique identifier for the counterparty. This field is mandatory both in the authored master and in the resolved runtime publication. Allowable values: any non-empty string.
- `legalEntityId` [Optional]: legal entity binding for the counterparty. This field lives in the master seed and is copied into the resolved publication when present. Allowable values: any string.

The publisher requires unique `counterpartyId` values in the master seed. The regulatory profile is not allowed to introduce new counterparties: every profile row must match an existing master identifier.

Core credit and clearing fields

- `clearingCounterparty` [Optional]: whether the counterparty is treated as a clearing counterparty. Allowable values in the current authored and resolved JSON surfaces: boolean only. If omitted, defaults to `false`.
- `creditQuality` [Optional]: supervisory credit-quality classification. Allowable values in the engine parser: `HY`, `IG`, `NR`. If omitted, defaults to `NR`.
- `cvaExempt` [Optional]: whether the counterparty is exempt from CVA capital treatment in the current run context. Allowable values: boolean. If omitted, defaults to `false`.
- `marketRiskCapitalCharge` [Optional]: whether the counterparty remains in-scope for market-risk capital charge treatment in downstream reporting and capital workflows. Allowable values in JSON: boolean. If omitted, defaults to `true`.

The historical XML allowed a wider range of textual boolean forms. The current native authored and resolved surfaces are stricter: these fields are booleans in JSON and are validated as such before publication and runtime load.

BA-CVA, SA-CCR and SA-CVA fields

- `baCvaRiskWeight` [Optional]: BA-CVA supervisory risk weight for the counterparty. Allowable values: numeric. If omitted, the runtime parser uses a null / not provided value. In the legacy XML parser the default was zero, so authors should populate this field explicitly when BA-CVA or SA-CVA runs depend on it.
- `saCcrRiskWeight` [Optional]: SA-CCR supervisory risk weight. Allowable values: numeric. If omitted, defaults to `1.0`.
- `saCvaRiskBucket` [Optional]: SA-CVA counterparty credit-spread bucket. Allowable values: non-empty string. Historical examples used numeric buckets `1` through `8`; current resolved examples may also use named supervisory buckets such as `SOVEREIGN` or `CORPORATES` where the broader workflow uses named taxonomies.

This is one of the important current-state extensions over the old guide: the runtime parser does not restrict `saCvaRiskBucket` to only the original integer representation. The user guide therefore documents the field as the published supervisory bucket identifier, not just as an integer slot.

SA-CR enrichment fields

The current counterparty surface also carries SA-CR specific enrichment used by the Basel III end-game credit-risk capital flows:

- `saCrRuleVersion` [Optional]: version tag for the SA-CR rule set or jurisdictional policy line applied to the counterparty.
- `saCrRatingPolicy` [Optional]: rating-policy identifier used when mapping the counterparty to SA-CR treatment.
- `saCrRatingGrade` [Optional]: resolved rating grade under the active SA-CR policy.
- `saCrCrmPolicyVersion` [Optional]: version tag for the credit-risk-mitigation policy applied to the counterparty.
- `saCrCrmSubstitutionRiskWeight` [Optional]: numeric substitution risk weight applied under SA-CR CRM rules.
- `saCrDueDiligenceFloorRiskWeight` [Optional]: numeric due-diligence floor risk weight applied under SA-CR.
- `saCrMdbZeroRiskWeightEligible` [Optional]: boolean flag for MDB zero-risk-weight eligibility. If omitted, defaults to `false`.
- `saCrOtherAssetRiskWeight` [Optional]: numeric residual or other-asset risk weight when the specific SA-CR workflow needs one.

These fields did not appear in the older user-guide chapter because they are part of the newer credit-risk capital extension of the counterparty contract. They must now be treated as first-class runtime data, not as ad hoc report-side annotations.

CCP qualification fields

The current publication also carries the CCP qualification state required for current CCR and capital treatment:

- `ccpQualificationStatus` [Optional]: current CCP qualification status. Allowable values in the engine parser: `UNSPECIFIED`, `QCCP`, `FORMER\_QCCP\_TRANSITION`, `NON\_QCCP`. If omitted, defaults to `UNSPECIFIED`.
- `ccpQualificationEffectiveFrom` [Optional]: effective-from date for the qualification status. Allowable values: ISO date string.
- `ccpQualificationEffectiveTo` [Optional]: effective-to date for the qualification status. Allowable values: ISO date string.
- `formerQccpSupervisoryOverrideTo` [Optional]: date through which the former-QCCP supervisory override applies. Allowable values: ISO date string.

This qualification block is one of the key ways in which the current guide must go beyond the old XML documentation. The field set is now used directly in regulatory analytics and reporting, and must therefore be documented with the same care as the older BA-CVA and SA-CCR fields.

14.5 Default and omission rules

The runtime JSON loader in `CounterpartyManager::fromJsonString()` uses the following defaults when optional fields are omitted:

- `clearingCounterparty` → `false`
- `creditQuality` → `NR`
- `cvaExempt` → `false`
- `saCcrRiskWeight` → `1.0`
- `saCrMdbZeroRiskWeightEligible` → `false`
- `ccpQualificationStatus` → `UNSPECIFIED`
- `marketRiskCapitalCharge` → `true`

Numeric fields such as `baCvaRiskWeight`, `saCrCrmSubstitutionRiskWeight`, `saCrDueDiligenceFloorRiskWeight` and `saCrOtherAssetRiskWeight` remain optional numerics. When omitted they stay unset rather than being forced to a synthetic value in the resolved JSON.

Operationally this means:

- authors should populate supervisory risk-weight fields explicitly when a workflow depends on them,
- omission is acceptable only when the downstream workflow can either apply a separate default or treat the value as unavailable by design,
- publication should not be used to invent supervisory values that were not authored upstream.

14.6 Correlation set

The historical XML chapter had a separate `Correlations` block. The same concept remains in the current publication, but is authored and validated as a dedicated JSON surface. Each entry in `correlations[]` has the form:

- `counterparty1`: first counterparty identifier,
- `counterparty2`: second counterparty identifier,
- `correlation`: numeric pairwise correlation.

The publisher and runtime loader both enforce important integrity rules:

- both counterparties must exist in the authored master perimeter,
- the correlation list must be an array of objects,
- unordered pairs must be unique, so `(A,B)` and `(B,A)` cannot both appear,
- the correlation value must be numeric.

The correlation set therefore remains conceptually identical to the historical `<Correlation cpty1="..."cpty2="...">...` entries, but it is now a first-class authored

contract that can be validated independently and joined deterministically into the runtime publication.

14.7 Compatibility XML

The current maintained guide is native-first, but the counterparty publisher also emits `published/counterparty_compat.xml`. This file is not the source of truth. It is a generated compatibility view for readers that still expect the historical `CounterpartyInformation` XML root.

The compatibility publication preserves the established XML tags:

- `CounterpartyId`
- `LegalEntityId`
- `CreditQuality`
- `SaCvaRiskBucket`
- `SaCrRuleVersion`, `SaCrRatingPolicy`, `SaCrRatingGrade`, `SaCrCrmPolicyVersion`
- `CcpQualificationStatus`, `CcpQualificationEffectiveFrom`, `CcpQualificationEffectiveTo`, `FormerQccpSupervisoryOverrideTo`
- `BaCvaRiskWeight`, `SaCcrRiskWeight`, `SaCrCrmSubstitutionRiskWeight`, `SaCrDueDiligenceFloorRiskWeight`, `SaCrOtherAssetRiskWeight`
- `ClearingCounterparty`, `CvaExempt`, `SaCrMdbZeroRiskWeightEligible`, `MarketRiskCapitalCharge`
- `Correlations/Correlation`

The important rule is that this XML is now downstream of the native sources. It should not be edited by hand and it should not be treated as the normative parameterisation surface for current work.

14.8 Operational interpretation

The maintained interpretation of the current counterparty publication is:

- define the counterparty perimeter and legal-entity binding once in the master seed,
- attach regulatory, clearing, qualification and capital attributes in the regulatory profile,
- express pairwise credit-factor correlation assumptions in the correlation set,
- publish a deterministic resolved runtime contract for use in configuration, evidence and release workflows,
- emit XML compatibility only as a generated bridge for remaining legacy consumers.

This preserves the practical detail of the historical XML chapter while documenting the materially richer current-native surface used by the engine.

14.9 Allowable Values

Date	
Date Fields	**Allowable Values**
All Date fields: StartDate EndDate Date ExerciseDate PayDate ValueDate NearDate FarDate etc	Any of the following date formats are supported: *yyyymmdd* *yyyy-mm-dd* *yyyy/mm/dd* *yyyy.mm.dd* *dd-mm-yy* *dd/mm/yy* *dd.mm.yy* *dd-mm-yyyy* *dd/mm/yyyy* *dd.mm.yyyy* and Dates as serial numbers, comparable to Microsoft Excel dates, with a minimum of 367 for Jan 1, 1901, and a maximum of 109574 for Dec 31, 2199.

Table 20: Allowable Values for Date

Convention	
Roll Convention Fields	**Allowable Values**
All Convention fields: Convention TermConvention PaymentConvention etc	*F, Following, FOLLOWING* *MF, ModifiedFollowing, Modified Following, MODIFIEDF* *P, Preceding, PRECEDING* *MP, ModifiedPreceding, Modified Preceding, MODIFIEDP* *U, Unadjusted, INDIFF* *HMMF, HalfMonthModifiedFollowing, HalfMonthMF, Half Month Modified* *NEAREST* (takes future date in case of equal distance)

Table 21: Allowable Values for Roll Conventions

`Currency`	
Category	**Allowable Values**
Fiat Currencies	*AED,AFN,ALL,AMD,ANG,AOA,ARS,AUD,AWG,AZN,* *BAM,BBD,BDT,BGN,BHD,BIF,BMD,BND,BOB,BOV,* *BRL,BSD,BTN,BWP,BYN,BZD,CAD,CDF,CHE,CHF,* *CHW,CLF,CLP,CNH,CNT,CNY,COP,COU,CRC,CUC,* *CUP,CVE,CZK,DJF,DKK,DOP,DZD,EGP,ERN,ETB,* *EUR,FJD,FKP,GBP,GEL,GGP,GHS,GIP,GMD,GNF,* *GTQ,GYD,HKD,HNL,HRK,HTG,HUF,IDR,ILS,IMP,* *INR,IQD,IRR,ISK,JEP,JMD,JOD,JPY,KES,KGS,* *KHR,KID,KMF,KPW,KRW,KWD,KYD,KZT,LAK,LBP,* *LKR,LRD,LSL,LYD,MAD,MDL,MGA,MKD,MMK,MNT,* *MOP,MRU,MUR,MVR,MWK,MXN,MXV,MYR,MZN,NAD,* *NGN,NIO,NOK,NPR,NZD,OMR,PAB,PEN,PGK,PHP,* *PKR,PLN,PYG,QAR,RON,RSD,RUB,RWF,SAR,SBD,* *SCR,SDG,SEK,SGD,SHP,SLL,SOS,SRD,SSP,STN,* *SVC,SYP,SZL,THB,TJS,TMT,TND,TOP,TRY,TTD,* *TWD,TZS,UAH,UGX,USD,USN,UYI,UYU,UYW,UZS,* *VES,VND,VUV,WST,XAF,XAU,XCD,XOF, XPF,XSU,XUA,YER,ZAR,ZMW,ZWL*
Minor Currencies	*GBp, GBX* (for pennies of GBP) *ILa, ILX, ILs, ILA* (for agorot of ILS) *ZAc, ZAC, ZAX* (for cents of ZAR) Note: Minor Currency codes are only supported for equity products.
Precious Metals treated as Currencies	*XAG, XAU, XPD, XPT*
Cryptocurrencies	*BTC, XBT, ETH, ETC, BCH, XRP, LTC*
This full list of currencies is available via loading the provided `currencies.xml` at start-up. Note: Currency codes must also match available currencies in the `simulation.xml` file.	

Table 22: Allowable Values for Currency

`Rule`	
Allowable Values	**Effect**
Backward	Backward from termination date to effective date.
Forward	Forward from effective date to termination date.
Zero	No intermediate dates between effective date and termination date.
ThirdWednesday	All dates but effective date and termination date are taken to be on the third Wednesday of their month (with forward calculation.)
LastWednesday	All dates but effective date and termination date are taken to be on the last Wednesday of their month (with forward calculation.)
ThirdThursday	All dates but effective date and termination date are taken to be on the third Thursday of their month (with forward calculation.)
ThirdFriday	All dates but effective date and termination date are taken to be on the third Friday of their month (with forward calculation.)
MondayAfterThird-Friday	All dates but effective date and termination date are taken to be on the Monday following the third Friday of their month (with forward calculation.)

TuesdayAfterThird-Friday	All dates but effective date and termination date are taken to be on the Tuesday following the third Friday of their month (with forward calculation.)
Twentieth	All dates but the effective date are taken to be the twentieth of their month (used for CDS schedules in emerging markets.) The termination date is also modified.
TwentiethIMM	All dates but the effective date are taken to be the twentieth of an IMM month (used for CDS schedules.) The termination date is also modified.
OldCDS	Same as TwentiethIMM with unrestricted date ends and long/short stub coupon period (old CDS convention).
CDS	Credit derivatives standard rule defined in 'Big Bang' changes in 2009. For quarterly periods (`Tenor` set to *3M*): (Assuming no `FirstDate`/`LastDate`) Dates fall on 20th of March, June, September, December. A *Following* roll convention will be applied if the 20th falls on a non-business day. If the `EndDate` in the schedule is set to a date beyond the rolled quarterly CDS date, the actual trade termination date will be on the following quarterly CDS date. The first coupon will be paid on the quarterly CDS date following the `StartDate`, and be for the period since the previous quarterly CDS date. For monthly periods (`Tenor` set to *1M*): (Assuming no `FirstDate`/`LastDate`) Dates fall on 20th of each month, but the termination is still adjusted to be in line with quarterly periods. If the `EndDate` in the schedule is set to a date beyond the rolled quarterly CDS date (i.e. the 20th+roll Mar, Jun, Sep, Dec), the actual termination date will be on the following quarterly CDS date, causing a long final stub. The first coupon will be paid on the next 20th monthly following the `StartDate`, and be for the period since the previous month's 20th.
CDS2015	Credit derivatives standard rule updated in 2015. Same as *CDS* but with termination dates adjusted to 20th June and 20th December. For schedule `EndDates` from the 20th of March to the 19th September, both included, the termination date will fall on the 20th June (with *Following* roll). For schedule `EndDates` from the 20th September to the 19th March, both included, the termination date will fall on the 20th December (with *Following* roll).
EveryThursday	If FirstDate is not given, all thursdays between start and end date. If FirstDate is given, FirstDate plus all thursdays between FirstDate and end date.

Table 23: Allowable Values for Rule

Calendar	
Allowable Values	**Resulting Calendar**
TARGET, TGT, EUR	Target Calendar
CA, CAN, CAD, TRB	Canada Calendar
JP, JPN, JPY, TKB	Japan Calendar
CH, CHE, CHF, ZUB	Switzerland Calendar
GB, GBR, GBP, LNB, UK	UK Calendar
US, USA, USD, NYB	US Calendar
US-SET	US Settlement Calendar
US-GOV	US Government Bond Calendar
US-NYSE, New York stock exchange	US NYSE Calendar
US with Libor impact	US Calendar for Libor fixings
US-NERC	US NERC Calendar
US-SOFR	US SOFR fixing Calendar
AR, ARG, ARS	Argentina Calendar
AU, AUD, AUS	Australia Calendar
AT, AUT, ATS	Austria Calendar
BE, BEL, BEF	Belgium Calendar
BW, BWA, BWP	Botswana Calendar
BR, BRA, BRL	Brazil Calendar
CL, CHL, CLP	Chile Calendar
CN, CHN, CNH, CNY	China Calendar
CO, COL, COP	Colombia Calendar
CY, CYP	Cyprus Calendar
CZ, CZE, CZK	Czech Republic Calendar
DK, DNK, DKK, DEN	Denmark Calendar
FI, FIN	Finland Calendar
FR, FRF	France Calendar
DE, DEU	Germany Calendar
GR, GRC	Greek Calendar
HK, HKG, HKD	Hong Kong Calendar
HU, HUN, HUF	Hungary Calendar
IS, ISL, ISK	Iceland Calendar
IN, IND, INR	India Calendar
ID, IDN, IDR	Indonesia Calendar
IE, IRL	Ireland Calendar
IL, ISR, ILS	Israel Calendar
Telbor	Tel Aviv Inter-Bank Offered Rate Calendar
IT, ITA, ITL	Italy Calendar
LU, LUX, LUF	Luxembourg Calendar
MX, MEX, MXN	Mexico Calendar
MY, MYS, MYR	Malaysia Calendar
NL, NLD, NZD	New Zealand Calendar
NO, NOR, NOK	Norway Calendar
PE, PER, PEN	Peru Calendar

PH, PHL, PHP	Philippines Calendar
PO, POL, PLN	Poland Calendar
RO, ROU, RON	Romania Calendar
RU, RUS, RUB	Russia Calendar
SAU, SAR	Saudi Arabia
AE, ARE, AED	United Arab Emirates
SG, SGP, SGD	Singapore Calendar
ZA, ZAF, ZAR, SA	South Africa Calendar
KR, KOR, KRW	South Korea Calendar
ES, ESP	Spain Calendar
SE, SWE, SEK, SS	Sweden Calendar
TW, TWN, TWD	Taiwan Calendar
TH, THA, THB	Thailand Calendar
TR, TUR, TRY	Turkey Calendar
UA, UKR, UAH	Ukraine Calendar
XASX	Australian Securities Exchange Calendar
BVMF	Brazil Bovespa Calendar
XTSE	Canada Toronto Stock Exchange Calendar
XSHG	China Shanghai Stock Exchange Calendar
XFRA	Germany Frankfurt Stock Exchange
XETR	Germany XETRA Calendar
ECAG	Germany EUREX Calendar
EUWA	Germany EUWAX Calendar
XJKT	Indonesia Jakarta Stock Exchange (now IDX) Calendar
XIDX	Indonesia Indonesia Stock Exchange Calendar
XDUB	Ireland Stock Exchange Calendar
XTAE	Israel Tel Aviv Stock Exchange Calendar
XMIL	Italy Italian Stock Exchange Calendar
MISX	Russia Moscow Exchange Calendar
XKRX	Korea Exchange Calendar
XSWX	Switzerland SIX Swiss Exchange Calendar
XLON	UK London Stock Exchange
XLME	UK London Metal Exchange
XNYS	US New York Stock Exchange Calendar
XPAR	Paris stock exchange
WMR	Thomson Reuters WM/Reuters Spot
IslamicWeekendsOnly	Islamic Weekends Only Calendar
WeekendsOnly	Weekends Only Calendar
IslamicWeekendsOnly	Islamic Weekends Only Calendar
ICE_FuturesUS	ICE Futures U.S. Currency, Stock and Credit Index, Metal, Nat Gas, Power, Oil and Environmental
ICE_FuturesUS_1	ICE Futures U.S. Sugar, Cocoa, Coffee, Cotton and FCOJ
ICE_FuturesUS_2	ICE Futures U.S. Canola
ICE_FuturesEU	ICE Futures Europe
ICE_FuturesEU_1	ICE Futures Europe for contracts where 26 Dec is a holiday
ICE_EndexEnergy	ICE Endex European power and natural gas products

ICE_EndexEquities	ICE Endex European equities
ICE_SwapTradeUS	ICE Swap Trade U.S.
ICE_SwapTradeUK	ICE Swap Trade U.K.
ICE_FuturesSingapore	ICE futures Singapore
CME	CME group exchange calendar
NullCalendar, Null	Null Calendar, i.e. all days are business days

Table 24: Allowable Values for Calendar. Combinations of calendars can be provided using comma separated calendar names.

DayCount Convention	
Allowable Values	**Resulting DayCount Convention**
A360, Actual/360, ACT/360, Act/360	Actual 360
A365, A365F, Actual/365 (Fixed), Actual/365 (fixed), ACT/365.FIXED, ACT/365, ACT/365L, Act/365, Act/365L	Actual 365 Fixed
A364, Actual/364, Act/364, ACT/364	Actual 364
Actual/365 (No Leap), Act/365 (NL), NL/365, Actual/365 (JGB)	Actual 365 Fixed (No Leap Year)
Act/365 (Canadian Bond)	Actual 365 Fixed (Canadian Bond)
T360, 30/360, ACT/nACT, 30/360 US, 30/360 (US), 30U/360, 30US/360	Thirty 360 (US)
30/360 (Bond Basis)	Thirty 360 (Bond Basis)
30E/360 (Eurobond Basis), 30E/360, 30/360 AIBD (Euro), 30E/360.ICMA, 30E/360 ICMA	Thirty 360 (European)
30E/360E, 30E/360 ISDA, 30E/360.ISDA, 30/360 German, 30/360 (German)	Thirty 360 (German)
30/360 Italian, 30/360 (Italian)	Thirty 360 (Italian)
ActActISDA, ACT/ACT.ISDA, Actual/Actual (ISDA), ActualActual (ISDA), ACT/ACT, Act/Act, ACT	Actual Actual (ISDA)
ActActISMA, Actual/Actual (ISMA), ActualActual (ISMA), ACT/ACT.ISMA	Actual Actual (ISMA)
ActActICMA, Actual/Actual (ICMA), ActualActual (ICMA), ACT/ACT.ICMA	Actual Actual (ICMA)
ActActAFB, ACT/ACT.AFB, Actual/Actual (AFB), ACT29	Actual Actual (AFB)
BUS/252, Business/252	Brazilian Bus/252
1/1	1/1
Simple	Simple Day Counter
Year	Year Counter

Table 25: Allowable Values for DayCount Convention

Index	
On form CCY-INDEX-TENOR, and matching available indices in the market data configuration.	
Index Component	**Allowable Values**
CCY-INDEX	*EUR-EONIA* *EUR-ESTER, EUR-ESTR, EUR-STR* *EUR-EURIBOR, EUR-EURIBOR365* *EUR-LIBOR* *EUR-CMS* *USD-FedFunds* *USD-SOFR* *USD-Prime* *USD-LIBOR* *USD-SIFMA* *USD-CMS* *GBP-SONIA* *GBP-LIBOR* *GBP-CMS* *GBP-BoEBase* *JPY-LIBOR* *JPY-TIBOR* *JPY-EYTIBOR* *JPY-TONAR* *JPY-CMS* *CHF-LIBOR* *CHF-SARON* *AUD-LIBOR* *AUD-BBSW* *CAD-CDOR* *CAD-BA* *SEK-STIBOR* *SEK-LIBOR* *SEK-STINA* *DKK-LIBOR* *DKK-CIBOR* *DKK-CITA* *SGD-SIBOR* *SGD-SOR* *HKD-HIBOR* *HKD-HONIA* *NOK-NIBOR* *HUF-BUBOR* *IDR-IDRFIX* *INR-MIFOR* *MXN-TIIE* *PLN-WIBOR* *RUB-MOSPRIME* *SKK-BRIBOR* *THB-THBFIX* *THB-THOR* *THB-BIBOR* *NZD-BKBM*
TENOR	An integer followed by *D, W, M or Y*

Defaults for FixingDays	
Index	**Default value**
Ibor indices	2, except for the Ibor indices below:
USD-SIFMA	1
GBP-LIBOR	0
AUD-BBSW	0
CAD-CDOR	0
CNY-SHIBOR	1
HKD-HIBOR	0
MXN-TIIE	1
MYR-KLIBOR	0
TRY-TRLIBOR	0
ZAR-JIBAR	0
Overnight indices	0, except for the Overnight indices below:
CHF-TOIS	1
CLP-CAMARA	2
PLN-POLONIA	1
DKK-DKKOIS	1
SEK-SIOR	1

Table 27: Defaults for FixingDays

FX Index	
Index Format	**Allowable Values**
FX-SOURCE-CCY1-CCY2	The FX- part of the string stays constant for all currency pairs. SOURCE is the market data fixing source defined in the market configuration. CCY1 and CCY2 are the ISO currency codes of the fx pair. Fixings are expressed as amount in CCY2 for one unit of CCY1.

Table 28: Allowable values for FX index fixings.

Inflation CPI Index	
Trade Data	**Allowable Values**
Index for CPI leg	Any string (provided it is the ID of an inflation index in the market configuration)

Table 29: Allowable values for CPI index.

Credit CreditCurveId	
Trade Data	**Allowable Values**
CreditCurveId for credit trades - single name and index	Any string (provided it is the ID of a single name or index reference entity in the market configuration). Typically a RED-code with the *RED:* prefix Examples: *RED:2I65BRHH6* (CDX N.A. High Yield, Series 13, Version 1) *RED:008CA0|SNRFOR|USD|MR14* (Agilent Tech Senior USD)

Table 30: Allowable values for credit CreditCurveId

Equity Name	
Trade Data	**Allowable Values**
Name for equity trades	Any string (provided it is the ID of an equity in the market configuration). Typically a RIC-code with the *RIC:* prefix Examples: *RIC:.SPX* (S&P 500 Index) *RIC:EEM.N* (iShares MSCI Emerging Markets ETF)

Table 31: Allowable values for equity Name.

Commodity Curve Name	
Trade Data	**Allowable Values**
Name for commodity trades	Any string (provided it is the ID of an commodity in the market configuration)

Table 32: Allowable values for commodity data.

Tier	
Value	**Description**
SNRFOR	Senior unsecured for corporates or foreign debt for sovereigns
SUBLT2	Subordinated or lower Tier 2 debt for banks
SNRLAC	Senior loss absorbing capacity
SECDOM	Secured for corporates or domestic debt for sovereigns
JRSUBUT2	Junior subordinated or upper Tier 2 debt for banks
PREFT1	Preference shares or Tier 1 capital for banks
LIEN1	First lien
LIEN2	Second lien
LIEN3	Third lien

Table 33: Allowable values for Tier

DocClause	
Value	**Description**
CR	Full or old restructuring referencing the 2003 ISDA Definitions
MM	Modified modified restructuring referencing the 2003 ISDA Definitions
MR	Modified restructuring referencing the 2003 ISDA Definitions
XR	No restructuring referencing the 2003 ISDA Definitions
CR14	Full or old restructuring referencing the 2014 ISDA Definitions
MM14	Modified modified restructuring referencing the 2014 ISDA Definitions
MR14	Modified restructuring referencing the 2014 ISDA Definitions
XR14	No restructuring referencing the 2014 ISDA Definitions

Table 34: Allowable values for DocClause

Exchange	
Trade Data	**Allowable Values**
Exchange	Any string, typically a MIC code (provided it is the ID of an exchange in the market configuration)

Table 35: Allowable Values for Exchange

Boolean nodes	
Node Value	**Evaluates To**
Y, YES, TRUE, true, 1	true
N, NO, FALSE, false, 0	false

Table 36: Allowable values for boolean node

15 Netting Set Information

The maintained netting-set surface is no longer a hand-authored `netting.xml` file. The current engine reads a resolved machine publication that joins the business concepts which used to be split across netting, CSA, collateral-balance and regulatory-profile XML files:

- the legal and operational identity of the netting agreement,
- the collateral agreement and CSA terms,
- regulatory and clearing attributes needed by SA-CCR, IMM and CVA capital,
- current collateral balances used by exposure and capital analytics.

The canonical runtime contract is the published JSON bundle `resolved_netting_set_information.json`. A representative record is shown in listing 24.

Listing 24: Current resolved netting-set publication

```
{
  "schemaVersion": "1.0.0",
  "publishedAt": "2026-03-13T00:00:00Z",
  "sourceLineage": {
    "nettingAgreementMaster": "examples/common/input/runtime_policy/netting_agreement_master.json",
    "csaTerms": "examples/common/input/runtime_policy/csa_terms.json",
    "clearingAndRegulatoryProfile": "examples/common/input/runtime_policy/clearing_and_regulatory_profile.json",
    "collateralBalance": "examples/common/input/runtime_policy/collateral_balance.json"
  },
  "nettingSets": [
    {
      "nettingSetId": "NS_AU_BANK_CPTY_A",
      "agreementType": "ISDA",
      "callType": "BILATERAL",
      "initialMarginType": "Bilateral",
      "legalEntityId": "LE_AU_BANK",
      "bookingEntityId": "LE_AU_BANK",
      "nettingAgreementId": "NA_AU_BANK_CPTY_A",
      "marginAgreementId": "MA_AU_BANK_CPTY_A",
      "riskWeight": 0.2,
      "ccpCleared": true,
      "ccpId": "LCH",
      "qccpFlag": true,
      "ccpRole": "CLIENT",
      "clientClearingFlag": true,
      "immApproved": true,
      "ccrMethod": "IMM",
      "ccrMethodScope": "NETTING_SET",
      "ccrExposureType": "TRADE",
      "activeCsa": true,
      "csaTerms": {
        "bilateralType": "Bilateral",
        "csaCurrency": "USD",
        "index": "SOFR",
        "marginPeriodOfRisk": "2W",
        "thresholdPay": 1000000.0,
        "thresholdReceive": 500000.0,
        "minimumTransferAmountPay": 10000.0,
        "minimumTransferAmountReceive": 10000.0,
        "applyInitialMargin": true,
        "calculateImAmount": true,
        "calculateVmAmount": true
      }
    }
  ]
}
```

15.1 Top-level publication structure

The top-level envelope contains:

- `schemaVersion`: the contract version understood by validators and runtime loaders;
- `publishedAt`: the publication timestamp for provenance and audit;
- `contentHash`: a deterministic content fingerprint for evidence and release control;
- `sourceLineage`: the source files used to build the publication;
- `nettingSets`: the resolved netting-set records consumed by the engine.

The important operational point is that the engine should be configured against the published resolved contract, not against the authoring fragments that were used to generate it.

15.2 Uncollateralised netting sets

An uncollateralised netting set is represented by a netting-set record with `activeCsa = false`. The minimum current shape is:

Listing 25: Uncollateralised netting-set record

```
{
  "nettingSetId": "NS_AU_BANK_CPTY_B",
  "agreementType": "ISDA",
  "callType": "BILATERAL",
  "legalEntityId": "LE_AU_BANK",
  "bookingEntityId": "LE_AU_BANK",
  "nettingAgreementId": "NA_AU_BANK_CPTY_B",
  "marginAgreementId": "MA_AU_BANK_CPTY_B",
  "riskWeight": 0.5,
  "ccrMethod": "SACCR",
  "ccrMethodScope": "NETTING_SET",
  "ccrExposureType": "TRADE",
  "activeCsa": false
}
```

The core record fields are:

- `nettingSetId`: unique identifier for the netting set.
- `agreementType`: agreement family, typically *ISDA* in the current examples.
- `callType`: call-direction semantics for collateral and margin logic. Even when `activeCsa` is false, the field remains part of the agreement identity.
- `legalEntityId`: the legal entity to which the netting set belongs. This must agree with the legal-entity and scope/reporting contracts used by regulatory analytics.
- `bookingEntityId`: the booking entity used by reporting and lineage joins. In many examples it matches `legalEntityId`, but it is documented separately because the distinction matters operationally.
- `nettingAgreementId`: stable identifier for the legal agreement.

- `marginAgreementId`: identifier for the linked margin agreement if one exists in the source estate.
- `riskWeight`: directly published risk-weight metadata when the regulatory method requires the netting-set level override.
- `ccrMethod`, `ccrMethodScope`, `ccrExposureType`: regulatory execution metadata used by SA-CCR and IMM workflows.
- `activeCsa`: boolean switch that determines whether the `csaTerms` block is expected and whether collateral logic should be applied.

15.3 Collateralised netting sets

A collateralised netting set keeps the same agreement identity and augments it with a `csaTerms` payload. In addition, current regulatory examples often carry clearing and IMM metadata at the same level:

- `initialMarginType`: initial-margin directionality used by dynamic IM and SA-CCR calculations.
- `ccpCleared`, `ccpId`, `qccpFlag`, `ccpRole`, `clientClearingFlag`, `ccpClearingMemberId`, `ccpClientProtectionLevel`: clearing and CCP classification metadata needed for Basel III end-game counterparty capital treatment.
- `immApproved`, `immScopeProducts`: IMM approval and scope metadata used when IMM and SA-CCR coexist in the same estate.
- `tradeCountOverride`: explicit trade-count override where the run should not infer the count from the trade population.
- `outsideNettingSetGroupId`, `outsideNettingSetAllocationWeight`: explicit linkage for collateral amounts or regulatory add-ons that sit outside the ordinary netting-set grouping.

CSA terms

The `csaTerms` block contains the collateral agreement details used by exposure, XVA, SA-CCR, IMM and CVA calculations. The current resolved fields are:

- `bilateralType`: collateral call direction. The current examples use *Bilateral*; the same business meaning as the historical bilateral / call-only / post-only distinction is retained.
- `csaCurrency`: the collateral agreement currency.
- `index`: the overnight or collateral remuneration index applied to cash collateral balances.
- `marginPeriodOfRisk`: MPOR assumption used in exposure, margining and capital calculations.
- `initialMarginType`: initial-margin directionality, typically aligned with bilateral / call-only / post-only semantics.

- `nonExemptImRegulations`: regulatory IM regime carried with the agreement when the run needs regime-aware IM behaviour.
- `thresholdPay`, `thresholdReceive`: collateral call thresholds in the CSA currency.
- `minimumTransferAmountPay`, `minimumTransferAmountReceive`: minimum transfer amounts for pay and receive directions.
- `collateralCompoundingSpreadReceive`, `collateralCompoundingSpreadPay`: remuneration spreads applied to posted and received collateral.
- `applyInitialMargin`: whether IM should be applied in addition to variation margin.
- `calculateImAmount`: whether the run should calculate IM amounts from the configured IM methodology rather than relying only on supplied balances.
- `calculateVmAmount`: whether VM should be derived from the agreement and NPV state.
- `hasDisputes`, `disputeDurationExceedsMpor`: operational dispute flags that can affect regulatory MPOR treatment.
- `ratingTriggerClause`, `illiquidCollateral`, `concentrationRisk`, `benchmarkTransitionIlliquidity`, `benchmarkTransitionIlliquidityEndDate`: additional regulatory and operational attributes required by current Basel end-game workflows.
- `remarginFrequencyDays`: numeric remargin cadence.
- `disputeCountLast2Q`: recent dispute count for regulatory tests that depend on collateral process quality.
- `marginingFrequency.call`, `marginingFrequency.post`: explicit call and post frequencies.
- `independentAmount.held`: the independent amount carried under the agreement. Negative values continue to mean that the counterparty holds the amount.
- `independentAmount.type`: independent-amount interpretation. The current examples use *FIXED*.
- `eligibleCollaterals.currencies`: eligible collateral currencies where cash collateral is permitted.

15.4 Collateral-balance linkage

The netting-set publication is intentionally paired with `collateral_balance.json`. A representative balance row is:

Listing 26: Collateral-balance row linked to a netting set

```
{
  "nettingSetId": "NS_AU_BANK_CPTY_A",
  "currency": "USD",
  "collateralType": "CASH",
  "initialMargin": 1250000.0,
  "variationMargin": 400000.0,
  "collateralHaircut": 0.02,
  "outsideNettingSet": true,
  "passThroughToCcp": true,
  "postedByClient": true,
  "bankruptcyRemote": false,
  "allocationRule": "DIRECT",
  "allocationGroupId": "ALLOCATION_A"
}
```

The balance file deliberately holds stateful amounts separately from the legal agreement definition:

- `nettingSetId` links the balance row to the resolved netting-set record.
- `currency` and `collateralType` identify the balance instrument.
- `initialMargin` and `variationMargin` provide current collateral state.
- `collateralHaircut` captures valuation or regulatory haircut.
- `outsideNettingSet`, `passThroughToCcp`, `postedByClient`, `bankruptcyRemote`, `allocationRule`, `allocationGroupId` document how the balance should be interpreted in SA-CCR, IMM and CCP-clearing workflows.

15.5 Current operational rule

Current runs should read the three linked published contracts together:

- `resolved_netting_set_information.json` for agreement and CSA structure,
- `collateral_balance.json` for current margin state,
- `resolved_counterparty_information.json` for counterparty and regulatory classification.

This split is deliberate. It keeps legal agreement structure, current balance state, and counterparty classification independently maintainable while still allowing deterministic publication into a single runtime-ready input estate.

16 Market Data

In this section we discuss the market data, which enters into the calibration of OREs risk factor evolution models. Market data in the `market.txt` file is given in three columns; Date, Quote and Quote value.

- **Date**: The as of date of the market quote value.

  Allowable values: See `Date` in Table 20.

- **Quote**: A generic description that contains Instrument Type and Quote Type, followed by instrument specific descriptions (see 16.1 ff.). The base of a quote consists of InstType/QuoteType followed by instrument specific information separated by slashes "/".

  Allowable values for Instrument Types and Quote Types are given in Table 37.

- **Quote Value**: The market quote value in decimal form for the given quote on the given as of date. Quote values are assumed to be mid-market.

  Allowable values: Any real number.

Market Data Parameter	Allowable Values
Instrument Type	*ZERO, DISCOUNT, MM, MM_FUTURE, FRA, IMM_FRA, IR_SWAP, BASIS_SWAP, CC_BASIS_SWAP, CDS, CDS_INDEX, FX_SPOT, FX_FWD, SWAPTION, CAPFLOOR, FX_OPTION, HAZARD_RATE, RECOVERY_RATE, ZC_INFLATIONSWAP, YY_INFLATIONSWAP, ZC_INFLATIONCAPFLOOR, SEASONALITY, EQUITY_SPOT, EQUITY_FWD, EQUITY_DIVIDEND, EQUITY_OPTION, BOND, INDEX_CDS_OPTION, CPR, COMMODITY, COMMODITY_FWD, COMMODITY_OPTION*
Quote Type	*BASIS_SPREAD, CREDIT_SPREAD, YIELD_SPREAD, HAZARD_RATE, RATE, RATIO, PRICE, RATE_LNVOL, RATE_NVOL, RATE_SLNVOL, BASE_CORRELATION, SHIFT*

Table 37: Allowable values for Instrument and Quote type market data.

An excerpt from a typical `market.txt` file is shown in Listing 27.

Listing 27: Excerpt of a market data file

```
2011-01-31 MM/RATE/EUR/0D/1D 0.013750
2011-01-31 MM/RATE/EUR/1D/1D 0.010500
2011-01-31 MM/RATE/EUR/2D/1D 0.010500
2011-01-31 MM/RATE/EUR/2D/1W 0.009500
2011-01-31 MM/RATE/EUR/2D/1M 0.008700
2011-01-31 MM/RATE/EUR/2D/2M 0.009100
2011-01-31 MM/RATE/EUR/2D/3M 0.010200
2011-01-31 MM/RATE/EUR/2D/4M 0.011000

2011-01-31 FRA/RATE/EUR/3M/3M 0.013080
2011-01-31 FRA/RATE/EUR/4M/3M 0.013890
2011-01-31 FRA/RATE/EUR/5M/3M 0.014630
2011-01-31 FRA/RATE/EUR/6M/3M 0.015230

2011-01-31 IR_SWAP/RATE/EUR/2D/3M/1Y 0.014400
2011-01-31 IR_SWAP/RATE/EUR/2D/3M/1Y3M 0.015400
2011-01-31 IR_SWAP/RATE/EUR/2D/3M/1Y6M 0.016500
2011-01-31 IR_SWAP/RATE/EUR/2D/3M/2Y 0.018675
2011-01-31 IR_SWAP/RATE/EUR/2D/3M/3Y 0.022030
2011-01-31 IR_SWAP/RATE/EUR/2D/3M/4Y 0.024670
2011-01-31 IR_SWAP/RATE/EUR/2D/3M/5Y 0.026870
2011-01-31 IR_SWAP/RATE/EUR/2D/3M/6Y 0.028700
2011-01-31 IR_SWAP/RATE/EUR/2D/3M/7Y 0.030125
2011-01-31 IR_SWAP/RATE/EUR/2D/3M/8Y 0.031340
2011-01-31 IR_SWAP/RATE/EUR/2D/3M/9Y 0.032450
```

16.1 Zero Rate

The instrument specific information to be captured for quotes representing Zero Rates is shown in Table 38.

Property	**Allowable values**	**Description**
Instrument Type	*ZERO*	
Quote Type	*RATE, YIELD_SPREAD*	
Currency	See `Currency` in Table 20	Currency of the Zero rate
CurveId	A CCY concatenated with a Tenor. Should match CurveIds in the `yield-curves.xml` file	Unique identifier for the yield curve associated with the zero quote
DayCounter	See `DayCount Convention` in Table 25	The day count basis associated with the zero quote
Tenor or ZeroDate	Tenor: An integer followed by D, W, M or Y, ZeroDate: See `Date` in Table 20	Either a Tenor for tenor based zero quotes, or an explicit maturity date (ZeroDate)

Table 38: Zero Rate

Examples with a Tenor and with a ZeroDate:

- ZERO/RATE/USD/USD6M/A365F/6M
- ZERO/RATE/USD/USD6M/A365F/12-05-2018

16.2 Discount Factor

The instrument specific information to be captured for quotes representing Discount Factors is shown in Table 39.

Property	Allowable values	Description
Instrument Type	*DISCOUNT*	
Quote Type	*RATE*	
Currency	See `Currency` in Table 20	Currency of the Discount rate
CurveId	A CCY concatenated with a Tenor. Should match CurveIds in the `yield-curves.xml` file	Unique identifier for the yield curve associated with the discount quote
Term or DiscountDate	Term: An integer followed by D, W, M or Y, DiscountDate: See `Date` in Table 20	Either a Term is used to determine the maturity date, or an explicit maturity date (Discount Date) is given.

Table 39: Discount Rate

If a Term is given in the last element of the quote, it is converted to a maturity date using the calendar, specified in the conventions. Bear in mind, only zero conventions from section 12.30 can be used for the discount factor instruments.

Examples with a Term and with a DiscountDate:

- DISCOUNT/RATE/EUR/EUR3M/3Y
- DISCOUNT/RATE/EUR/EUR3M/12-05-2018

16.3 FX Spot Rate

Property	Allowable values	Description
Instrument Type	*FX*	
Quote Type	*RATE*	
Unit currency	See `Currency` in Table 20	Unit/Source currency
Target currency	See `Currency` in Table 20	Target currency

Table 40: FX Spot Rate

Example:

- FX/RATE/EUR/USD

16.4 FX Forward Rate

An FX Forward quote is expected in either a "forward points" quotation, or an "outright" quotation.

The forward points convention is given by:

$$\text{Forward Points} = \frac{\text{FX Forward} - \text{FX Spot}}{\text{Conversion Factor}}$$

with conversion factor set to 1.

Property	**Allowable values**	**Description**
Instrument Type	*FX_FWD*	
Quote Type	*RATE*	
Unit currency	See `Currency` in Table 20	Unit/Source currency
Target currency	See `Currency` in Table 20	Target currency
Term	An integer followed by D, W, M or Y.	Period from today to maturity. Alternatively, the maturity date.

Table 41: FX Forward Rate

The forward outright is given by:

$$\text{Forward Outright} = \text{FX Spot} + \text{Forward Points} \times \text{Conversion Factor}$$

with conversion factor set to 1.

Property	**Allowable values**	**Description**
Instrument Type	*FX_FWD*	
Quote Type	*PRICE*	
Unit currency	See `Currency` in Table 20	Unit/Source currency
Target currency	See `Currency` in Table 20	Target currency
Term	An integer followed by D, W, M or Y.	Period from today to maturity. Alternatively, the maturity date.

Table 42: FX Forward Rate

Example:

- FXFWD/RATE/EUR/USD/1M
- FXFWD/PRICE/EUR/USD/3M
- FXFWD/PRICE/EUR/USD/2026-03-02

16.5 Deposit Rate

Property	Allowable values	Description
Instrument Type	*MM*	
Quote Type	*RATE*	
Currency	See `Currency` in Table 20	Currency of the Deposit rate
IndexName	Optional, any string	Generally used to differentiate money market rates referencing different interest rate indices with the same tenor
Forward start	An integer followed by D, W, M or Y.	Period from today to start
Term	An integer followed by D, W, M or Y.	Period from start to maturity

Table 43: Deposit Rate

Deposits are usually quoted as ON (Overnight), TN (Tomorrow Next), SN (Spot Next), SW (Spot Week), 3W (3 Weeks), 6M (6 Months), etc.
Forward start for ON is today (i.e. forward start = 0D), for TN tomorrow (forward start = 1D), for SN two days from today (forward start = 2D). For longer term Deposits, forward start is derived from conventions, see 12.30, and is between 0D and 2D, i.e. "spot days" are between 0 and 2.

Example:

- MM/RATE/EUR/2D/3M

16.6 FRA Rate

Property	Allowable values	Description
Instrument Type	*FRA*	
Quote Type	*RATE*	
Currency	See `Currency` in Table 20	Currency of the FRA rate
Forward start	An integer followed by D, W, M or Y	Period from today to start
Term	An integer followed by D, W, M or Y	Period from start to maturity

Table 44: FRA Rate

FRAs are typically quoted as e.g. 6x9 which means forward start 6M from today, maturity 9M from today, with appropriate adjustment of dates.

IMM FRA quotes are represented as follows.

Property	Allowable values	Description
Instrument Type	*IMM_FRA*	
Quote Type	*RATE*	
Currency	See `Currency` in Table 20	Currency of the FRA rate
Start	An integer	Number of IMM dates from today to start
End	An integer	Number of IMM dates from today to maturity

Table 45: IMM FRA Rate

Example:

- FRA/RATE/EUR/9M/3M
- IMM_FRA/RATE/EUR/2/3

16.7 Money Market Futures Price

Property	Allowable values	Description
Instrument Type	*MM_FUTURE*	
Quote Type	*PRICE*	
Currency	See `Currency` in Table 20	Currency of the MM Future price
Expiry	Alphanumeric string of the form YYYY-MM	Expiry month and year
Contract	String	Contract name
Term	An integer followed by D, W, M or Y	Underlying Term

Table 46: Money Market Futures Price

Expiry month is quoted here as YYYY-MM. The exact expiry date follows from a date rule defined in the future conventions, see 12.30.3.

Example:

- MM_FUTURE/PRICE/EUR/2018-06/LIF3ME/3M

16.8 Overnight Index Futures Price

Property	Allowable values	Description
Instrument Type	*OI_FUTURE*	
Quote Type	*PRICE*	
Currency	See `Currency` in Table 20	Currency of the Overnight Index Future price
Expiry	Alphanumeric string of the form YYYY-MM	Expiry month and year
Contract	String	Contract name
Term	An integer followed by M or Y	Underlying Term in months or years

Table 47: Overnight Index Futures Price

Expiry month is quoted here as YYYY-MM. The exact expiry date follows from a date rule defined in the future conventions, see 12.30.3.

Example: Three Months SOFR Futures (DEC 2019):

- OI_FUTURE/PRICE/USD/2019-12/CME:SR3Z2019/3M

16.9 Swap Rate

Property	Allowable values	Description
Instrument Type	*IR_SWAP*	
Quote Type	*RATE*	
Currency	See `Currency` in Table 20	Currency of the Swap rate
IndexName	Optional, any string	Generally used to differentiate swaps referencing different interest rate indices with the same tenor
Forward start	An integer followed by D, W, M or Y	Generic period from today to start
Tenor	An integer followed by D, W, M or Y	Underlying index period
Term	An integer followed by D, W, M or Y	Swap length from start to maturity

Table 48: Swap Rate

Property	Allowable values	Description
Instrument Type	*IR_SWAP*	
Quote Type	*RATE*	
Currency	See `Currency` in Table 20	Currency of the Swap rate
IndexName	Optional, any string	Generally used to differentiate swaps referencing different interest rate indices with the same tenor
Start Date	A valid date	
Tenor	An integer followed by D, W, M or Y	Underlying index period
End Date	A valid date	

Table 49: Swap Rate with Start and End Date

Forward start for the non-dated variant is usually not quoted, but needs to be derived from conventions.

Example:

- IR_SWAP/RATE/EUR/2D/6M/10Y
- IR_SWAP/RATE/GBP/20230921/1D/20231102

16.10 Basis Swap Spread

Property	Allowable values	Description
Instrument Type	*BASIS_SWAP*	
Quote Type	*BASIS_SPREAD*	
Flat tenor	An integer followed by D, W, M or Y	Zero spread leg's index tenor
Tenor	An integer followed by D, W, M or Y	Non-zero spread leg's index tenor
Currency	See `Currency` in Table 20	Currency of the basis swap spread
Optional Identifier	String	Basis swap name
Term	An integer followed by D, W, M or Y	Swap length from start to maturity

Table 50: Basis Swap Spread

Examples:

- BASIS_SWAP/BASIS_SPREAD/6M/3M/CHF/10Y
- BASIS_SWAP/BASIS_SPREAD/3M/1D/USD/2Y
- BASIS_SWAP/BASIS_SPREAD/3M/1D/USD/LIBOR_PRIME/2Y
- BASIS_SWAP/BASIS_SPREAD/3M/1D/USD/LIBOR_FEDFUNDS/2Y

16.11 Cross Currency Basis Swap Spread

Property	Allowable values	Description
Instrument Type	*CC_BASIS_SWAP*	
Quote Type	*BASIS_SPREAD*	
Flat currency	See `Currency` in Table 20	Currency for zero spread leg
Flat tenor	An integer followed by D, W, M or Y	Zero spread leg's index tenor
Currency	See `Currency` in Table 20	Currency for non-zero spread leg
Tenor	An integer followed by D, W, M or Y	Non-zero spread leg's index tenor
Term	An integer followed by D, W, M or Y	Swap length from start to maturity

Table 51: Cross Currency Basis Swap Spread

Example:

- CC_BASIS_SWAP/BASIS_SPREAD/USD/3M/JPY/6M/10Y

16.12 CDS Spread

Property	Allowable values	Description
Instrument Type	`CDS`	
Quote Type	`CREDIT_SPREAD` or `CONV_CREDIT_SPREAD`	
Entity	String	The CDS reference entity name
Tier	String	The CDS tier
DocClause	String	Optional, the CDS doc clause
Currency	See `Currency` in Table 20	The CDS currency
Term	A valid tenor string	The CDS tenor
RunningSpread	A number	The CDS running coupon in bps e.g. 100 for 0.01

Table 52: CDS spread quote

Currently, only par CDS spreads (quote type `CREDIT_SPREAD`) are supported. Conventional CDS spreads (quote type `CONV_CREDIT_SPREAD`) are not supported at this time. As noted in the table above, the CDS documentation clause and CDS running spread is optional. The following list shows valid CDS spread quote examples.

- `CDS/CREDIT_SPREAD/JPM/SNRFOR/USD/5Y`
- `CDS/CREDIT_SPREAD/JPM/SNRFOR/USD/5Y/100`
- `CDS/CREDIT_SPREAD/JPM/SNRFOR/USD/XR14/5Y`

- CDS/CREDIT_SPREAD/JPM/SNRFOR/USD/XR14/5Y/100
- CDS/CREDIT_SPREAD/RBS/SUBLT2/EUR/MR14/10Y
- CDS/CREDIT_SPREAD/RBS/SUBLT2/EUR/MR14/10Y/500
- CDS/CREDIT_SPREAD/RBS/SUBLT2/EUR/1Y
- CDS/CREDIT_SPREAD/RBS/SUBLT2/EUR/1Y/500

16.13 CDS Upfront Price

Property	Allowable values	Description
Instrument Type	CDS	
Quote Type	PRICE	
Entity	String	The CDS reference entity name
Tier	String	The CDS tier
Currency	See Currency in Table 20	The CDS currency
DocClause	String	Optional, the CDS doc clause
Term	A valid tenor string	The CDS tenor
RunningSpread	A number	The CDS running coupon in bps e.g. 100 for 0.01

Table 53: CDS upfront price quote

As noted in the table above, the CDS documentation clause and CDS running spread is optional. Note that if the running spread is omitted from the CDS upfront price quote string, it should be included in any default curve configuration that uses those quotes. In other words, to bootstrap a default curve from CDS price quotes, the contractual running spread needs to be provided in either the quote string or in the default curve configuration. If both are provided, the running spread in the quote string takes precedence. The following list shows valid CDS upfront price quote examples.

- CDS/PRICE/JPM/SNRFOR/USD/5Y
- CDS/PRICE/JPM/SNRFOR/USD/5Y/100
- CDS/PRICE/JPM/SNRFOR/USD/XR14/5Y
- CDS/PRICE/JPM/SNRFOR/USD/XR14/5Y/100
- CDS/PRICE/RBS/SUBLT2/EUR/MR14/10Y
- CDS/PRICE/RBS/SUBLT2/EUR/MR14/10Y/500
- CDS/PRICE/RBS/SUBLT2/EUR/1Y
- CDS/PRICE/RBS/SUBLT2/EUR/1Y/500

16.14 CDS Recovery Rate

Property	Allowable values	Description
Instrument Type	`RECOVERY_RATE`	
Quote Type	`RATE`	
Entity	String	The CDS reference entity name
Tier	String	The CDS tier
Currency	See `Currency` in Table 20	The CDS currency
DocClause	String	Optional, the CDS doc clause

Table 54: CDS Recovery Rate

As noted in the table above, the CDS documentation clause is optional. The following list shows valid recovery rate quote examples.

- `RECOVERY_RATE/RATE/JPM/SNRFOR/USD`
- `RECOVERY_RATE/RATE/JPM/SNRFOR/USD/XR14`
- `RECOVERY_RATE/RATE/RBS/SUBLT2/EUR/MR14`
- `RECOVERY_RATE/RATE/RBS/SUBLT2/EUR`

16.15 CDS Option Implied Volatility

A CDS option implied volatility quote can take any one of the following four forms:

1. `INDEX_CDS_OPTION/RATE_LNVOL/[NAME]/[EXPIRY]`
2. `INDEX_CDS_OPTION/RATE_LNVOL/[NAME]/[EXPIRY]/[STRIKE]`
3. `INDEX_CDS_OPTION/RATE_LNVOL/[NAME]/[TERM]/[EXPIRY]`
4. `INDEX_CDS_OPTION/RATE_LNVOL/[NAME]/[TERM]/[EXPIRY]/[STRIKE]`

The terms in the quote string have the following interpretations:

- The `[NAME]` is the name of the CDS reference entity or index CDS.
- The `[EXPIRY]` is the expiry of the CDS option and may be a tenor or an explicit date.
- The `[TERM]` is optional and gives the term of the underlying CDS or index CDS. This should be a tenor e.g. 3Y, 5Y, etc.
- The `[STRIKE]` is optional and gives the strike of the CDS or index CDS option.

16.16 Security Recovery Rate

Bond recovery rates can also be specified per security. This requires only one key, the security ID, no need to specify a seniority or currency as for CDS:

Property	Allowable values	Description
Instrument Type	*RECOVERY_RATE*	
Quote Type	*RATE*	
ID	String	Security ID

Table 55: Security Recovery Rate

Example:

- RECOVERY_RATE/RATE/SECURITY_1

16.17 Hazard Rate (Instantaneous Probability of Default)

This allows to directly pass hazard rates as instantaneous probabilities of default.

Property	Allowable values	Description
Instrument Type	*HAZARD_RATE*	
Quote Type	*RATE*	
Issuer	String	Issuer name
Seniority	String	Seniority status
Currency	See `Currency` in Table 20	Hazard rate currency
Term	An integer followed by D, W, M or Y	Generic period from start to maturity

Table 56: Hazard Rate

Example:

- HAZARD_RATE/RATE/CPTY_A/SR/USD/30Y
- HAZARD_RATE/RATE/CPTY_C/SR/EUR/0Y

16.18 FX Option Implied Volatility

Property	Allowable values	Description
Instrument Type	*FX_OPTION*	
Quote Type	*RATE_LNVOL*	
Unit currency	See `Currency` in Table 20	Unit/Source currency
Target currency	See `Currency` in Table 20	Target currency
Expiry	An integer followed by D, W, M or Y	Period from today to expiry
Strike	*ATM, RR, BF*	ATM (Straddle), RR (Risk Reversal), BF (Butterfly)

Table 57: FX Option Implied Volatility

Volatilities are quoted in terms of strategies - at-the-money straddle, risk reversal and butterfly.

Example:

- FX_OPTION/RATE_LNVOL/EUR/USD/3M/ATM

16.19 Cap Floor Implied Volatility

Property	Allowable values	Description
Instrument Type	*CAPFLOOR*	
Quote Type	*RATE_LNVOL, RATE_NVOL, RATE_SLNVOL, SHIFT, PRICE*	Lognormal quoted volatility, normal quoted volatility, shifted lognormal quoted volatility, shift quote, premium quote.
Currency	See `Currency` in Table 20	Currency of the cap floor quote.
Index Name	A string	(optional) An interest rate index name giving the index underlying the cap floor quotes. See Table 26.
Term	A valid tenor string	Period from start to cap or floor maturity.
Index Tenor	A valid tenor string	Underlying index tenor e.g. `3M` for `EUR-EURIBOR-3M`.
ATM	1 or `0`	True, i.e. 1, for an ATM quote and false, i.e. `0`, for a strike quote.
Relative	1 or `0`	Should be set to 1 for a quote relative to ATM and to `0` for an absolute strike quote.
Strike	Real number	Strike of cap or floor. Should be set to `0` for an ATM quote.
Option Style	C or F	(optional) Valid for premium quotes only, indicates whether the datum is a cap or floor quote respectively.

Table 58: Cap floor implied volatility quote

An index name should be used where a currency has more than one index of a given tenor with an options surface. It must match `IborIndex` in its corresponding `CapFloorVolatility` configuration, see section 12.2.8.

If a cap floor shift quote needs to be provided, i.e. in the case of a shifted lognormal surface, the quote is of the form `CAPFLOOR/SHIFT/Currency/Index Tenor` where the meaning of `Currency` and `Index Tenor` are given in Table 58.

We have the following examples of cap floor implied volatility, shift, and premium quotes:

- `CAPFLOOR/RATE_LNVOL/EUR/10Y/6M/1/1/0`: 10Y ATM cap floor implied lognormal volatility quote where the index tenor is 6M.
- `CAPFLOOR/RATE_LNVOL/EUR/10Y/6M/0/0/0.035`: 10Y 3.5% strike cap floor implied lognormal volatility quote where the index tenor is 6M.

- `CAPFLOOR/RATE_SLNVOL/EUR/EURIBOR/5Y/6M/0/0/0.03`: 5Y 3% strike cap floor implied shifted lognormal volatility quote where the underlying rate is the EUR-EURIBOR-6M.
- `CAPFLOOR/SHIFT/EUR/EURIBOR/6M`: Strike shift convention corresponding to shifted lognormal implied capfloor volatility quotes where the underlying rate is the EUR-EURIBOR-6M.
- `CAPFLOOR/PRICE/EUR/EURIBOR/5Y/6M/0/0/0.03/C`: 5Y 3% strike cap floor premium quote where the underlying rate is the EUR-EURIBOR-6M.

16.20 Swaption Implied Volatility

Property	Allowable values	Description
Instrument Type	*SWAPTION*	
Quote Type	*RATE_LNVOL, RATE_NVOL, RATE_SLNVOL, SHIFT, PRICE*	Lognormal quoted volatility, Normal quoted volatility, shifted lognormal quoted volatility, shift, premium quote.
Currency	See `Currency` in Table 20	Currency of the Swaption volatility
Quote Tag	A string	(optional) A tag to differentiate different sets of swaption data in a currency. See note below.
Expiry	An integer followed by D, W, M or Y	Period from start to expiry
Term	An integer followed by D, W, M or Y	Underlying Swap term
Dimension	*Smile, ATM*	Whether volatility quote is a Smile or ATM
Strike	Real number	(not required for ATM), deviation from the ATM strike. Note that trailing 0s are not ignored.
Option Style	P or R	(optional) Valid for premium quotes only, indicates whether the datum represents a payer or receiver swaption premium respectively.

Table 59: Swaption Implied Volatility

A quote tag should be used where a currency has more than one index with a swaption surface. It must match `QuoteTag` in its corresponding `SwaptionVolatility` configuration, see section 12.2.7.

Note: The volatility quote is expected to be an absolute volatility, and not the deviation from the at-the-money volatility (the latter is e.g. the quotation convention used by BGC partners). If a swaption shift quote needs to be provided, i.e. in the case of a shifted lognormal surface, the quote is of the form `SWAPTION/SHIFT/Currency/Term` where the meaning of `Currency` and `Term` are given in Table 59.

Examples:

- SWAPTION/RATE_LNVOL/EUR/5Y/10Y/ATM (absolute ATM vol quote)
- SWAPTION/RATE_LNVOL/EUR/5Y/10Y/Smile/0.0050 (absolute vol quote for ATM strike plus 50bp)

16.21 Equity Spot Price

Property	Allowable values	Description
Instrument Type	*EQUITY_SPOT*	
Quote Type	*PRICE*	
Name	String	Identifying name of the equity
Currency	See `Currency` in Table 20	Currency of the equity

Table 60: Equity Spot Price

16.22 Equity Forward Price

Property	Allowable values	Description
Instrument Type	*EQUITY_FWD*	
Quote Type	*PRICE*	
Name	String	Identifying name of the equity
Currency	See `Currency` in Table 20	Currency of the equity
Maturity	Date string, or integer followed by D, W, M or Y	Maturity of the forward quote

Table 61: Equity Forward Price

Examples:

- EQUITY_FWD/PRICE/SP5/USD/2016-06-16
- EQUITY_FWD/PRICE/SP5/USD/2Y

16.23 Equity Dividend Yield

Property	Allowable values	Description
Instrument Type	*EQUITY_DIVIDEND*	
Quote Type	*RATE*	
Name	String	Identifying name of the equity
Currency	See `Currency` in Table 20	Currency of the equity
Maturity	Date string, or integer followed by D, W, M or Y	Maturity of the forward quote

Table 62: Equity Dividend Yield Rate

Examples:

- EQUITY_DIVIDEND/RATE/SP5/USD/2016-06-16
- EQUITY_DIVIDEND/RATE/SP5/USD/2Y

16.24 Equity Option Implied Volatility

Property	Allowable values	Description
Instrument Type	*EQUITY_OPTION*	
Quote Type	*RATE_LNVOL*	
Name	String	Identifying name of the equity
Currency	See `Currency` in Table 20	Currency of the equity
Expiry	Date string, or integer followed by D, W, M or Y	Maturity of the forward quote
Strike	`ATM/AtmSpot` (= `ATM`), `ATM/AtmFwd` (=`ATMF`), `MNY/[Spot|Fwd]/1.2` where 1.2 is the moneyness level, or a `Real` for an absolute strike	strike
CallPut	`C` for Call, `P` for Put	Optional Call/Put flag (defaults to Call)

Table 63: Equity Option Implied Volatility

Volatilities are quoted as a function of strike price - either at-the-money spot or forward, a moneyness level or else a specified real number, corresponding to the absolute strike value. Only log-normal implied volatilities (`RATE_LNVOL`) are supported.

If K is the absolute strike, S the spot, F the forward and m the moneyness level, we have $K = Sm$ if spot moneyness and $K = Fm$ if forward moneyness is specified.

Example:

- EQUITY_OPTION/RATE_LNVOL/SP5/USD/6M/ATMF
- EQUITY_OPTION/RATE_LNVOL/SP5/USD/2018-06-30/ATMF
- EQUITY_OPTION/RATE_LNVOL/SP5/USD/6M/MNY/Fwd/1.2

16.25 Equity Option Premium

Property	Allowable values	Description
Instrument Type	*EQUITY_OPTION*	
Quote Type	*PRICE*	
Name	String	Identifying name of the equity
Currency	See `Currency` in Table 20	Currency of the equity
Expiry	Date string, or integer followed by D, W, M or Y	Maturity of the forward quote
Strike	`ATM/AtmSpot` (= `ATM`), `ATM/AtmFwd` (=`ATMF`) or a `Real` for an absolute strike	strike
CallPut	`C` for Call, `P` for Put	Optional Call/Put flag (defaults to Call)

Table 64: Equity Option Premium

Premiums are quoted as a function of strike price - either at-the-money spot or forward or else a specified real number, corresponding to the absolute strike value.

Example:

- EQUITY_OPTION/PRICE/SP5/USD/6M/ATMF
- EQUITY_OPTION/PRICE/SP5/USD/2018-06-30/2000
- EQUITY_OPTION/PRICE/SP5/USD/2018-06-30/2000/C

16.26 Commodity Spot Price

Property	Allowable values	Description
Instrument Type	*COMMODITY*	
Quote Type	*PRICE*	
Name	String	Identifying name of the commodity
Currency	See `Currency` in Table 20	Currency of the commodity

Table 65: Commodity Spot Price

Examples:

- COMMODITY/PRICE/PM:XAUUSD/USD

16.27 Commodity Forward Price

Property	Allowable values	Description
Instrument Type	*COMMODITY_FWD*	
Quote Type	*PRICE*	
Name	String	Identifying name of the commodity
Currency	See `Currency` in Table 20	Currency of the commodity
Maturity	Date string, or integer followed by D, W, M or Y	Maturity of the forward quote

Table 66: Commodity Forward Price

Examples:

- COMMODITY_FWD/PRICE/NYMEX:CL/USD/2030-11-20

16.28 Commodity Option Implied Volatility

Property	Allowable values	Description
Instrument Type	*COMMODITY_OPTION*	
Quote Type	*RATE_LNVOL*	
Name	String	Identifying name of the commodity
Currency	See `Currency` in Table 20	Currency of the commodity
Expiry	Date string, or integer followed by D, W, M or Y or continuation notation c1, c2, etc	Expiry of the volatility quote, for continuation notation c1 indicates the next expiry, c2 the one after that, etc.
Vol Quote Type, or Absolute Strike	`DEL`, `ATM` or `MNY` or a `Real` for absolute strikes	`DEL` is for delta quotes, i.e. the volatility is for a call or put option with a given delta. `ATM` is for At-The-Money volatility quotes. `MNY` is for volatility smile quotes for given relative moneyness levels. Each Vol Quote Type is described further in sub-tables below. Note that instead of a Vol Quote Type, an absolute strike level can be entered

Table 67: Commodity Option Implied Volatility - Root table

Property	Allowable values	Description
Vol Quote Type	`DEL`, `ATM` or `MNY`	In this table it is assumed `DEL` is chosen.
Delta Convention	`Fwd` or `Spot`	Delta forward or spot quote
Option Type	`Call` or `Put`	Option type for the delta quote
Delta	a positive `Real` number, typically between 0.1 and 0.45	Delta, e.g. a delta of 0.40 for a Call means that if the underlying commodity price increases by 1 unit, the call option price increases by 0.40 units

Table 68: Commodity Option Implied Volatility - Delta Quotes Table

Property	Allowable values	Description
Vol Quote Type	`DEL`, `ATM` or `MNY`	In this table it is assumed `ATM` is chosen.
Atm Convention	`AtmFwd`, `AtmSpot`, or `AtmDeltaNeutral`	Atm Forward or Atm Spot quote, to be used standalone, or when the smile is given by moneyness (`MNY`) quotes. The Delta Neutral Atm to be used as standalone, or when the smile is given by delta (`DEL`) quotes. Note that when `AtmFwd` or `AtmSpot` are used, the string stops here, no further entries "tokens" are required
Atm Quote Type	`DEL`	When `AtmDeltaNeutral` is used, the quote type must be set to Delta (`DEL`)
Atm Delta Convention	`Fwd` or `Spot`	When `AtmDeltaNeutral` is used, the Atm delta quote can be a Spot or Forward quote.

Table 69: Commodity Option Implied Volatility - ATM Quotes Table

Property	Allowable values	Description
Vol Quote Type	`DEL`, `ATM` or `MNY`	In this table it is assumed `MNY` is chosen.
Moneyness Type	`Fwd` or `Spot`	Moneyness Forward or Spot quote
Moneyness	a positive `Real` number	The relative moneyness expressed in decimal form, relative to the AtmSpot or the AtmFwd strikes

Table 70: Commodity Option Implied Volatility - Moneyness Quotes Table

Volatilities are quoted:

- as a function of the delta - either the delta neutral at-the-money spot or forward, or for a call or put option with a given delta, or
- as a function of strike price - either at-the-money spot or forward, or a relative strike moneyness level, or else
- as a specified real number, corresponding to the absolute strike value.

Only log-normal implied commodity volatilities (`RATE_LNVOL`) are supported.

For strike quoted volatilities, If K is the absolute strike, S the spot, F the forward and m the moneyness level, we have $K = Sm$ if spot moneyness and $K = Fm$ if forward moneyness is specified.

Example of delta forward quotes:

COMMODITY_OPTION/RATE_LNVOL/ICE:B/USD/c9/DEL/Fwd/Put/0.40
COMMODITY_OPTION/RATE_LNVOL/ICE:B/USD/c9/DEL/Fwd/Put/0.45
COMMODITY_OPTION/RATE_-
LNVOL/ICE:B/USD/c9/ATM/AtmDeltaNeutral/DEL/Fwd
COMMODITY_OPTION/RATE_LNVOL/ICE:B/USD/c9/DEL/Fwd/Call/0.45
COMMODITY_OPTION/RATE_LNVOL/ICE:B/USD/c9/DEL/Fwd/Call/0.40

Example of delta spot quotes:

COMMODITY_OPTION/RATE_LNVOL/PM:XAGEUR/EUR/1Y/DEL/Spot/Put/0.35
COMMODITY_OPTION/RATE_LNVOL/PM:XAGEUR/EUR/1Y/DEL/Spot/Put/0.45
COMMODITY_OPTION/RATE_-
LNVOL/PM:XAGEUR/EUR/1Y/ATM/AtmDeltaNeutral/DEL
/Spot
COMMODITY_OPTION/RATE_LNVOL/PM:XAGEUR/EUR/1Y/DEL/Spot/Call/0.25
COMMODITY_OPTION/RATE_LNVOL/PM:XAGEUR/EUR/1Y/DEL/Spot/Call/0.15

Example of forward strike quotes with relative moneyness:

COMMODITY_OPTION/RATE_LNVOL/NYMEX:AA5/USD/c11/MNY/Fwd/1.40
COMMODITY_OPTION/RATE_LNVOL/NYMEX:AA5/USD/c11/MNY/Fwd/1.20
COMMODITY_OPTION/RATE_LNVOL/NYMEX:AA5/USD/c11/ATM/AtmFwd
COMMODITY_OPTION/RATE_LNVOL/NYMEX:AA5/USD/c11/MNY/Fwd/0.80
COMMODITY_OPTION/RATE_LNVOL/NYMEX:AA5/USD/c11/MNY/Fwd/0.60

Example of absolute moneyness quotes:

COMMODITY_OPTION/RATE_LNVOL/PM:XAUUSD/USD/c12/1600
COMMODITY_OPTION/RATE_LNVOL/PM:XAUUSD//USD/c12/1700
COMMODITY_OPTION/RATE_LNVOL/PM:XAUUSD//USD/c12/1800
COMMODITY_OPTION/RATE_LNVOL/PM:XAUUSD//USD/c12/1900
COMMODITY_OPTION/RATE_LNVOL/PM:XAUUSD//USD/c12/2000

16.29 Zero Coupon Inflation Swap Rate

Property	Allowable values	Description
Instrument Type	*ZC_INFLATION-SWAP*	
Quote Type	*RATE*	
Index	String	Identifying name of the inflation index
Maturity	integer followed by D, W, M or Y	Maturity of the swap quote

Table 71: Zero Coupon Inflation Swap Rate

Examples:

- ZC_INFLATIONSWAP/RATE/EUHICPXT/1Y
- ZC_INFLATIONSWAP/RATE/EUHICPXT/2Y

Examples for inflation index names include EUHICP, EUHICPXT, FRHICP, FRCPI, UKRPI, USCPI, ZACPI, BEHICP, AUCPI.

16.30 Year on Year Inflation Swap Rate

Property	Allowable values	Description
Instrument Type	*YY_INFLATION-SWAP*	
Quote Type	*RATE*	
Index	String	Identifying name of the inflation index
Maturity	integer followed by D, W, M or Y	Maturity of the swap quote

Table 72: Year on Year Inflation Swap Rate

Examples:

- YY_INFLATIONSWAP/RATE/EUHICPXT/1Y
- YY_INFLATIONSWAP/RATE/EUHICPXT/2Y

Examples for inflation index names include EUHICP, EUHICPXT, FRHICP, FRCPI, UKRPI, USCPI, ZACPI, BEHICP.

16.31 Zero Coupon Inflation Cap Floor Price

Property	Allowable values	Description
Instrument Type	*ZC_INFLATION-CAPFLOOR*	
Quote Type	*PRICE*	
Index	String	Identifying name of the inflation index
Maturity	integer followed by D, W, M or Y	Maturity of the swap quote
Cap/Floor	C or F	Cap or Floor tag
Strike	Real number	Strike

Table 73: Zero Coupon Inflation Cap Floor Price

Examples:

- ZC_INFLATIONCAPFLOOR/PRICE/EUHICPXT/1Y/F/-0.02
- ZC_INFLATIONCAPFLOOR/PRICE/EUHICPXT/2Y/C/0.01

Examples for inflation index names include EUHICP, EUHICPXT, FRHICP, FRCPI, UKRPI, USCPI, ZACPI, BEHICP.

16.32 Inflation Seasonality Correction Factors

Property	Allowable values	Description
Instrument Type	*SEASONALITY*	
Quote Type	*RATE*	
Type	MULT	Type of the correction factor
Index	String	Identifying name of the inflation index
Month	JAN, ..., DEC	Month of the correction factor

Table 74: Inflation Seasonality Correction Factors

Examples:

- SEASONALITY/RATE/MULT/EUHICPXT/JAN
- SEASONALITY/RATE/MULT/EUHICPXT/FEB
- SEASONALITY/RATE/MULT/EUHICPXT/NOV

Examples for inflation index names include EUHICP, EUHICPXT, FRHICP, FRCPI, UKRPI, USCPI, ZACPI, BEHICP.

16.33 Bond Yield Spreads

Property	Allowable values	Description
Instrument Type	*BOND*	
Quote Type	*YIELD_SPREAD*	
Name	String	Identifying name of the bond

Table 75: Bond Yield Spreads

This quote provides the spread for a specified bond over the benchmark rate.

Examples:

- BOND/YIELD_SPREAD/SECURITY_1

16.34 Bond Prices

Property	Allowable values	Description
Instrument Type	*BOND*	
Quote Type	*PRICE*	
Name	String	Identifying name of the bond

Table 76: Bond Prices

This quote provides the clean price for a specified bond in units.

Examples:

- BOND/PRICE/SECURITY_1

16.35 Bond Conversion Factor

Property	Allowable values	Description
Instrument Type	*BOND*	
Quote Type	*CONVERSION_-FACTOR*	
Name	String	Identifying name of the bond

Table 77: Bond Conversion Factor

This quote provides the conversion factor for a bond future contract.

Examples:

- BOND/CONVERSION_FACTOR/SECURITY_1

16.36 Base Correlations

Property	Allowable values	Description
Instrument Type	*CDS_INDEX*	
Quote Type	*BASE_CORRELA-TION*	
Index	String	CDS index name
Term	Period (e.g. 5Y)	Term on the base correlation curve, the curve is flat if quotes for only one term are provided
DetachmentPoint	Real in (0...1)	Detachment point of the equity tranche this quote refers to

Table 78: Base correlation quotes

This quote provides the base correlation for a CDS index' equity tranche with the specified detachment point. Example:

- CDS_INDEX/BASE_CORRELATION/2I65BYBD6/5Y/0.03

Typically there are several base correlation quotes per term for several detachment points such as 0.03, 0.07, 0.15.

16.37 Correlations

Property	Allowable values	Description
Instrument Type	*Correlation*	
Quote Type	*RATE or PRICE*	
Index1	String	Identifying name of the first index
Index2	String	Identifying name of the second index

Table 79: Correlation quotes

This quote either provides the correlation between two indices, in which case Quote Type is RATE, or a premium that can be used to bootstrap the correlations, in which case Quote Type is Price. Currently only CMS Spread correlations are supported, in this case the Price quote is the price of a CMS Spread Cap.

Examples:

- CORRELATION/RATE/INDEX1/INDEX2/1Y/ATM

16.38 Conditional Prepayment Rates

Property	**Allowable values**	**Description**
Instrument Type	*CPR*	
Quote Type	*RATE*	
Name	String	Identifying name of the bond

Table 80: Conditional Prepayment Rates

This quote provides the spread for a specified bond over the benchmark rate.

Examples:

- CPR/RATE/SECURITY_1

17 Fixing History

Historical fixings data in the `fixings.txt` file is given in three columns; Index Name, Fixing Date and Index value. Columns are separated by semicolons ";" or blanks. Fixings are used in cases where the current coupon of a trade has been fixed in the past, or other path dependent features.

- Fixing Date: The date of the fixing.

  Allowable values: See `Date` in Table 20.

- Index Name: The name of the Index.

  Allowable values are given in Table 81.

- Index Value: The index value for the given fixing date.

  Allowable values: Any real number (not expressed as a percentage or basis points).

An excerpt of a fixings file is shown in Listing 28. Note that alternative index name formats are used (Table 81).

Listing 28: Excerpt of a fixings file

```
20150202 EUR-EONIA -0.00024
20150202 EUR-EURIBOR-1M 0.00003
20150202 EUR-EURIBOR-1W -0.00022
20150202 EUR-EURIBOR-2W -0.00017
20150202 EUR-EURIBOR-3M 0.00055
20150202 EUR-EURIBOR-3M 0.00055
20150202 EUR-EURIBOR-6M 0.00134
20150202 EUR-EURIBOR-6M 0.00271
20150202 GBP-LIBOR-12M 0.009565
20150202 GBP-LIBOR-1M 0.0050381
20150202 GBP-LIBOR-1W 0.0047938
20150202 GBP-LIBOR-3M 0.0056338
20150202 GBP-LIBOR-6M 0.006825
20150202 JPY-LIBOR-12M 0.0026471
20150202 JPY-LIBOR-1M 0.0007143
20150202 JPY-LIBOR-1W 0.0004357
20150202 JPY-LIBOR-3M 0.0010429
20150202 JPY-LIBOR-6M 0.0014357
20150202 USD-LIBOR-12M 0.006194
20150202 USD-LIBOR-1M 0.001695
20150202 USD-LIBOR-1W 0.00136
20150202 USD-CMS-10Y 0.01500
20150202 EUR-CMS-20Y 0.01700
20150202 FX-ECB-EUR-USD 1.0919
20150801 FRHICP 100.36
```

IR Index of form CCY-INDEX-TENOR:	
Index Component	**Allowable Values**
CCY-INDEX	*EUR-EONIA* *EUR-EURIBOR* *EUR-LIBOR* *USD-FedFunds* *USD-Prime* *USD-LIBOR* *GBP-SONIA* *GBP-LIBOR* *JPY-TONAR* *JPY-LIBOR* *CHF-LIBOR* *CHF-TOIS* *AUD-LIBOR* *AUD-BBSW* *CAD-CDOR* *CAD-BA* *CAD-LIBOR* *SEK-STIBOR* *SEK-STINA* *SEK-LIBOR* *DKK-LIBOR* *DKK-CIBOR* *DKK-CITA* *SGD-SIBOR* *HKD-HIBOR* *CZK-PRIBOR* *HUF-BUBOR* *IDR-IDRFIX* *INR-MIFOR* *JPY-TIBOR* *JPY-EYTIBOR* *KRW-KORIBOR* *MXN-TIIE* *MYR-KLIBOR* *NOK-NIBOR* *NZD-BKBM* *PLN-WIBOR* *RUB-MOSPRIME* *SEK-STIBOR* *SGD-SOR* *SKK-BRIBOR* *TWD-TAIBOR* *THB-THBFIX* *THB-BIBOR* *ZAR-JIBAR* *DEM-LIBOR*
TENOR	An integer followed by *D, W, M or Y*

Table 81: Allowable values for IR indices.

If the interest rate index is for an overnight rate (e.g. EONIA), then the third token (i.e. the tenor) is not needed.

IR Swap Index of form CCY-CMS-TENOR or CCY-CMS-TAG-TENOR:	
Index Component	**Allowable Values**
CCY	Any supported currency code
CMS	Must be "CMS" (to denote a swap index)
TAG	An optional tag that allows to define several CMS indices per currency
TENOR	An integer followed by *D, W, M or Y*

Table 82: Allowable values for IR swap indices.

FX fixings of form FX-SOURCE-FOR-DOM:	
Index Component	**Allowable Values**
FX	Must be "FX" (to denote an FX fixing)
SOURCE	Any string
FOR	Any supported currency code
DOM	Any supported currency code

Table 83: Allowable values for FX fixings.

Zero Inflation Indices of form NAME:	
Index Component	**Allowable Values**
NAME	*CACPI* *DKCPI* *EUHICP* *EUHICPXT* *FRHICP* *FRCPI* *SECPI* *UKRPI* *USCPI* *ZACPI* *BEHICP*

Table 84: Allowable values for zero inflation indices.

Generic Indices of form GENERIC-NAME:	
Index Component	**Allowable Values**
GENERIC	Must be "GENERIC" (to denote a generic fixing)
NAME	Any string

Table 85: Allowable values for generic indices.

18 Dividends History

Historical dividend payments data in the `dividends.txt` file is given in three columns; Equity Name, Ex-Dividend Date and Dividend Amount in the Currency of the Equity Curve. Columns are separated by semicolons ";" or blanks. Dividends are used in some trades with path dependent features.

- Ex-Dividend Date: The day the stock starts trading without the value of the dividend payment

  Allowable values: See `Date` in Table 20.

- Equity Name: The name of the dividend paying equity.

  Allowable values are the names of the Equity Curves defined in `curveconfig.xml`.

- Dividend Amount: The amount of the dividend payment date.

  Allowable values: Any real number (not expressed as a percentage).

An excerpt of a fixings file is shown in Listing 29.

Listing 29: Excerpt of a dividends file

```
20130411 DAI:GR 2.2
20140410 DAI:GR 2.25
20150402 DAI:GR 2.45
20160407 DAI:GR 3.25
20170330 DAI:GR 3.25
20180406 DAI:GR 3.65
20190523 DAI:GR 3.25
20120815 HSBA:LN 5.5538
20121024 HSBA:LN 5.604
20130320 HSBA:LN 11.585
20130522 HSBA:LN 6.58
20130821 HSBA:LN 6.2033
20131023 HSBA:LN 6.102
20140312 HSBA:LN 11.2919
20140521 HSBA:LN 5.8768
20140820 HSBA:LN 6.1622
20141023 HSBA:LN 6.3633
20150305 HSBA:LN 13.4
20150521 HSBA:LN 6.3709
20150813 HSBA:LN 6.4436
20151022 HSBA:LN 6.6015
20160303 HSBA:LN 14.7908
20160519 HSBA:LN 7.5421
20160811 HSBA:LN 7.6633
20161020 HSBA:LN 8.0417
20170223 HSBA:LN 16.6757
20170518 HSBA:LN 7.8636
20170803 HSBA:LN 7.577
20171012 HSBA:LN 7.6405
20180222 HSBA:LN 14.762
20180517 HSBA:LN 7.5502
20180816 HSBA:LN 7.632
20181011 HSBA:LN 7.78
20190221 HSBA:LN 15.9271
20190516 HSBA:LN 7.8368
```

Trade data and trade components.

The maintained parameterisation chapters above describe the current authored and resolved native surfaces used to drive VRE runs. Portfolio payloads, trade payloads and scripted-trade payloads remain an explicit compatibility exception in the VRE v0.10.0 release line: those payloads are still authored and validated as XML because they encode the contractual trade representation consumed by the engine and the existing example, regression and client portfolios.

This chapter therefore remains an in-line maintained reference for the trade-level XML surface. It is not historical material. It documents the portfolio envelope, the trade-specific payloads for each supported product family, and the reusable XML trade components that appear across those payloads. Where relevant, the trade-data sections should be read together with the current native parameterisation chapters, because valuation, risk, reporting, compliance, capital and scope-based runs now depend on both surfaces:

- the native run bundle and resolved runtime publications for engine, market, simulation, sensitivity, reporting and capital configuration; and
- the XML portfolio and scripted-trade payloads that describe the actual instruments processed by those runs.

19 Trade Data

The trades that make up the portfolio are specified in an XML file where the portfolio data is specified in a hierarchy of nodes and sub-nodes. The nodes containing individual trade data are referred to as elements or XML elements. These are generally the lowest level nodes.

The top level portfolio node is delimited by an opening <Portfolio> and a closing </Portfolio> tag. Within the portfolio node, each trade is defined by a starting <Trade id="[Tradeid]"> and a closing </Trade> tag. Further, the trade type is set by the TradeType XML element. Each trade has an Envelope node that includes the same XML elements for all trade types (Id, Type, Counterparty, Rating, NettingSetId) plus the Additional fields node, and after that, a node containing trade specific data.

An example of a `portfolio.xml` file with one Swap trade including the full envelope node is shown in Listing 30.

Listing 30: Portfolio

```
<Portfolio>
  <Trade id="Swap#1">
    <TradeType> Swap </TradeType>
    <Envelope>
      <CounterParty> Counterparty#1 </CounterParty>
      <NettingSetId> NettingSet#2 </NettingSetId>
      <PortfolioIds>
          <PortfoliodId> PF#1 </PortfolioId>
          <PortfoliodId> PF#2 </PortfolioId>
      </PortfolioIds>
      <AdditionalFields>
        <Sector> SectorA </Sector>
        <Book> BookB </Book>
        <Rating> A1 </Rating>
      </AdditionalFields>
    </Envelope>
    <SwapData>
        ...
        [Trade specific data for a Swap]
        ...
    </SwapData>
  </Trade>
</Portfolio>
```

A description of all portfolio data, i.e. of each node and XML element in the portfolio file, with examples and allowable values follows below. There is only one XML elements directly under the top level `Portfolio` node:

- `TradeType`: The trade families documented in this chapter cover the maintained portfolio and scripted-trade XML surfaces available in the VRE v0.10.0 release line.

  Allowable values: *ForwardRateAgreement, Swap, CapFloor, Swaption, FxForward, FxSwap, FxOption, EquityForward, EquityOption, VarianceSwap, CommodityForward, CommodityOption, CreditDefaultSwap, Bond*

19.1 Envelope

The envelope node contains basic identifying details of a trade (`Id`, `Type`, `Counterparty`, `NettingSetId`), a `PortfolioIds` node containing a list of portfolio assignments, plus an `AdditionalFields` node where custom elements can be added for informational purposes such as `Book` or `Sector`. Beside the custom elements within the `AdditionalFields` node, the envelope contains the same elements for all Trade types. The `Id`, `Type`, `Counterparty` and `NettingSetId` elements must have non-blank entries for VRE to run. The meanings and allowable values of the various elements in the `Envelope` node follow below.

- `Id`: The `Id` element in the envelope is used to identify trades within a portfolio. It should be set to identical values as the `Trade id=" "` element.

  Allowable values: Any alphanumeric string. The underscore (_) sign may be used as well.

- `Counterparty`: Specifies the name of the counterparty of the trade. It is used to show exposure analytics by counterparty.

Allowable values: Any alphanumeric string. Underscores (_) and blank spaces may be used as well.

- `NettingSetId` [Optional]: The `NettingSetId` element specifies the identifier for a netting set. If a `NettingSetId` is specified, the trade is eligible for close-out netting under the terms of an associated ISDA agreement. The specified `NettingSetId` must be defined within the netting set definitions file (see section 15). If left blank or omitted the trade will not belong to any netting set, and thus not be eligible for netting.

  Allowable values: Any alphanumeric string. Underscores (_) and blank spaces may be used as well.

- `PortfolioIds` [Optional]: The PortfolioIds node allows the assignment of a given trade to several portfolios, each enclosed in its own pair of tags <PortfolioId> and </PortfolioId> . Note that VRE does not assume a hierarchical organisation of such portfolios. If present, the portfolio IDs will be used in the generation of some VRE reports such as the VaR report which provides breakdown by any portfolio id that occurs in the trades' envelopes.

  Allowable values for each PortfolioId: Any string.

- `AdditionalFields` [Optional]: The AdditionalFields node allows the insertion of additional trade information using custom XML elements. For example, elements such as Sector, Desk or Folder can be used. The elements within the `AdditionalFields` node are used for informational purposes only, and do not affect any analytics in VRE.

  Allowable values: Any custom element.

19.1.1 Netting Set Details

Instead of a single netting set ID, defined by a `NettingSetId` node, an alternative `NettingSetDetails` node can be provided, which itself contains a `NettingSetId` sub-node, and four other optional sub-nodes, which altogether allow for extending the uniqueness of netting sets beyond the netting set ID. The allowable values for each sub-node are any alphanumeric string. The underscore ('_') sign may be used as well.

The `NettingSetDetails` node is given in the following XML format:

Listing 31: Netting set details

```
<NettingSetDetails>
    <NettingSetId> </NettingSetId>
    <AgreementType> </AgreementType>
    <CallType> </CallType>
    <InitialMarginType> </InitialMarginType>
    <LegalEntityId> </LegalEntityId>
</NettingSetDetails>
```

19.2 Trade Specific Data

After the envelope node, trade-specific data for each trade type supported by VRE is included. Each trade type has its own trade data container which is defined by an XML node

containing a trade-specific configuration of individual XML tags - called elements - and trade components. The trade components are defined by XML sub-nodes that can be used within multiple trade data containers, i.e. by multiple trade types.

Details of trade-specific data for all trade types follow below.

19.2.1 Swap

The `SwapData` node is the trade data container for the *Swap* trade type. A Swap must have at least one leg, and can have an unlimited number of legs. Each leg is represented by a `LegData` trade component sub-node, described in section 19.3.3. An example structure of a two-legged `SwapData` node is shown in Listing 32.

- Settlement [Optional]: Delivery type applicable to cross currency swaps, and ignored for all other swap types. Delivery type does not impact pricing in VRE, but npv results are produced with and without SIMM exemptions.

  Settlement *Cash* indicates that principal exchanges on the cross currency swap should be included in Initial Margin (IM). According to ISDA non-deliverable (*Cash*) trades are excluded from the exemption from IM for the principal exchange, i.e. the principal exchanges are included in IM.

  Settlement *Physical* indicates that principal exchanges on the cross currency swap should be excluded in IM (the ISDA exemption applies).

  Allowable values: *Cash* or *Physical*. Defaults to *Physical* if left blank or omitted.

Listing 32: Swap data

```
<SwapData>
  <Settlement>Cash</Settlement>
  <LegData>
    ...
  </LegData>
  <LegData>
    ...
  </LegData>
</SwapData>
```

Note that Swaps in non-deliverable currencies with payment in a deliverable currency are supported by setting Settlement to *Cash* and - on both legs - using the Indexings node (19.3.8), as well as setting the Currency to the deliverable currency, while keeping the Notional expressed in the non-deliverable currency amount.

Within the Indexings node, an fx Index field is mandatory defining the deliverable and non-deliverable currencies and fixing source. The Indexing node can also include optional FixingCalendar, IsInArrears and FixingDays fields to determine the date(s) of the fx fixing(s). See Listing 33 for an example non-deliverable IR swap where USD is the payment currency and CLP is the non-deliverable currency.

Listing 33: Non deliverable single currency IR Swap

```
<SwapData>
  <Settlement>Cash</Settlement>
  <LegData>
   <LegType>Fixed</LegType>
   <Payer>false</Payer>
   <Currency>USD</Currency><!-- Payment currency is USD rather than CLP -->
    <Notionals>
     <Notional>850000000</Notional><!-- in CLP -->
    </Notionals>
    <Indexings>
      <Indexing>
        <Index>FX-TR20H-CLP-USD</Index><!-- to convert CLP flows into USD -->
        <FixingCalendar>CLP,USD</FixingCalendar>
        <IsInArrears>true</IsInArrears>
        <FixingDays>2</FixingDays>
      </Indexing>
    </Indexings>
    ...
  </LegData>
  <LegData>
   <LegType>Floating</LegType>
   <Payer>true</Payer>
   <Currency>USD</Currency><!-- Payment currency is USD rather than CLP -->
    <Notionals>
     <Notional>850000000</Notional><!-- in CLP -->
     </Notionals>
     <Indexings>
      <Indexing>
        <Index>FX-TR20H-CLP-USD</Index><!-- to convert CLP flows into USD -->
        <FixingCalendar>CLP,USD</FixingCalendar>
        <IsInArrears>true</IsInArrears>
        <FixingDays>2</FixingDays>
      </Indexing>
     </Indexings>
    ...
  </LegData>
</SwapData>
```

19.2.2 Zero Coupon Swap

A Zero Coupon swap is set up as a swap (trade type *Swap*) , with one leg of type `ZeroCouponFixed`. Listing 34 shows an example. The ZeroCouponFixed leg contains an additional `ZeroCouponFixedLegData` block. See 19.3.19 for details on the ZeroCouponFixed leg specification.

Listing 34: Zero Coupon Swap Data

```
<SwapData>
  <LegData>
    <LegType>Floating</LegType>
    <Payer>true</Payer>
    ...
  </LegData>
  <LegData>
    <LegType>ZeroCouponFixed</LegType>
    <Payer>false</Payer>
    ...
    <ZeroCouponFixedLegData>
        <Rates>
            <Rate>0.02</Rate>
        </Rates>
        <Compounding>Simple</Compounding>
    </ZeroCouponFixedLegData>
  </LegData>
</SwapData>
```

19.2.3 Cap/Floor

The `CapFloorData` node is the trade data container for the *CapFloor* trade type. It's a cap, floor or collar (i.e. a portfolio of a long cap and a short floor for a long position in the collar) on a series of Ibor, SIFMA, OIS, CMS, Duration-adjusted CMS, CMS Spread, CPI, YY coupons.

The `CapFloorData` node contains a `LongShort` sub-node which indicates whether the cap (floor, collar) is long or short, and a `LegData` sub-node where the LegType can be set to *Floating*, *CMS*, *CMSSpread*, *DurationAdjustedCMS*, *CPI* or *YY*, plus elements for the Cap and Floor rates. An example structure with Cap rates is shown in in Listing 35. The optional node *PaymentDates* in the `LegData` subnode is currently only used for OIS and IBOR indices (see 19.3.3).

A `CapFloorData` node must have either `Caps` or `Floors` elements, or both. In the case of both (I.e. a collar with long cap and short floor) the sequence is that `Caps` elements must be above the `Floors` elements. Note that the `Caps` and `Floors` elements must be outside the `LegData` sub-node, i.e. a *CapFloor* can't have a capped or floored *Floating* or *CMS* leg. The *Payer* flag in the LegData subnode is ignored for this instrument. Notice that the signs in the definition of a collar (long cap, short floor) for the CapFloor instruments is exactly opposite to 19.3.6.

Listing 35: Cap/Floor data

```
<CapFloorData>
  <LongShort>Long</LongShort>
  <LegData>
    <Payer>false</Payer>
    <LegType>Floating</LegType>
     ...
  </LegData>
  <Caps>
    <Cap>0.05</Cap>
  </Caps>
  <Premiums>
    <Premium>
      <Amount>1000</Amount>
      <Currency>EUR</Currency>
      <PayDate>2021-01-27</PayDate>
    </Premium>
  </Premiums>
</CapFloorData>
```

The meanings and allowable values of the elements in the `CapFloorData` node follow below.

- LongShort: This node defines the position in the cap (floor, collar) and can take values *Long* or *Short*.
- LegData: This is a trade component sub-node outlined in section 19.3.3. Exactly one `LegData` node is allowed, and the LegType element must be set to *Floating* (Ibor and OIS), *CMS*, *CMSSpread*, *DurationAdjustedCMS*, *CPI* or *YY*.
- Caps: This node has child elements of type `Cap` capping the floating leg (after applying spread if any). The first rate value corresponds to the first coupon, the second rate value corresponds to the second coupon, etc. If the number of coupons exceeds the number of rate values, the rate will be kept flat at the value of last entered rate for the remaining coupons. For a fixed cap rate over all coupons, one single rate value is sufficient. The number of entered rate values cannot exceed the number of coupons.

  Allowable values for each `Cap` element: Any real number. The rate is expressed in decimal form, eg 0.05 is a rate of 5%
- Floors: This node has child elements of type `Floor` flooring the floating leg (after applying spread if any). The first rate value corresponds to the first coupon, the second rate value corresponds to the second coupon, etc. If the number of coupons exceeds the number of rate values, the rate will be kept flat at the value of last entered rate for the remaining coupons. For a fixed floor rate over all coupons, one single rate value is sufficient. The number of entered rate values cannot exceed the number of coupons.

  Allowable values for each `Floor` element: Any real number. The rate is expressed in decimal form, eg 0.05 is a rate of 5%
- Premiums [Optional]: Option premium amounts paid by the option buyer to the option seller.

  Allowable values: See section 19.3.2

19.2.4 Forward Rate Agreement

A forward rate agreement (trade type *ForwardRateAgreement* is set up using a `ForwardRateAgreementData` block as shown in listing 36. The forward rate agreement specific elements are:

- StartDate: A FRA expires/settles on the startDate.
  Allowable values: See `Date` in Table 20.
- EndDate: EndDate is the date when the forward loan or deposit ends. It follows that (EndDate - StartDate) is the tenor/term of the underlying loan or deposit.

  Allowable values: See `Date` in Table 20.
- Currency: The currency of the FRA notional.
  Allowable values: See Table 22 `Currency`.
- Index: The name of the interest rate index the FRA is benchmarked against.

  Allowable values: An alphanumeric string of the form CCY-INDEX-TENOR. CCY, INDEX and TENOR must be separated by dashes (-). CCY and INDEX must be among the supported currency and index combinations. TENOR must be an integer followed by D, W, M or Y, except for Overnight indices which do not require a TENOR. See Table 26.
- LongShort: Specifies whether the FRA position is long (one receives the agreed rate) or short (one pays the agreed rate).

  Allowable values: *Long*, *Short*.
- Strike: The agreed forward interest rate.

  Allowable values: Any real number. The strike rate is expressed in decimal form, e.g. 0.05 is a rate of 5%.
- Notional: No accretion or amortisation, just a constant notional.
  Allowable values: Any positive real number.

Listing 36: Forward Rate Agreement Data

```
<ForwardRateAgreementData>
    <StartDate>20161028</StartDate>
    <EndDate>20351028</EndDate>
    <Currency>EUR</Currency>
    <Index>EUR-EURIBOR-6M</Index>
    <LongShort>Long</LongShort>
    <Strike>0.001</Strike>
    <Notional>1000000000</Notional>
</ForwardRateAgreementData>
```

19.2.5 Swaption

The `SwaptionData` node is the trade data container for the *Swaption* trade type. The `SwaptionData` node has one and exactly one `OptionData` trade component sub-node, and at least one `LegData` trade component sub-node. These trade components are outlined in section 19.3.1 and section 19.3.3.

Supported swaption exercise styles are *European*, *Bermudan*, *American*. Swaptions of all exercise styles can have an arbitrary number of legs, with each leg represented by a `LegData` sub-node. Cross currency swaptions are not supported for either exercise style, i.e. the Currency element must have the same value for all `LegData` sub-nodes of a swaption. There must be at least one full coupon period after the exercise date for European Swaptions, and after the last exercise date for Bermudan and American Swaptions. See Table 86 for further details on requirements for swaptions.

The structure of an example `SwaptionData` node of a European swaption is shown in Listing 37.

Listing 37: Swaption data

```
<SwaptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <Style>European</Style>
        <Settlement>Physical</Settlement>
        <ExerciseDates>
          <ExerciseDate>2027-03-02</ExerciseDate>
        </ExerciseDates>
        ...
        <Premiums>
          <Premium>
            <Amount>807000</Amount>
            <Currency>GBP</Currency>
            <PayDate>2021-06-15</PayDate>
          </Premium>
        </Premiums>
    </OptionData>
    <LegData>
        <LegType>Fixed</LegType>
        <Payer>false</Payer>
        <Currency>GBP</Currency>
        ...
    </LegData>
    <LegData>
        <LegType>Floating</LegType>
        <Payer>true</Payer>
        <Currency>GBP</Currency>
        ...
    </LegData>
</SwaptionData>
```

	A Swaption requires:
`OptionData`	One `OptionData` sub-node
`Style`	*Bermudan* or *European* or *American*
`ExerciseDates`	*European* swaptions can only have one `ExerciseDate` child element. *American* swaptions must have two `ExerciseDate` child elements. *Bermudan* swaptions must have at least two `ExerciseDate` child elements, or a Rules or Dates based exercise schedule.
`LegData`	At least one `LegData` sub-node
`Currency`	The same currency for all `LegData` sub-nodes.
`LegType`	Allowed types are *Cashflow*, *Fixed* or *Floating*. Floating coupons can be (capped / floored) Ibor, (capped / floored) compounded or averaged OIS, or BMA/SIFMA. Standalone options (nakedOption = true) are not allowed, neither are local OIS cap/floors.

Table 86: Requirements for Swaptions

The `OptionData` trade component sub-node is outlined in section 19.3.1. The relevant fields in the `OptionData` node for a Swaption are:

- `LongShort`: The allowable values are *Long* or *Short*. Note that for Swaptions the payer and receiver legs in the underlying swap are always from the perspective of the party that is *Long*. E.g. for a *Short* swaption with a fixed leg where the Payer flag is set to *false*, it means that the counterparty receives the fixed flows.

`LongShort`	`Payer` for Fixed leg on underlying Swap	`Payer` for Floating leg on underlying Swap	Resulting Set Up and Flows
Long	*true*	*false*	The Party to the trade buys an option to enter a swap where the Party pays fixed and receives floating
Short	*true*	*false*	The Party to the trade sells an option to the Counterparty to enter a swap where the Counterparty pays fixed and receives floating
Long	*false*	*true*	The Party to the trade buys an option to enter a swap where the Party receives fixed and pays floating
Short	*false*	*true*	The Party to the trade sells an option to the Counterparty to enter a swap where the Counterparty receives fixed and pays floating

Table 87: Swaption set up and resulting flows

Note that for CallableSwaps, contrary to the above, the payer and receiver legs in the

underlying swap are instead from the perspective of the client, like for a standalone Swap trade.

- `OptionType`[Optional]: This flag is optional for swaptions, and even if set, has no impact. Whether a swaption is a payer or receiver swaption is determined by the Payer flags on the legs of the underlying swap.
- `Style`: The exercise style of the Swaption. The allowable values are *European*, *Bermudan* or *American*.
- `NoticePeriod`[Optional]: The notice period defining the date (relative to the exercise date) on which the exercise decision has to be taken. If not given the notice period defaults to *0D*, i.e. the notice date is identical to the exercise date. Allowable values: A number followed by *D, W, M, or Y*
- `NoticeCalendar`[Optional]: The calendar used to compute the notice date from the exercise date. If not given defaults to the *NullCalendar* (no holidays, weekends are no holidays either). Allowable values: See Table 24 `Calendar`.
- `NoticeConvention`[Optional]: The roll convention used to compute the notice date from the exercise date. Defaults to *Unadjusted* if not given. Allowable values: See Table 21 Roll Convention.
- `Settlement`: Delivery Type. The allowable values are *Cash* or *Physical*. Note that for TradeType *CallableSwap* only *Physical* is allowed. Also note that *Cash* means the last Exercise date will be the Maturity date, whereas for *Physical* the Maturity date will be the End date of the underlying Swap.
- `SettlementMethod`[Optional]: Specifies the method to calculate the settlement amount for Swaptions and CallableSwaps. Allowable values: *PhysicalOTC*, *PhysicalCleared*, *CollateralizedCashPrice*, *ParYieldCurve*. Defaults to *ParYieldCurve* if Settlement is *Cash* and defaults to *PhysicalOTC* if Settlement is *Physical*.

  PhysicalOTC = OTC traded swaptions with physical settlement
  PhysicalCleared = Cleared swaptions with physical settlement
  CollateralizedCashPrice = Cash settled swaptions with settlement price calculation using zero coupon curve discounting
  ParYieldCurve = Cash settled swaptions with settlement price calculation using par yield discounting [2] [3]
- `MidCouponExercise` [Optional]: If *false*, the exercise-into underlying comprises all coupons with accrual start date greater or equal to notification date. I.e. one exercises into the next coupon, not the current one.
  If *true*, the exercise-into underlying comprises all coupons with accrual end date greater than the effective exercise date which is computed from the notification date by adding the notice period. The accrual paid for such coupons on exercise is calculated from the effective exercise date to the accrual end date (short coupon).

  Allowable values: *true*, *false*. If omitted, defaults to *false* for *European* and *Bermudan* swaptions and *true* for *American* swaptions.

[2] https://www.isda.org/book/2006-isda-definitions/
[3] https://www.isda.org/a/TlAEE/Supplement-No-58-to-ISDA-2006-Definitions.pdf

- `ExerciseFees`[Optional]: This node contains child elements of type `ExerciseFee`. Similar to a list of notionals (see 19.3.3) the fees can be given either
  - as a list where each entry corresponds to an exercise date and the last entry is used for all remaining exercise dates if there are more exercise dates than exercise fee entries, or
  - using the `startDate` attribute to specify a change in a fee from a certain day on (w.r.t. the exercise date schedule)

  Fees can either be given as an absolute amount or relative to the current notional of the period immediately following the exercise date using the `type` attribute together with specifiers `Absolute` resp. `Percentage`. If not given, the type defaults to `Absolute`. `Percentage` fees are expressed in decimal form, e.g. 0.05 is a fee of 5% of notional.

  If a fee is given as a positive number the option holder has to pay a corresponding amount if they exercise the option. If the fee is negative on the other hand, the option holder receives an amount on the option exercise.

  Only supported for Swaptions and Callable Swaps currently.
- `ExerciseFeeSettlementPeriod`[Optional]: The settlement lag for exercise fee payments. Defaults to 0D if not given. This lag is relative to the exercise date (as opposed to the notice date). Allowable values: A number followed by *D, W, M, or Y*
- `ExerciseFeeSettlementCalendar`[Optional]: The calendar used to compute the exercise fee settlement date from the exercise date. If not given defaults to the *NullCalendar* (no holidays, weekends are no holidays either). Allowable values: See Table 24 Calendar.
- `ExerciseFeeSettlementConvention`[Optional]: The roll convention used to compute the exercise fee settlement date from the exercise date. Defaults to *Unadjusted* if not given. Allowable values: See Table 21 Roll Convention.
- An `ExerciseDates` node where for *European* style swaptions exactly one `ExerciseDate` date element must be given, and for *American* style swaptions exactly two `ExerciseDate` date element must be given, defining the start and the end of the American exercise period. *Bermudan* style swaptions can have `ExerciseDate` elements given directly (at least two `ExerciseDate` elements must be given). See Listing 38
- *Bermudan* style swaptions can also have Rules or Dates based exercise dates using an `ExerciseSchedule` node instead of `ExerciseDates`. See Listings 39 and 40.

Listing 38: Bermudan Swaption ExerciseDate:s

```
<SwaptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <Style>Bermudan</Style>
        <Settlement>Physical</Settlement>
        <ExerciseDates>
          <ExerciseDate>2027-03-02</ExerciseDate>
          <ExerciseDate>2028-03-02</ExerciseDate>
          <ExerciseDate>2029-03-02</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
   ...
```

Listing 39: Bermudan Swaption Rules based

```
<SwaptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <Style>Bermudan</Style>
        <Settlement>Physical</Settlement>
        <ExerciseSchedule>
         <Rules>
          <StartDate>2027-03-02</StartDate>
          <EndDate>2029-03-02</EndDate>
          <Tenor>1Y</Tenor>
          <Calendar>US</Calendar>
          <Convention>MF</Convention>
         </Rules>
        </ExerciseSchedule>
        ...
    </OptionData>
   ...
```

Listing 40: Bermudan Swaption Dates based

```
<SwaptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <Style>Bermudan</Style>
        <Settlement>Physical</Settlement>
        <ExerciseSchedule>
         <Dates>
          <Calendar>NullCalendar</Calendar>
          <Convention>Unadjusted</Convention>
          <Dates>
           <Date>2027-03-02</Date>
           <Date>2028-03-02</Date>
           <Date>2029-03-02</Date>
          </Dates>
         </Dates>
        </ExerciseSchedule>
        ...
    </OptionData>
   ...
```

- `Premiums` [Optional]: Option premium node with amounts paid by the option buyer to the option seller.

  Allowable values: See section 19.3.2

- An `ExerciseData` [Optional] node where one `Date` element must be given, and one `Price` element can optionally also be given. See Listing 41

  This node marks the Swaption as exercised. If the `ExerciseData` node is omitted it is assumed the Swaption has not been exercised.

  The effective exercise date is the next `ExerciseDate` in the `ExerciseDates` node greater or equal the given `Date` in `ExerciseData`.

  For a cash-settled Swaption, the `Price` given in `ExerciseData` represents the cash settlement amount. It is paid according to the `PaymentData` node: If an explicit list of payment dates is given, the payment takes place on the next date following the effective exercise date. If the `PaymentData` is rules-based, the payment date is derived from the effective exercise date using the given calendar, lag and convention.

  If a Swaption is cash-settled and has an `ExerciseData` node with a `Date` but no `Price`, then the Swaption is considered exercised on the given date, but without a settlement amount being paid.

Listing 41: ExerciseData to mark a Swaption or CallableSwap as exercised

```
<ExerciseData>
  <Date>2023-09-03</Date>
  <Price>112000</Price>
</ExerciseData>
```

- A `PaymentData` [Optional] node can be added which defines dates or rules-based

settlement date(s) for cash-settled Swaptions. Note that if rules-based, only *Exercise* is allowed in the `RelativeTo` field for Swaptions. See `PaymentData` in 19.3.1

19.2.6 Callable Swap

The `CallableSwapData` node is the trade data container for the *CallableSwap* trade type. A Callable Swap is a swap that can be cancelled at predefined dates by one of the counterparties. A Callable Swap must have at least one leg, each leg described by a `LegData` trade component sub-node as described in section 19.3.3.

Unless MidCouponExercise is *true*, there must be at least one full coupon period after the exercise date for European Callable Swaps, and after the last exercise date for Bermudan and American Callable Swaps.

The `CallableSwapData` node also contains an `OptionData` node which describes the exercise dates and specifies which party holds the call right, see 19.3.1. An example structure of a `CallableSwapData` node is shown in Listing 42.

Listing 42: Callable Swap data

```
<CallableSwapData>
    <OptionData>
      <LongShort>Short</LongShort>
      <Style>Bermudan</Style>
      <Settlement>Physical</Settlement>
      <MidCouponExercise>true<MidCouponExercise>
      <ExerciseDates>
        <ExerciseDate>2031-10-01</ExerciseDate>
        <ExerciseDate>2032-10-01</ExerciseDate>
        <ExerciseDate>2033-10-01</ExerciseDate>
      </ExerciseDates>
      ...
    </OptionData>
  <LegData>
        <LegType>Fixed</LegType>
        <Payer>false</Payer>
        <Currency>USD</Currency>
        ...
  </LegData>
  <LegData>
        <LegType>Floating</LegType>
        <Payer>true</Payer>
        <Currency>USD</Currency>
        ...
  </LegData>
</CallableSwapData>
```

The meanings and allowable values of the elements in the `CallableSwapData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The exercise dates specify the dates on which one of the counterparties may terminate the swap. The counterpart holding the call right is specified by the `LongShort` flag. The Settlement should be set to *Physical* always. See also the OptionData node outlined for

a Swaption - see 19.2.5, which is identical for a CallableSwap with the exception of the requirement that Settlement must be *Physical*, and that the leg directions on a CallableSwap are from the perspective of the client, whereas they are from the perspective of the party that is long on a Swaption. A callable swap can be marked as exercised as explained in 19.2.5 using the `ExerciseData` node within OptionData.

- LegData: This is a trade component sub-node described in section 19.3.3 outlining each leg of the underlying Swap. A Callable Swap must have at least one leg on the underlying Swap, but can have multiple legs, i.e. multiple `LegData` nodes. The LegType elements must be of types *Floating*, *Fixed* or *Cashflow*. All legs must have the same `Currency`.

  Note that the direction of the legs, determined by the `Payer` tag, is like for a Swap, from the perspective of the party to the trade. I.e. unlike for a Swaption where the direction of the legs is from the perspective of the party that is long.

19.2.7 Cash Position

The `CashPositionData` node is the trade data container for the *CashPosition* trade type. The structure - including example values - of the `CashPositionData` node is shown in Listing 43.

A cash position can be used both as a stand alone trade type (TradeType: *CashPosition*) or as a trade component within the *TotalReturnSwap* (Generic TRS) trade type.

Listing 43: Cash Position data

```
<CashPositionData>
    <Currency>EUR</Currency>
    <Amount>1000000</Amount>
</CashPositionData>
```

The meanings and allowable values of the various elements in the `CashPositionData` node follow below.

- Currency: The currency of cash position.
  Allowable values: See Table 22 `Currency`.
- Amount: The amount of cash position.
  Allowable values: Any real number.

19.2.8 FX Forward

The `FXForwardData` node is the trade data container for the *FxForward* trade type. The structure - including example values - of the `FXForwardData` node is shown in Listing 44.

Listing 44: FX Forward data

```
<FxForwardData>
    <ValueDate>2023-04-09</ValueDate>
    <BoughtCurrency>EUR</BoughtCurrency>
    <BoughtAmount>1000000</BoughtAmount>
    <SoldCurrency>USD</SoldCurrency>
    <SoldAmount>1500000</SoldAmount>
    <Settlement>Physical</Settlement>
    <SettlementData>
      ...
    </SettlementData>
</FxForwardData>
```

The meanings and allowable values of the various elements in the `FXForwardData` node follow below.

- ValueDate: The value date of the FX Forward.
  Allowable values: See `Date` in Table 20.
- BoughtCurrency: The currency to be bought on value date.
  Allowable values: See Table 22 `Currency`.
- BoughtAmount: The amount to be bought on value date.
  Allowable values: Any positive real number.
- SoldCurrency: The currency to be sold on value date.
  Allowable values: See Table 22 `Currency`.
- SoldAmount: The amount to be sold on value date.
  Allowable values: Any positive real number.
- Settlement [Optional]: Delivery type. Note that Non-Deliverable Forwards can be represented by *Cash* settlement.
  Allowable values: *Cash* or *Physical*. Defaults to *Physical* if left blank or omitted.
- SettlementData [Optional]: This node is used to specify the settlement of the cash flows on the value date.

A `SettlementData` node is shown in Listing 45, and the meanings and allowable values of its elements follow below.

- Currency: The currency in which the FX Forward is settled. This field is only used if settlement is *Cash*.
  Allowable values: See Table 22 `Currency`. Defaults to the sold currency if left blank or omitted.
- FXIndex: The FX reference index for determining the FX fixing at the value date. This field is required if settlement is *Cash* and the payment date is greater than the value date. Otherwise, it is ignored.
  Allowable values: The format of the `FXIndex` is "FX-FixingSource-CCY1-CCY2" as described in Table 28.
- Date [Optional]: If specified, this will be the payment date.

Allowable values: See `Date` in Table 20. If left blank or omitted, defaults to the value date with some adjustments applied from the `Rules` sub-node.

- Rules [Optional]: If `Date` is left blank or omitted, this node will be used to derive the payment date from the value date. The `Rules` sub-node is shown in Listing 45, and the meanings and allowable values of its elements follow below.
  - PaymentLag [Optional]: The lag between the value date and the payment date.
    Allowable values: Any valid period, i.e. a non-negative whole number, optionally followed by *D* (days), *W* (weeks), *M* (months), *Y* (years). For cash settlement and if a FXIndex is specified defaults to the fx convention (field "SpotDays") if blank or omitted, otherwise to 0. If a whole number is given and no letter, it is assumed that it is a number of *D* (days).
  - PaymentCalendar [Optional]: The calendar to be used when applying the payment lag.
    Allowable values: See Table 24 `Calendar`. For cash settlement and if a FXIndex is specified defaults to the fx convention (field "AdvanceCalendar") if left blank or omitted, otherwise to NullCalendar (no holidays).
  - PaymentConvention [Optional]: The roll convention to be used when applying the payment lag.
    Allowable values: See Table 21 Roll Convention. For cash settlement and if a FXIndex is specified defaults to the fx convention ((field "Convention") if left blank or omitted, otherwise to Unadjusted.

Note that FX Forwards also cover Precious Metals forwards, i.e. with currencies XAU, XAG, XPT, XPD, and Cryptocurrency forwards, see supported Cryptocurrencies in Table 22.

Listing 45: Example `SettlementData` *node with* `Rules` *sub-node*

```
<SettlementData>
  <Currency>USD</Currency>
  <FXIndex>FX-ECB-EUR-USD</FXIndex>
  <Date>2020-09-03</Date>
  <Rules>
    <PaymentLag>2D</PaymentLag>
    <PaymentCalendar>USD</PaymentCalendar>
    <PaymentConvention>Following</PaymentConvention>
  </Rules>
</SettlementData>
```

19.2.9 FX Average Forward

The `FXAverageForwardData` node is the trade data container for the *FxAverageForward* trade type. The structure with example values node is shown in Listing 46.

Listing 46: FX Average Forward data

```
<FxAverageForwardData>
    <PaymentDate>2023-04-09</PaymentDate>
    <!-- Schedule block that determines observation dates for FX averaging -->
    <ObservationDates>
       ...
    </ObservationDates>
    <FixedPayer>true</FixedPayer>
    <ReferenceNotional>8614</ReferenceNotional>
    <ReferenceCurrency>EUR</ReferenceCurrency>
    <SettlementNotional>10000</SettlementNotional>
    <SettlementCurrency>USD</SettlementCurrency>
    <FXIndex>FX-ECB-EUR-USD</FXIndex>
    <Settlement>Cash</Settlement>
</FxAverageForwardData>
```

The instrument's payoff is driven by an arithmetic average of observed FX rates, expressed in terms of the node names:

$$\omega \times (\text{ReferenceNotional} \times \text{AverageFX} - \text{SettlementNotional})$$

The meanings and allowable values of the various elements in the `FXAverageForwardData` node follow below.

- PaymentDate: The date of the settlement cash flow.
  Allowable values: See `Date` in Table 20.
- ObservationDates: Schedule data that determine the observation dates that are taken into account in the FX rate averaging. See section 19.3.4
- FixedPayer: If *true*, the payoff multiplier ω is set to 1, otherwise -1.
  Allowable values: *true*, *false*
- ReferenceNotional: The amount to be converted into settlement currency at the average FX rate
  Allowable values: Any positive real number.
- ReferenceCurrency: The currency of the reference notional above.
  Allowable values: See Table 22 `Currency`.
- SettlementNotional: The fixed amount to be paid or received depending on the fixed payer flag above
  Allowable values: Any positive real number.
- SettlementCurrency: The currency of the settlement notional above.
  Allowable values: See Table 22 `Currency`.
- FXIndex: The FX reference index for determining the FX fixing for averaging.
  Allowable values: The format of the `FXIndex` is "FX-FixingSource-CCY1-CCY2" as described in Table 28. Notice that since the payoff is based on an arithmetic average, the order of the currencies in the FX index matters: The averaging will be done on fx rates quoted as CCY1-CCY2 (foreign-domestic).

19.2.10 FX Swap

The `FXSwapData` node is the trade data container for the *FxSwap* trade type. The structure - including example values - of the `FXSwapData` node is shown in Listing 47. It contains no sub-nodes.

Listing 47: FX Swap data

```
<FxSwapData>
    <NearDate>2018-09-01</NearDate>
    <NearBoughtCurrency>EUR</NearBoughtCurrency>
    <NearBoughtAmount>1000000</NearBoughtAmount>
    <NearSoldCurrency>USD</NearSoldCurrency>
    <NearSoldAmount>1140000</NearSoldAmount>
    <FarDate>2028-09-01</FarDate>
    <FarBoughtAmount>1300000</FarBoughtAmount>
    <FarSoldAmount>1000000</FarSoldAmount>
    <Settlement>Cash</Settlement>
</FxSwapData>
```

The meanings and allowable values of the various elements in the `FXSwapData` node follow below. All elements are required.

- NearDate: The date of the initial fx exchange of the FX Swap.
  Allowable values: See `Date` in Table 20.
- NearBoughtCurrency: The currency to be bought in the initial exchange at near date, and sold in the final exchange at far date.
  Allowable values: See Table 22 `Currency`.
- NearBoughtAmount: The amount to be bought on near date.
  Allowable values: Any positive real number.
- NearSoldCurrency: The currency to be sold in the initial fx exchange at near date, and bought in the final exchange at far date.
  Allowable values: See Table 22 `Currency`.
- NearSoldAmount: The amount to be sold on near date.
  Allowable values: Any positive real number.
- FarDate: The date of the final fx exchange of the FX Swap.
  Allowable values: Any date further into the future than NearDate. See `Date` in Table 20.
- FarBoughtAmount: The amount to be bought on far date.
  Allowable values: Any positive real number.
- FarSoldAmount: The amount to be sold on far date.
  Allowable values: Any positive real number.
- Settlement [Optional]: Delivery type. Note that Non-Deliverable FX Swaps can be represented by *Cash* settlement, and that deliverable FX Swaps will be excluded from the CRIF output. Delivery type does not impact pricing in VRE.

  Allowable values: *Cash* or *Physical*. Defaults to *Physical* if left blank or omitted.

Note that FX Swaps also cover Precious Metals swaps, i.e. with currencies XAU, XAG, XPT, XPD, and Cryptocurrency swaps, see supported Cryptocurrencies in Table 22.

19.2.11 FX Option

The `FXOptionData` node is the trade data container for the *FxOption* trade type. FX options with exercise styles *European* or *American* are supported. The `FXOptionData` node includes one and only one `OptionData` trade component sub-node plus elements specific to the FX Option. The structure of an `FXOptionData` node for an FX Option is shown in Listing 48.

Listing 48: FX Option data

```
<FxOptionData>
  <OptionData>
    <LongShort>Long</LongShort>
    <OptionType>Call</OptionType>
    <Style>European</Style>
    <Settlement>Cash</Settlement>
    <PayOffAtExpiry>false</PayOffAtExpiry>
    <ExerciseDates>
       <ExerciseDate>2026-03-01</ExerciseDate>
     </ExerciseDates>
     <Premiums>
       <Premium>
         <Amount>10900</Amount>
         <Currency>EUR</Currency>
         <PayDate>2020-03-01</PayDate>
       </Premium>
     </Premiums>
  </OptionData>
  <BoughtCurrency>EUR</BoughtCurrency>
  <BoughtAmount>1000000</BoughtAmount>
  <SoldCurrency>USD</SoldCurrency>
  <SoldAmount>1700000</SoldAmount>
</FxOptionData>
```

The meanings and allowable values of the elements in the `FXOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the `OptionData` node for an FxOption are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*. For option type *Put*, Bought and Sold currencies/amounts are switched compared to the trade data node. For example, a holder of BoughtCurrency EUR SoldCurrency USD FX Call Option has the right to buy EUR using USD, while holder of the Put counterpart has the right to buy USD using EUR, or equivalently sell EUR for USD.
  - `Style` The allowable values are *European* or *American*.
  - `Settlement` The allowable values are *Cash* or *Physical*.
  - `PayOffAtExpiry` [Optional] The allowable values are *true* for payoff at expiry, or *false* for payoff at exercise (relevant for *American* style FxOptions). Defaults to

true if left blank or omitted.

- `AutomaticExercise` [Optional] The allowable values are *true* indicating Automatic Exercise is applicable and *false* indicates that it is not. Used if the FXOption expiry date is on the current date or in the past, and the payment date is in the future - so that there still is an outstanding cashflow if the FXOption was in the money on the expiry date. In this case, if AutomaticExercise is applied, the FX fixing on the expiry date is used to automatically determine the payoff and thus whether the option was exercised or not. Defaults to *false* if left blank or omitted.
- An `ExerciseDates` node where exactly one ExerciseDate date element must be given. For *American* style FxOptions the ExerciseDate represents the Expiry date, i.e. they can be exercised up until this date.
- A `PaymentData` [Optional] node can be added which defines the settlement date of the option payoff. See `PaymentData` in 19.3.1
- `Premiums` [Optional]: Option premium amounts paid by the option buyer to the option seller. See section 19.3.2

See 19.3.1 for further specifications of the `OptionData` node.

- BoughtCurrency: The bought currency of the FX option. See OptionData above for more details.

  Allowable values: See Table 22.

- BoughtAmount: The amount in the BoughtCurrency.

  Allowable values: Any positive real number.

- SoldCurrency: The sold currency of the FX option. See OptionData above for more details.

  Allowable values: See Table 22.

- SoldAmount [Optional]: The amount in the SoldCurrency. Note that if Delta is omitted, the SoldAmount field is mandatory.

  Allowable values: Any positive real number.

- Delta [Optional]: The FX option delta. When a delta value is given the FX Option strike is derived from the delta, and the SoldAmount is ignored.

  Allowable values: Any non null real number. A SoldAmount or a Delta field is required, as the strike is derived from one or the other. Note: The delta to strike conversion is based on the valuation date. Therefore the strike will change day to day based on the market data variation. It is not possible to enter a seasoned trade with a Delta such that the trade strike (SoldAmount) is derived from the Delta on the trade date and then kept constant throughout the life of the trade.

- FXIndex [Optional]: If the option *European*, has cash settlement and is subject to *Automatic Exercise*, as indicated by the `AutomaticExercise` node under `OptionData`, this node must be populated with a valid FX index. The FX index is used to retrieve an FX rate on the expiry date that is in turn used to determine the payoff on the cash settlement date. The payoff is in the `SoldCurrency` i.e. the domestic currency.

Allowable values: A valid FX index from the Table 28.

Note that FX Options also cover Precious Metals Options, i.e. with currencies XAU, XAG, XPT, XPD, and Cryptocurrency options, see supported Cryptocurrencies in Table 22.

19.2.12 FX Asian Option

The `FxAsianOptionData` node is the trade data container for the *FxAsianOption* trade type. The `FxAsianOptionData` node includes one `OptionData` trade component sub-node plus elements specific to the FX Asian Option.

A FX Asian Option is a path-dependent option whose payoff depends upon the averaged foreign exchange rate over a pre-set period of time.

The structure of an example `FxAsianOptionData` node for a FX Asian Option is shown in Listing 49.

Listing 49: FX Asian Option data

```
<Trade id="FxAsianOption">
        <TradeType>FxAsianOption</TradeType>
        <Envelope>
                <CounterParty>CPTY_A</CounterParty>
                <NettingSetId>CPTY_A</NettingSetId>
                <AdditionalFields />
        </Envelope>
        <FxAsianOptionData>
                <Currency>USD</Currency>
                <Quantity>100</Quantity>
                <Strike>1.05</Strike>
                <Underlying>
                        <Type>FX</Type>
                        <Name>ECB-EUR-USD</Name>
                </Underlying>
                <OptionData>
                        <LongShort>Long</LongShort>
                        <OptionType>Call</OptionType>
                        <PayoffType>Asian</PayoffType>
                        <PayoffType2>Arithmetic</PayoffType2>
                        <ExerciseDates>
                                <ExerciseDate>2020-07-15</ExerciseDate>
                        </ExerciseDates>
                </OptionData>
                <Settlement>2020-07-20</Settlement>
                <ObservationDates>
                        <Rules>
                                <StartDate>2019-12-27</StartDate>
                                <EndDate>2020-07-06</EndDate>
                                <Tenor>1D</Tenor>
                                <Calendar>US</Calendar>
                                <Convention>F</Convention>
                                <TermConvention>F</TermConvention>
                                <Rule>Forward</Rule>
                        </Rules>
                </ObservationDates>
        </FxAsianOptionData>
</Trade>
```

The meanings and allowable values of the elements in the `FxAsianOptionData` node follow below.

- Currency: The payoff currency.
  Allowable values: See Table 22 `Currency`.
- Quantity: The quantity of the underlying currency (CCY1). See payoff formula above.
  Allowable values: all positive real numbers
- Strike: The strike of the option, expressed as amount of CCY2 per one unit of CCY1.
  Allowable values: all positive real numbers
- Underlying: An `Underlying` node where `Type` must be set to *FX* and `Name` is the foreign exchange currency pair (on the form SOURCE-CCY1-CCY2) including the `Currency` above typically as CCY2 and another currency defined as the underlying currency as CCY1.
  Allowable values: See 19.3.29
- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the `OptionData` node for an FxAsianOption are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*.
  - `PayoffType` which must be set to *Asian* or *AverageStrike* to identify a fixed or floating strike asian payoff,
  - `PayoffType2` [Optional] can be optionally set to *Arithmetic* or *Geometric* and defaults to *Arithmetic* if not given.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - A `PaymentData` [Optional] node can be added which defines the settlement of the option payoff.
  - A `Premiums` [Optional] node can be added to represent deterministic option premia to be paid by the option holder. See section 19.3.2
- Settlement[Optional]: The settlement date.
  Allowable values: See `Date` in Table 20. Defaults to the ExerciseDate if left blank or omitted.
- ObservationDates: The observation dates for the asian period, given as a rules-based or dates-based schedule, analogous to a `ScheduleData` node but called `ObservationDates`.
  Allowable values: See the definition in 19.3.4

19.2.13 FX Barrier Option

European exercise, American barrier.

An FX Barrier option is a path-dependent option whose existence depends upon an FX spot rate reaching a pre-set barrier level. Exercise is European.

This product has a continuously monitored single barrier (American Barrier style) with a Vanilla European FX Option Underlying.

The `FxBarrierOptionData` node is the trade data container for the *FxBarrierOption* trade type. The barrier level of an FX Barrier Option is quoted as the amount in SoldCurrency per unit BoughtCurrency. The `FxBarrierOptionData` node includes one `OptionData` trade component sub-node and one `BarrierData` trade component sub-node plus elements specific to the FX Barrier Option.

The structure of an example `FxBarrierOptionData` node for a FX Barrier Option is shown in Listing 50.

Listing 50: FX Barrier Option data

```
<FxBarrierOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <!-- Bought and Sold currencies/amounts are switched for Put -->
        <OptionType>Call</OptionType>
        <Style>European</Style>
        <Settlement>Cash</Settlement>
        <ExerciseDates>
         <ExerciseDate>2021-12-14</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <BarrierData>
     <Type>UpAndIn</Type>
     <Levels>
        <Level>1.2</Level>
     </Levels>
     <Rebate>0.0</Rebate>
    </BarrierData>
    <StartDate>2019-01-25</StartDate>
    <Calendar>TARGET</Calendar>
    <FXIndex>FX-ECB-EUR-USD</FXIndex>
    <BoughtCurrency>EUR</BoughtCurrency>
    <BoughtAmount>1000000</BoughtAmount>
    <SoldCurrency>USD</SoldCurrency>
    <SoldAmount>1100000</SoldAmount>
</FxBarrierOptionData>
```

The meanings and allowable values of the elements in the `FxBarrierOptionData` node follow below.

- `OptionData`: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the `OptionData` node for an FxBarrierOption are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*.
    Call means that the holder of the option, upon expiry - assuming knock-in or no knock-out - has the right to receive the BoughtAmount and pay the SoldAmount.
    Put means that the Bought and Sold currencies/amounts are switched compared to the trade data node. For example, holder of BoughtCurrency EUR

SoldCurrency JPY FX Barrier Call Option has the right to buy EUR using JPY, while holder of the Put counterpart has the right to buy JPY using EUR, or equivalently sell EUR for JPY. An alternative to define the latter option is to copy the Call option with following changes:
a) swapping BoughtCurrency with SoldCurrency, b) swapping BoughtAmount with SoldAmount and c) inverting the barrier level (for example changing 110 to 0.0090909). Here barrier level is quoted as amount of EUR per unit JPY, which is not commonly seen on market and inconsistent with the format in Call options. For these reasons, using Put/Call flag instead is recommended.

  - `Style` The FX Barrier Option type allows for *European* option exercise style only.
  - `Settlement` The allowable values are *Cash* or *Physical*.
  - A `PaymentData` [Optional] node can be added which defines the settlement of the option payoff.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - `Premiums` [Optional]: Option premium amounts paid by the option buyer to the option seller. See section 19.3.2

- `BarrierData`: This is a trade component sub-node outlined in section 19.3.31. `Level` specified in BarrierData should be quoted as the amount in SoldCurrency per unit BoughtCurrency, with both currencies as defined in FxBarrierOptionData node. Note that the barrier `Level` stays quoted as SoldCurrency per unit BoughtCurrency, regardless of Put/Call.
- `StartDate` [Optional]: The start date for checking if a barrier has been breached prior to today's date. If omitted or left blank no check is made and it is assumed no barrier has been breached in the past. Has no impact if set to today's date or a date in the future.

  Allowable values: See `Date` in Table 20.
- `Calendar` [Optional]: The calendar associated with the FX Index. Required if StartDate is set to a date prior to today's date, otherwise optional.

  Allowable values: See Table 24 Calendar.
- `FXIndex` [Optional]: A reference to an FX Index source to check if the barrier has been breached. Required if StartDate is set to a date prior to today's date, otherwise optional and can be omitted but not left blank.

  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.
- `FXIndexDailyLows` [Optional]: Refers to an FX Index that tracks the daily low quotes. This is used to check if the barrier was breached at any point during the day. If not provided, VRE will automatically derive the index name by appending the suffix *_LOW* to the FXIndex source (e.g. *FX-SOURCE_LOW-CCY1-CCY2*). If no fixings are available, the system will fall back to using the fixings from the FXIndex.

  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.

- `FXIndexDailyHighs` [Optional]: Refers to an FX Index that tracks the daily high quotes. This is used to check if the barrier was breached at any point during the day. If not provided, VRE will automatically derive the index name by appending the suffix *_HIGH* to the FXIndex source (e.g. *FX-SOURCE_HIGH-CCY1-CCY2*). If no fixings are available, the system will fall back to using the fixings from the FXIndex.

  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.

- `BoughtCurrency`: The bought currency of the FX barrier option. See OptionData above for more details.

  Allowable values: See Table 22 `Currency`.

- `BoughtAmount`: The amount in the BoughtCurrency.

  Allowable values: Any positive real number.

- `SoldCurrency`: The sold currency of the FX barrier option. See OptionData above for more details.

  Allowable values: See Table 22 `Currency`.

- `SoldAmount`: The amount in the SoldCurrency.

  Allowable values: Any positive real number.

Note that FX Barrier Options also cover Precious Metals, i.e. with currencies XAU, XAG, XPT, XPD, and Cryptocurrencies, see supported Cryptocurrencies in Table 22.

19.2.14 FX Digital Barrier Option

An FX Digital Barrier Option pays a given cash amount in domestic currency at expiry, if the underlying fx rate has hit (or not hit) a continuously monitored barrier (as for the FxTouchOption) and the fx rate at the expiry date is above (call) or below (put) a given strike.

The `FxDigitalBarrierOptionData` node is the trade data container for the *FxDigitalBarrierOption* trade type. The `FxDigitalBarrierOptionData` node includes one `OptionData` trade component sub-node and one `BarrierData` trade component sub-node plus elements specific to the FX Digital Barrier Option.

The structure of an example `FxDigitalBarrierOptionData` node for a FX Digital Barrier Option is shown in Listing 51.

Listing 51: FX Digital Barrier Option data

```
<FxDigitalBarrierOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <OptionType>Call</OptionType>
        <Style>European</Style>
        <ExerciseDates>
         <ExerciseDate>2021-12-14</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <BarrierData>
        <Type>DownAndIn</Type>
        <Levels>
            <Level>1.18</Level>
        </Levels>
    </BarrierData>
    <StartDate>2019-01-25</StartDate>
    <Calendar>TARGET</Calendar>
    <FXIndex>FX-ECB-EUR-USD</FXIndex>
    <Strike>1.1</Strike>
    <PayoffAmount>100000</PayoffAmount>
    <PayoffCurrency>USD</PayoffCurrency>
    <ForeignCurrency>EUR</ForeignCurrency>
    <DomesticCurrency>USD</DomesticCurrency>
</FxDigitalBarrierOptionData>
```

The meanings and allowable values of the elements in the `FxDigitalBarrierOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the `OptionData` node for an FxDigitalBarrierOption are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*. Given knock-in or no knock-out, *Call* means that the digital payout will occur if the fx rate at the expiry date is above the given strike, and *Put* means that the digital payout will occur if the fx rate at the expiry date is below the given strike.
  - `Style` The FX Digital Barrier Option type allows for *European* option exercise style only.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - `Premiums` [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2
- BarrierData: This is a trade component sub-node outlined in section 19.3.31.

  Note that the *FxDigitalBarrierOption* is a single barrier instrument, and can have only one BarrierData node with one barrier level.

Level specified in BarrierData should be quoted as the amount in DomesticCurrency per one unit of ForeignCurrency, with both currencies as defined in FxDigitalBarrierOptionData node.

Type specified in BarrierData can be one of: *UpAndIn, DownAndIn, UpAndOut, DownAndOut*

- StartDate[Optional]: The start date for checking if a barrier has been breached prior to today's date. If omitted or left blank no check is made and it is assumed no barrier has been breached in the past. Has no impact if set to today's date or a date in the future. If'StartDate' is provided then the fixings for dates between this date and the asof date are checked to see if the option was triggered. If no fixing is available then we skip that date. This is to allow for backwards compatibility.

  Allowable values: See `Date` in Table 20.

- Calendar[Optional]: The calendar associated with the FX Index. Required if StartDate is set to a date prior to today's date, otherwise optional.

  Allowable values: See Table 24 Calendar.

- FXIndex[Optional]: A reference to an FX Index source to check if the barrier has been breached. Required if StartDate is set to a date prior to today's date, otherwise optional, and can be omitted but not left blank.

  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.

- `FXIndexDailyLows` [Optional]: Refers to an FX Index that tracks the daily low quotes. This is used to check if the barrier was breached at any point during the day. If not provided, VRE will automatically derive the index name by appending the suffix *_LOW* to the FXIndex source (e.g. *FX-SOURCE_LOW-CCY1-CCY2*). If no fixings are available, the system will fall back to using the fixings from the FXIndex.

  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.

- `FXIndexDailyHighs` [Optional]: Refers to an FX Index that tracks the daily high quotes. This is used to check if the barrier was breached at any point during the day. If not provided, VRE will automatically derive the index name by appending the suffix *_HIGH* to the FXIndex source (e.g. *FX-SOURCE_HIGH-CCY1-CCY2*). If no fixings are available, the system will fall back to using the fixings from the FXIndex.

  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.

- Strike: The FX strike price, expressed as the amount in DomesticCurrency per one unit of ForeignCurrency.

  Allowable values: Any positive real number.

- PayoffAmount: The fixed payoff amount expressed in the PayoffCurrency. It is cash-or-nothing payoff that depends on the option being in or out of the money, and whether the barrier has been breached.

  Allowable values: Any positive real number.

- PayoffCurrency[Optional]: The payoff currency of the FX digital option is the currency of the payoff amount. Must be either the Domestic or Foreign currency for this trade, If omitted this defaults to DomesticCurrency as defined in FxDigitalBarrierOptionData node.

  Allowable values: See Table 22 `Currency`.

- ForeignCurrency: The foreign currency of the FX digital barrier option is equivalent to the bought currency.

  Allowable values: See Table 22 `Currency`.

- DomesticCurrency: The domestic currency of the FX digital barrier option is equivalent to the sold currency.

  Allowable values: See Table 22 `Currency`.

Note that FX Digital Barrier Options also cover Precious Metals, i.e. with currencies XAU, XAG, XPT, XPD, and Cryptocurrencies, see supported Cryptocurrencies in Table 22.

19.2.15 FX Digital Option

The `FxDigitalOptionData` node is the trade data container for the *FxDigitalOption* trade type. The `FxDigitalOptionData` node includes one `OptionData` trade component sub-node plus elements specific to the FX Digital Option. The structure of an example `FxDigitalOptionData` node for a FX Digital Option is shown in Listing 52.

Listing 52: FX Digital Option data

```
<FxDigitalOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <OptionType>Call</OptionType>
        <Style>European</Style>
        <ExerciseDates>
         <ExerciseDate>2021-12-14</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <Strike>1.1</Strike>
    <PayoffCurrency>USD</PayoffCurrency>
    <PayoffAmount>100000</PayoffAmount>
    <ForeignCurrency>EUR</ForeignCurrency>
    <DomesticCurrency>USD</DomesticCurrency>
</FxDigitalOptionData>
```

The meanings and allowable values of the elements in the `FxDigitalOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the `OptionData` node for an FxDigitalOption are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*. *Call* means that the digital payout will occur if the fx rate at the expiry date is above the given strike, and *Put*

means that the digital payout will occur if the fx rate at the expiry date is below the given strike.

  - `Style` The FX Digital Option type allows for *European* option exercise style only.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - `Premiums` [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2

- Strike: The FX strike price, expressed as the amount in DomesticCurrency per one unit of ForeignCurrency.

  Allowable values: Any positive real number.

- PayoffCurrency[Optional]: The payoff currency of the FX digital option is the currency of the payoff amount. Must be either the Domestic or Foreign currency for this trade, If omitted this defaults to the domestic currency.

  Allowable values: See Table 22 `Currency`.

- PayoffAmount: The fixed payoff amount expressed in payoff currency. It is cash-or-nothing payoff that depends on the option being in or out of the money.

  Allowable values: Any positive real number.

- ForeignCurrency: The foreign currency of the FX digital option is equivalent to the bought currency.

  Allowable values: See Table 22 `Currency`.

- DomesticCurrency: The domestic currency of the FX digital option is equivalent to the sold currency.

  Allowable values: See Table 22 `Currency`.

Note that FX Digital Options also cover Precious Metals, i.e. with currencies XAU, XAG, XPT, XPD, and Cryptocurrencies, see supported Cryptocurrencies in Table 22.

19.2.16 FX Double Barrier Option

An FX Double Barrier Option is a path-dependent option whose existence depends upon an FX spot rate reaching one of the two pre-set barrier levels. Exercise is European, and barriers are American (continuously monitored).

FX Double Barrier options can be knock-in or knock-out:

- A knock-in option is a barrier option that only comes into existence/becomes active when the FX spot rate reaches the one of the barrier level at any point in the option's life. Once a barrier is knocked-in, the option will not cease to exist until the option expires and effectively it becomes a Vanilla FX Option.
- A knock-out option starts its life active, but ceases to exist/becomes inactive, if the one of the barriers is reached during the life of the option.

The FxDoubleBarrierOptionData node is the trade data container for the *FxDoubleBarrierOption* trade type.

The barrier levels of an FX Double Barrier Option are quoted as the amount in SoldCurrency per unit BoughtCurrency. The FxDoubleBarrierOptionData node includes one OptionData trade component sub-node and one BarrierData trade component sub-node plus elements specific to the FX Double Barrier Option. The structure of an example FxDoubleBarrierOptionData node for a FX Double Barrier Option is shown in Listing 53.

Listing 53: FX Double Barrier Option data

```
<FxDoubleBarrierOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <!-- Bought and Sold currencies/amounts are switched for Put -->
        <OptionType>Call</OptionType>
        <Style>European</Style>
        <ExerciseDates>
         <ExerciseDate>2021-12-14</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <BarrierData>
        <Type>KnockOut</Type> <!-- KnockOut or KnockIn -->
        <Levels>
            <Level>1.1</Level>
            <Level>1.2</Level>
        </Levels>
        <Rebate>0.0</Rebate>
    </BarrierData>
    <StartDate>2019-01-25</StartDate>
    <Calendar>TARGET</Calendar>
    <FXIndex>FX-ECB-EUR-USD</FXIndex>
    <BoughtCurrency>EUR</BoughtCurrency>
    <BoughtAmount>1000000</BoughtAmount>
    <SoldCurrency>USD</SoldCurrency>
    <SoldAmount>1100000</SoldAmount>
</FxDoubleBarrierOptionData>
```

The meanings and allowable values of the elements in the FxDoubleBarrierOptionData node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the OptionData node for an FxDoubleBarrierOption are:
  - LongShort The allowable values are *Long* or *Short*.
  - OptionType The allowable values are *Call* or *Put*.
    Call means that the holder of the option, upon expiry - assuming knock-in or no knock-out - has the right to receive the BoughtAmount and pay the SoldAmount.
    Put means that the Bought and Sold currencies/amounts are switched compared to the trade data node. For example, holder of BoughtCurrency EUR SoldCurrency JPY FX Double Barrier Call Option has the right to buy EUR using JPY, while holder of the Put counterpart has the right to buy JPY using

EUR, or equivalently sell EUR for JPY. An alternative to define the latter option is to copy the Call option with following changes:
a) swapping BoughtCurrency with SoldCurrency, b) swapping BoughtAmount with SoldAmount and c) inverting the barrier level (for example changing 110 to 0.0090909). Here barrier level is quoted as amount of EUR per unit JPY, which is not commonly seen on market and inconsistent with the format in Call options. For these reasons, using Put/Call flag instead is recommended.

  - `Style` The FX Double Barrier Option type allows for *European* option exercise style only.
  - `Settlement` The allowable values are *Cash* or *Physical*.
  - A `PaymentData` [Optional] node can be added which defines the settlement of the option payoff.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - `Premiums` [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2

- `BarrierData`: This is a trade component sub-node outlined in section 19.3.31. Levels specified in BarrierData should be quoted as the amount in SoldCurrnecy per unit BoughtCurrency, with both currencies as defined in FxDoubleBarrierOptionData node. Changing the option from Call to Put or vice versa does not require switching the barrier levels. Two levels in ascending order should be defined in `Levels`. `Type` should be *KnockOut* or *KnockIn*.
- `StartDate` [Optional]: The start date for checking if a barrier has been breached prior to today's date. If omitted or left blank no check is made and it is assumed no barrier has been breached in the past. Has no impact if set to today's date or a date in the future.

  Allowable values: See `Date` in Table 20.
- `Calendar` [Optional]: The calendar associated with the FX Index. Required if StartDate is set to a date prior to today's date, otherwise optional.

  Allowable values: See Table 24 Calendar.
- `FXIndex` [Optional]: A reference to an FX Index source to check if the barrier has been breached. Required if StartDate is set to a date prior to today's date, otherwise optional and can be omitted but not left blank.

  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.
- `BoughtCurrency`: The bought currency of the FX barrier option. See OptionData above for more details.

  Allowable values: See Table 22 `Currency`.
- `BoughtAmount`: The amount in the BoughtCurrency.

  Allowable values: Any positive real number.

- `SoldCurrency`: The sold currency of the FX barrier option. See OptionData above for more details.

  Allowable values: See Table 22 `Currency`.

- `SoldAmount`: The amount in the SoldCurrency.

  Allowable values: Any positive real number.

19.2.17 FX Double Touch Option

An FX Double Touch Option pays a given cash amount (PayoffAmount) at expiry or at hit if the underlying fx rate has hit either of the barriers (KnockIn) resp. has not hit any of barriers (KnockOut) using continuous monitoring between start and expiry date. No rebates are supported.

The `FxDoubleTouchOptionData` node is the trade data container for the *FxDoubleTouchOption* trade type. The `FxDoubleTouchOptionData` node includes one `OptionData` trade component sub-node and one `BarrierData` trade component sub-node plus elements specific to the FX Double Touch Option.

The structure of an example `FxDoubleTouchOptionData` node for an FX Double Touch Option is shown in Listing 54.

Listing 54: FX Double Touch Option data

```
<FxDoubleTouchOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <PayOffAtExpiry>true</PayOffAtExpiry>
        <ExerciseDates>
         <ExerciseDate>2021-12-14</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <BarrierData>
        ...
        <Type>KnockOut</Type> <!-- KnockOut or KnockIn -->
        <Levels>
            <Level>1.1</Level>
            <Level>1.2</Level>
        </Levels>
        ...
    </BarrierData>
    <ForeignCurrency>EUR</ForeignCurrency>
    <DomesticCurrency>USD</DomesticCurrency>
    <PayoffCurrency>USD</PayoffCurrency>
    <PayoffAmount>100000</PayoffAmount>
    <StartDate>2019-01-25</StartDate>
    <FXIndex>FX-ECB-EUR-USD</FXIndex>
    <Calendar>TARGET</Calendar>
</FxDoubleTouchOptionData>
```

The meanings and allowable values of the elements in the `FxDoubleTouchOptionData` node follow below.

- `OptionData`: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the `OptionData` node for an FxDoubleTouchOption are as below. Note that the `OptionType` can be omitted.
  - `LongShort` The allowable values are *Long* or *Short*.
  - `PayOffAtExpiry` [Optional] *true* for payoff at expiry and *false* for payoff at hit. Currently, for both *KnockOut* and *KnockIn* barriers, only payoff at expiry (i.e. *true*) is supported. Defaults to *true* if left blank or omitted.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - `PaymentData` [Optional]: This defines the settlement of the option payoff.
  - `Premiums` [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2
- `BarrierData`: This is a trade component sub-node outlined in section 19.3.31. Two levels in ascending order should be defined in *Levels*. *Type* should be KnockOut or KnockIn. Levels specified in BarrierData should be quoted as the amount in DomesticCurrency (sold currency) per unit ForeignCurrency (bought currency).
- `ForeignCurrency`: The foreign currency of the FX touch option is equivalent to the bought currency.

  Allowable values: See Table 22 `Currency`.
- `DomesticCurrency`: The domestic currency of the FX touch option is equivalent to the sold currency.

  Allowable values: See Table 22 `Currency`.
- `PayoffCurrency`: The payoff currency of the FX touch option is the currency of the payoff amount.

  Allowable values: See Table 22 `Currency`.
- `PayoffAmount`: The fixed payoff amount expressed in payoff currency. It is cash-or-nothing payoff that depends on the option being in or out of the money, and whether the barrier has been touched.

  Allowable values: Any positive real number.
- `StartDate` [Optional]: The start date for checking if a barrier has been breached prior to today's date. If omitted or left blank no check is made and it is assumed no barrier has been breached in the past. Has no impact if set to today's date or a date in the future.

  Allowable values: See `Date` in Table 20.
- `FXIndex` [Optional]: A reference to an FX Index source to check if the barrier has been breached. Required if StartDate is set to a date prior to today's date, otherwise optional, and can be omitted but not left blank.

  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.

- `Calendar` [Optional]: The calendar associated with the FX Index. Required if StartDate is set to a date prior to today's date, otherwise optional.

  Allowable values: See Table 24 Calendar.

19.2.18 FX European Barrier Option

European exercise, European barrier.

An FX European Barrier option gives the buyer the right, but not the obligation, to exchange a set amount of one currency for another, at a predetermined exchange rate, at one predetermined time in the future. This right may be withdrawn depending upon an FX spot rate reaching a predetermined barrier level at the predetermined time, the underlying is monitored only at expiry with a single barrier (European Barrier style).

The `FxEuropeanBarrierOptionData` node is the trade data container for the *FxEuropeanBarrierOption* trade type. The barrier level of an FX European Barrier Option is quoted as the amount in SoldCurrency per unit BoughtCurrency. The `FxEuropeanBarrierOptionData` node includes one `OptionData` trade component sub-node and one `BarrierData` trade component sub-node plus elements specific to the FX Barrier Option.

The structure of an example `FxEuropeanBarrierOptionData` node for a FX European Barrier Option is shown in Listing 55.

Listing 55: FX European Barrier Option data

```
<FxEuropeanBarrierOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <!-- Bought and Sold currencies/amounts are switched for Put -->
        <OptionType>Call</OptionType>
        <Style>European</Style>
        <Settlement>Cash</Settlement>
        <ExerciseDates>
         <ExerciseDate>2021-12-14</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <BarrierData>
     <Type>UpAndIn</Type>
     <Levels>
        <Level>1.2</Level>
     </Levels>
     <Rebate>100000</Rebate>
    </BarrierData>
    <BoughtCurrency>EUR</BoughtCurrency>
    <BoughtAmount>1000000</BoughtAmount>
    <SoldCurrency>USD</SoldCurrency>
    <SoldAmount>1100000</SoldAmount>
</FxEuropeanBarrierOptionData>
```

The meanings and allowable values of the elements in the `FxEuropeanBarrierOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the `OptionData` node for an FxEuropeanBarrierOption are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*.
    Call means that the holder of the option, upon expiry - assuming knock-in or no knock-out - has the right to receive the BoughtAmount and pay the SoldAmount. *Put* means that the Bought and Sold currencies/amounts are switched compared to the trade data node. For example, holder of BoughtCurrency EUR SoldCurrency JPY FX European Barrier Call Option has the right to buy EUR using JPY, while holder of the Put counterpart has the right to buy JPY using EUR, or equivalently sell EUR for JPY. An alternative to define the latter option is to copy the Call option with following changes:
    a) swapping BoughtCurrency with SoldCurrency, b) swapping BoughtAmount with SoldAmount and c) inverting the barrier level (for example changing 110 to 0.0090909). Here barrier level is quoted as amount of EUR per unit JPY, which is not commonly seen on market and inconsistent with the format in Call options. For these reasons, using Put/Call flag instead is recommended.
  - `Style` The FX European Barrier Option type allows for *European* option exercise style only.
  - `Settlement` The allowable values are *Cash* or *Physical*.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - A `PaymentData` [Optional] node can be added which defines the settlement date of the option payoff.
  - `Premiums` [Optional]: Option premium amounts paid by the option buyer to the option seller. See section 19.3.2
- BarrierData: This is a trade component sub-node outlined in section 19.3.31. Level specified in BarrierData should be quoted as the amount in SoldCurrency per unit BoughtCurrency, with both currencies as defined in FxEuropeanBarrierOptionData node. Changing the option from Call to Put or vice versa does not require switching the barrier level, i.e. the level stays quoted as SoldCurrency per unit BoughtCurrency, regardless of Put/Call.
- BoughtCurrency: The bought currency of the FX barrier option. See OptionData above for more details.

  Allowable values: See Table 22 `Currency`.
- BoughtAmount: The amount in the BoughtCurrency.

  Allowable values: Any positive real number.
- SoldCurrency: The sold currency of the FX barrier option. See OptionData above for more details.

  Allowable values: See Table 22 `Currency`.
- SoldAmount: The amount in the SoldCurrency.

Allowable values: Any positive real number.

Note that FX European Barrier Options also cover Precious Metals, i.e. with currencies XAU, XAG, XPT, XPD, and Cryptocurrencies, see supported Cryptocurrencies in Table 22.

19.2.19 FX KIKO Barrier Option

European exercise, American barriers.

An FX KIKO Barrier option is an option with both a knock-out and a knock-in barrier. The knock-out barrier can happen at any time (American barrier), and once the knock-in barrier is hit the trade becomes a single (American) barrier knock-out trade. The KIKO option can only be exercised (one time, European style) if the knock-out barrier is never touched and the knock-in barrier is touched at least once.

The strike rate and barrier levels of an FX KIKO Barrier Option are expressed as amount in SoldCurrency per unit BoughtCurrency.

The `FXKIKOBarrierOptionData` node is the trade data container for the *FxKIKOBarrierOption* trade type.

The `FXKIKOBarrierOptionData` node includes one `OptionData` trade component sub-node and two `BarrierData` trade component sub-nodes plus elements specific to the FX KIKO Barrier Option. The structure of an example `FXKIKOBarrierOptionData` node for a FX KIKO Barrier Option is shown in Listing 56.

Listing 56: FX KIKO Barrier Option data

```
<FxKIKOBarrierOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <!-- Bought and Sold currencies/amounts are switched for Put -->
        <OptionType>Call</OptionType>
        <Style>European</Style>
        <Settlement>Cash</Settlement>
        <ExerciseDates>
         <ExerciseDate>2021-12-14</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <Barriers>
        <BarrierData>
            <Type>UpAndIn</Type>
            <Levels>
                <Level>1.2</Level>
            </Levels>
        </BarrierData>
        <BarrierData>
            <Type>DownAndOut</Type>
            <Levels>
                <Level>1.2</Level>
            </Levels>
        </BarrierData>
    </Barriers>
    <StartDate>2019-01-25</StartDate>
    <Calendar>TARGET</Calendar>
    <FXIndex>FX-ECB-EUR-USD</FXIndex>
    <BoughtCurrency>EUR</BoughtCurrency>
    <BoughtAmount>1000000</BoughtAmount>
    <SoldCurrency>USD</SoldCurrency>
    <SoldAmount>1100000</SoldAmount>
</FxKIKOBarrierOptionData>
```

The meanings and allowable values of the elements in the `FXKIKOBarrierOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the `OptionData` node for an FxKIKOBarrierOption are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*.
    Call means that the holder of the option, upon expiry - assuming knock-in or no knock-out - has the right to receive the BoughtAmount and pay the SoldAmount.
    Put means that the Bought and Sold currencies/amounts are switched compared to the trade data node. For example, holder of BoughtCurrency EUR SoldCurrency JPY FX KIKO Barrier Call Option has the right to buy EUR using JPY, while holder of the Put counterpart has the right to buy JPY using EUR, or equivalently sell EUR for JPY. An alternative to define the latter option is to copy the Call option with following changes:
    a) swapping BoughtCurrency with SoldCurrency, b) swapping BoughtAmount

with SoldAmount and c) inverting the barrier level (for example changing 110 to 0.0090909). Here barrier level is quoted as amount of EUR per unit JPY, which is not commonly seen on market and inconsistent with the format in Call options. For these reasons, using Put/Call flag instead is recommended.

  - `Style` The FX KIKO Barrier Option type allows for *European* option exercise style only.
  - `Settlement` The allowable values are *Cash* or *Physical*.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - Premiums [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2

- Barriers: This node contains two barrierData nodes, one must be a KnockIn barrier (*UpAndIn* or *DownAndIn*) and the other must be a KnockOut barrier (*UpAndOut* or *DownAndOut*).
- BarrierData: This is a trade component sub-node outlined in section 19.3.31. FxKIKOBarrierOptions do not currently support rebates. Level specified in BarrierData should be quoted as the amount in SoldCurrency per unit BoughtCurrency, with both currencies as defined in FxKIKOBarrierOptionData node. Changing the option from Call to Put or vice versa does not require switching the barrier level, i.e. the level stays quoted as SoldCurrency per unit BoughtCurrency, regardless of Put/Call.
- StartDate[Optional]: The start date for checking if a barrier has been breached prior to today's date. If omitted or left blank no check is made and it is assumed no barrier has been breached in the past. Has no impact if set to today's date or a date in the future.

  Allowable values: See `Date` in Table 20.
- Calendar[Optional]: The calendar associated with the FX Index. Required if StartDate is set to a date prior to today's date, otherwise optional.

  Allowable values: See Table 24 Calendar.
- FXIndex[Optional]: A reference to an FX Index source to check if the barrier has been breached. Required if StartDate is set to a date prior to today's date, otherwise optional and can be omitted but not left blank.

  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.
- BoughtCurrency: The bought currency of the FX barrier option. See OptionData above for more details.

  Allowable values: See Table 22 `Currency`.
- BoughtAmount: The amount in the BoughtCurrency.

  Allowable values: Any positive real number.

- SoldCurrency: The sold currency of the FX barrier option. See OptionData above for more details.

  Allowable values: See Table 22 `Currency`.

- SoldAmount: The amount in the SoldCurrency.

  Allowable values: Any positive real number.

Note that FX KIKO Options also cover Precious Metals, i.e. with currencies XAU, XAG, XPT, XPD, and Cryptocurrencies, see supported Cryptocurrencies in Table 22.

19.2.20 FX Touch Option

An FX Touch Option pays a given cash amount (PayoffAmount) at expiry or at hit if the underlying fx rate has hit a barrier (UpAndIn, DownAndIn - called One Touch) resp. has not hit a barrier (UpAndOut, DownAndOut - called No Touch) using continuous monitoring between start and expiry date. No rebates are supported.

The `FxTouchOptionData` node is the trade data container for the *FxTouchOption* trade type. The `FxTouchOptionData` node includes one `OptionData` trade component sub-node and one `BarrierData` trade component sub-node plus elements specific to the FX Touch Option.

The structure of an example `FxTouchOptionData` node for an FX Touch Option is shown in Listing 57.

Listing 57: FX Touch Option data

```
<FxTouchOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <PayOffAtExpiry>true</PayOffAtExpiry>
        <ExerciseDates>
         <ExerciseDate>2021-12-14</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <BarrierData>
     <Type>DownAndOut</Type>
     <Levels>
      <Level>0.009</Level>
     </Levels>
    </BarrierData>
    <ForeignCurrency>JPY</ForeignCurrency>
    <DomesticCurrency>USD</DomesticCurrency>
    <PayoffCurrency>USD</PayoffCurrency>
    <PayoffAmount>100000</PayoffAmount>
    <StartDate>2019-01-25</StartDate>
    <FXIndex>FX-TR20H-USD-JPY</FXIndex>
    <Calendar>NYB,TKB</Calendar>
</FxTouchOptionData>
```

The meanings and allowable values of the elements in the `FxTouchOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The `OptionType` sub-node is not required and is inferred from the `BarrierData` type (i.e. *Call* for an Up barrier, and *Put* for a Down barrier). The relevant fields in the `OptionData` node for an FxTouchOption are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `PayOffAtExpiry` [Optional] *true* for payoff at expiry and *false* for payoff at hit. For UpAndOut and DownAndOut barrier, only payoff at expiry (*true*) is supported. Defaults to *true* if left blank or omitted. This field is ignored in pricing, and the option payoff will be calculated at expiry. This field only has an impact on the description of the trade economics. The *GenericBarrierOption* can also be used to 'replicate' the *FXTouchOption* with payoff at hit if required.
  - An `ExerciseDates` node where exactly one `ExerciseDate` date element must be given.
  - A `PaymentData` [Optional] node can be added which defines the settlement of the option payoff. If the option is payoff at hit, (i.e. `PayoffAtExpiry` is *false*), the option payment data must be rules-based, and the `RelativeTo` sub-node of (`Rules`) must be set to *Exercise*.
  - `Premiums` [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2
- `BarrierData`: This is a trade component sub-node outlined in section 19.3.31. Level specified in BarrierData should be quoted as the amount in DomesticCurrency (sold currency) per unit ForeignCurrency (bought currency). Note that the level stays quoted as DomesticCurrency per unit ForeignCurrency, regardless of barrier type.
- `ForeignCurrency`: The foreign (bought) currency of the FX touch option.

  Allowable values: See Table 22 `Currency`.
- `DomesticCurrency`: The domestic (sold) currency of the FX touch option.

  Allowable values: See Table 22 `Currency`.
- `PayoffCurrency`: The payoff currency of the FX touch option is the currency of the payoff amount.

  Allowable values: See Table 22 `Currency`.
- `PayoffAmount`: The fixed payoff amount expressed in payoff currency. It is cash-or-nothing payoff that depends on the option being in or out of the money, and whether the barrier has been touched.

  Allowable values: Any positive real number.
- `StartDate` [Optional]: The start date for checking if a barrier has been breached prior to today's date. If omitted or left blank no check is made and it is assumed no barrier has been breached in the past. Has no impact if set to today's date or a date in the future.

  Allowable values: See `Date` in Table 20.

- `FXIndex` [Optional]: A reference to an FX Index source to check if the barrier has been breached. Required if StartDate is set to a date prior to today's date, otherwise optional, and can be omitted but not left blank.

  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.

- `FXIndexDailyLows` [Optional]: Refers to an FX Index that tracks the daily low quotes. This is used to check if the barrier was breached at any point during the day. If not provided, VRE will automatically derive the index name by appending the suffix *_LOW* to the FXIndex source (e.g. *FX-SOURCE_LOW-CCY1-CCY2*). If no fixings are available, the system will fall back to using the fixings from the FXIndex.

  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.

- `FXIndexDailyHighs` [Optional]: Refers to an FX Index that tracks the daily high quotes. This is used to check if the barrier was breached at any point during the day. If not provided, VRE will automatically derive the index name by appending the suffix *_HIGH* to the FXIndex source (e.g. *FX-SOURCE_HIGH-CCY1-CCY2*). If no fixings are available, the system will fall back to using the fixings from the FXIndex.

  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.

- `Calendar` [Optional]: The calendar associated with the FX Index. Required if StartDate is set to a date prior to today's date, otherwise optional.

  Allowable values: See Table 24 Calendar.

Note that FX Touch Options also cover Precious Metals, i.e. with currencies XAU, XAG, XPT, XPD, and Cryptocurrencies, see supported Cryptocurrencies in Table 22.

19.2.21 FX Variance and Volatility Swap

The `FxVarianceSwapData` node is the trade data container for the *FxVarianceSwap* trade type. Only vanilla variance swaps are supported by this trade type - exotic variance swaps are supported by ScriptedTrade. The structure of an example `VarianceSwapData` node for an FX variance swap is shown in Listing 58.

Listing 58: Variance Swap data

```
<FxVarianceSwapData>
        <StartDate>2018-05-10</StartDate>
        <EndDate>2018-11-12</EndDate>
        <Currency>EUR</Currency>
        <Underlying>
          <Type>FX</Type>
          <Name>ECB-EUR-JPY</Name>
        </Underlying>
        <LongShort>Long</LongShort>
        <Strike>0.05</Strike>
        <Notional>200000</Notional>
        <Calendar>EUR</Calendar>
        <MomentType>Variance</MomentType>
</FxVarianceSwapData>
```

The meanings and allowable values of the elements in the `FxVarianceSwapData` node below.

- StartDate: The variance swap start date.
  Allowable values: See `Date` in Table 20.
- EndDate: The variance swap end date.
  Allowable values: See `Date` in Table 20.
- Currency: The bought currency of the variance swap.
  Allowable values: See Table 22 `Currency`.
- Name: The identifier of the underlying currency pair.
  Allowable values: A string of the form SOURCE-CCY1-CCY2, where SOURCE is the fixing source and the fixing is expressed as amount in CCY2 per one unit of CCY1. See Table 28. Note that FxVarianceSwap is an exception in that the ordering of CCY1 and CCY2 must be set up as for `FxIndex`.
- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying FX. The `Underlying` node is described in further detail in Section 19.3.29.
- LongShort: Defines whether the trade is long in the FX variance. For the avoidance of doubt, a long FX swap has positive value if the realised variance exceeds the variance strike.
  Allowable values: *Long, Short*
- Strike: The volatility strike K_{vol} of the variance swap quoted absolutely (i.e. not as a percent). If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any positive real number.
- Notional: The vega notional of the variance swap. This is the notional in terms of volatility units (like the strike). If the swap was struck in terms of a variance notional N_{var}, the corresponding vega notional is given by $N_{vol} = N_{var} * 2 * 100 * K_{vol}$ (where K_{vol} is in absolute terms).
  Allowable values: Any non-negative real number.

- Calendar: The calendar determining the observation/fixing dates according to which variance is accrued is the combination of the calendar(s) given here plus the combined calendars of the two involved currencies.
  Allowable values: See Table 24.
- MomentType[Optional]: A flag to distinguish if the swap is struck in terms of volatility or variance. The MomentType should be set to *Volatility* or *Variance* depending on the payoff. Note that MomentType does not necessarily need to be equivalent to the way the Strike is quoted which is always as a Volatility.
  Allowable values: *Volatility* or *Variance*. Defaults to *Variance* if left blank or omitted.

Note that FX Variance and Volatility Swaps also cover Precious Metals, i.e. with currencies XAU, XAG, XPT, XPD, and Cryptocurrencies, see supported Cryptocurrencies in Table 22.

19.2.22 Equity Option

Quanto payoff means that the payoff `Currency` is different than currency the underlying equity is quoted in.
Composite or "compo" equity options have a `StrikeCurrency` that is different than currency the underlying equity is quoted in. (This is unrelated to the *CompositeTrade* trade type.)

The `EquityOptionData` node is the trade data container for the *EquityOption* trade type. Equity options with exercise styles *European* and *American* are supported.

The `EquityOptionData` node includes one and only one `OptionData` trade component sub-node plus elements specific to the equity option. The structure of an example `EquityOptionData` node for an equity option is shown in Listing 59.

Listing 59: Equity Option data

```
<EquityOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <OptionType>Call</OptionType>
        <Style>American</Style>
        <Settlement>Cash</Settlement>
        <PayOffAtExpiry>true</PayOffAtExpiry>
        <ExerciseDates>
            <ExerciseDate>2022-03-01</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <Name>RIC:.SPX</Name>
    <Currency>USD</Currency>
    <Strike>2147.56</Strike>
    <StrikeCurrency>USD</StrikeCurrency>
    <Quantity>17000</Quantity>
</EquityOptionData>
```

The meanings and allowable values of the elements in the `EquityOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1 Option Data. The relevant fields in the `OptionData` node for an EquityOption are:

- `LongShort`: The allowable values are *Long* or *Short*.
- `OptionType`: The allowable values are *Call* or *Put*. *Call* means that the option holder has the right to buy the given quantity of the underlying equity at the strike price. *Put* means that the option holder has the right to sell the given quantity of the underlying equity at the strike price.
- `Style`: The allowable values are *European* and *American*.
- `Settlement`: The allowable values are *Cash* or *Physical*. If `Currency` and underlying equity currency are different, i.e. Quanto payoff, this must be set to *Cash*.
- `PayOffAtExpiry` [Optional]: The allowable values are *true* for payoff at expiry, or *false* for payoff at exercise. This field is relevant for *American* style EquityOptions, and defaults to *true* if left blank or omitted.
- An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
- `PaymentData` [Optional]: Node used to set the payment date if it differs from the exercise date. Note that for quanto and compo EquityOptions the payment date cannot differ from the exercise date.
- `Premiums` [Optional]: Node for Option premium amounts paid by the option buyer to the option seller.

  Allowable values: See section 19.3.2

- Name: The identifier of the underlying equity or equity index.

  Allowable values: See `Name` for equity trades in Table 31.

- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29.

- Currency: The payment currency of the equity option.

  Allowable values: See `Currency` and `Minor Currencies` in Table 20. If this is different to the currency that the underlying equity is quoted in, then a Quanto payoff will be applied. Using the corresponding major currency for an equity quoted in the minor currency will not correspond to a Quanto payoff.

- Strike[Mandatory except if StrikeData node is used]: The option strike price.

  Allowable values: Any positive real number.

- StrikeCurrency [Mandatory for Quanto/Compo, Optional otherwise]: The currency that the `Strike` is quoted in. If the option is Quanto, then this field must not be left blank, and must equal the currency that the underlying equity is quoted in, up to the minor/major currency. For example, if the underlying equity is quoted in GBP, then`StrikeCurrency` must be either *GBP* or *GBp*. If the option is a Compo option, then this field must not be left blank, and it must equal the payment currency of the option and different to the underlying currency.

  Note:

Quanto: Payment currency and the currency the underlying equity is quoted in differ. StrikeCurrency is in the currency the equity is quoted in.
Compo (Composite): Payment currency and the currency the underlying equity is quoted in differ. StrikeCurrency is in the payment currency.

Allowable values: See Fiat Currencies and Minor Currencies in Table 22. Must be the major or minor currency of the `Currency` field above, or in the Quanto case it must be the major or minor currency the underlying is quoted in. If left blank or omitted, and payment currency is the same as the equity currency, it defaults to the `Currency` field (payment currency) above.

- StrikeData[Optional]: Alternatively, instead of the `Strike` and the `StrikeCurrency` fields above a `StrikeData` node can be used as described in Section 19.3.30. Note that for EquityOptions only `StrikePrice` is supported within the `StrikeData` node, and not `StrikeYield`.
- Quantity: The number of units of the underlying covered by the transaction.

  Allowable values: Any positive real number.

19.2.23 Equity Futures Option

The `EquityFutureOptionData` node is the trade data container for the *EquityFutureOption* trade type. Equity options with exercise styles *European* and *American* are supported. The `EquityFutureOptionData` node includes one and only one `OptionData` trade component sub-node plus elements specific to the equity future option. The structure of an example `EquityFutureOptionData` node for an equity option is shown in Listing 60.

Listing 60: Equity Future Option data

```
<EquityFutureOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <OptionType>Call</OptionType>
        <Style>American</Style>
        <Settlement>Cash</Settlement>
        <PayOffAtExpiry>true</PayOffAtExpiry>
        <ExerciseDates>
            <ExerciseDate>2022-03-01</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <Name>RIC:.SPX</Name>
    <Currency>USD</Currency>
    <StrikeData>
                <StrikePrice>
                        <Value>2147.56</Value>
                        <Currency>USD</Currency>
                </StrikePrice>
        </StrikeData>
    <Quantity>17000</Quantity>
    <FutureExpiryDate>2021-01-29</FutureExpiryDate>
</EquityFutureOptionData>
```

The meanings and allowable values of the elements in the `EquityFutureOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1 Option Data. The relevant fields in the `OptionData` node for an EquityOption are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*. *Call* means that the option holder has the right to buy the given quantity of the underlying equity at the strike price. *Put* means that the option holder has the right to sell the given quantity of the underlying equity at the strike price.
  - `Style` The allowable value is *European*.
  - `Settlement` The allowable values are *Cash* or *Physical*.
  - `PayOffAtExpiry` [Optional] The allowable values are *true* for payoff at expiry, or *false* for payoff at exercise. This field is relevant for *American* style EquityOptions, and defaults to *true* if left blank or omitted.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - Premiums [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2
- Name: The identifier of the underlying equity or equity index.
  Allowable values: See `Name` for equity trades in Table 30.
- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29.
- Currency: The currency of the equity option.
  Allowable values: See Table 22.
- StrikeData: The option strike price.
  Allowable values: Only supports `StrikePrice` as described in Section 19.3.30.
- Quantity: The number of units of the underlying covered by the transaction.
  Allowable values: Any positive real number.
- FutureExpiryDate [Optional]: If `IsFuturePrice` is `true` and the underlying is a future contract settlement price, this node allows the user to specify the expiry date of the underlying future contract.

  Allowable values: This should be a valid date as outlined in Table 20. If not provided, it is assumed that the future contract's expiry date is equal to the option expiry date provided in the `OptionData` node.

19.2.24 Equity Forward

The `EquityForwardData` node is the trade data container for the *EquityForward* trade type. Vanilla equity forwards are supported. The structure of an example `EquityForwardData`

node for an equity forward is shown in Listing 61.

Listing 61: Equity Forward data

```
<EquityForwardData>
  <LongShort>Long</LongShort>
  <Maturity>2018-06-30</Maturity>
  <Name>RIC:.SPX</Name>
  <Currency>USD</Currency>
  <Strike>2147.56</Strike>
  <StrikeCurrency>USD</StrikeCurrency>
  <Quantity>17000</Quantity>
</EquityForwardData>
```

The meanings and allowable values of the elements in the `EquityForwardData` node follow below.

- LongShort: Defines whether the underlying equity will be bought (long) or sold (short). Allowable values: *Long*, *Short*.
- Maturity: The maturity date of the forward contract, i.e. the date when the underlying equity will be bought/sold.
  Allowable values: Any date string, see `Date` in Table 20.
- Name: The identifier of the underlying equity or equity index. Allowable values: See `Name` for equity trades in Table 30.
- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29.
- Currency: The payment currency of the equity forward. If the equity underlying is quoted in a different currency, a `FXIndex` in the `SettlementData` sub-node is required to convert the payoff into the payment currency.
  Allowable values: See Fiat Currencies and Minor Currencies in Table 22.
- Strike: The agreed buy/sell price of the equity forward.
  Allowable values: Any positive real number.
- StrikeCurrency: [Optional] The currency of the strike value. The strike value has to be in underlying quotation currency. If the strike currency is quoted in the minor currency, the strike value will be converted to the major currency. Defaults to the payment currency if omitted or blank.
  Allowable values: See Fiat Currencies and Minor Currencies in Table 22.
- Quantity: The number of units of the underlying equity to be bought/sold.
  Allowable values: Any positive real number.
- SettlementData [Optional]: This node is used to specify the settlement of the cash flows.

The strike value must be quoted in the same currency as the underlying. The underlying prices are always converted to the major underlying currency during curve building. If the strike is quoted in the minor underlying currency, it will be also converted to the major

underlying currency. If the strike currency is blank or omitted, it defaults to payment currency, in this case the payment currency needs to be the same as the underlying currency and same logic applies for minor to major currency conversion.

A `SettlementData` node is shown in Listing 62, and the meanings and allowable values of its elements follow below.

- FXIndex: The FX reference index for determining the FX fixing used to convert the amount from the underlying equity quotation currency to the payment currency. This field is required if the underlying currency doesn't match the deal currency. Otherwise, it is ignored.
  Allowable values: The format of the `FXIndex` is "FX-FixingSource-CCY1-CCY2" as described in Table 28.
- Date [Optional]: If specified, this will be the payment date.
  Allowable values: See `Date` in Table 20. If left blank or omitted, the payment date will be derived from the maturity date applying the `PaymentLag`, `PaymentCalendar` and the `PaymentConvention` as defined in the `Rules` sub-node.
- Rules [Optional]: If `Date` is left blank or omitted, this node will be used to derive the payment date from the maturity date. The `Rules` sub-node is shown in Listing 62, and the meanings and allowable values of its elements follow below.
  - PaymentLag [Optional]: The lag between the maturity date and the payment date.
    Allowable values: Any valid period, i.e. a non-negative whole number, optionally followed by *D* (days), *W* (weeks), *M* (months), *Y* (years). Defaults to 0. If a whole number is given and no letter, it is assumed that it is a number of *D* (days).
  - PaymentCalendar [Optional]: The calendar to be used when applying the payment lag.
    Allowable values: See Table 24 `Calendar`. Defaults to to NullCalendar (no holidays) if left blank or omitted.
  - PaymentConvention [Optional]: The roll convention to be used when applying the payment lag.
    Allowable values: See Table 21 Roll Convention. Defaults to Unadjusted if left blank or omitted.

Listing 62: Example `SettlementData` node with `Rules` sub-node

```
<SettlementData>
  <FXIndex>FX-ECB-EUR-USD</FXIndex>
  <Date>2020-09-03</Date>
  <Rules>
    <PaymentLag>2D</PaymentLag>
    <PaymentCalendar>USD</PaymentCalendar>
    <PaymentConvention>Following</PaymentConvention>
  </Rules>
</SettlementData>
```

19.2.25 Equity Swap

An Equity Swap uses its own trade type *EquitySwap*, and is set up using a `EquitySwapData` node with one leg of type *Equity* and one more leg - called Funding leg - that can be either

Fixed or *Floating*. Listing 63 shows an example. The Equity leg contains an additional `EquityLegData` block. See 19.3.16 for details on the Equity leg specification.

Note that the *Equity* leg of an *EquitySwap* can only include one single underlying equity name (that can be an equity index name). For instruments with more than one underlying equity name, TradeType *TotalReturnSwap* (GenericTRS) should be used instead.

Cross currency *EquitySwaps* are supported, i.e. the Equity and the Funding legs do not need to have the same currency. However, if the Funding leg uses `Indexings` with `FromAssetLeg` set to *true* to derive the notionals from the Equity leg, then the Funding leg must use the same currency as the Equity leg.

Note that pricing for an *EquitySwap* is based on discounted cashflows, whereas pricing for a *TotalReturnSwap* (GenericTRS) on an equity underlying uses the accrual method. The accrual method is common practice when daily unwind rights are present in the trade terms.

Also note that, unlike other leg types, the `DayCounter` field is optional for an *Equity* leg, and defaults to *ACT/365* if left blank or omitted. The daycount convention for the equity leg of an equity swap does not impact pricing, only the accrued amount (displayed in cashflows).

Listing 63: Equity Swap Data

```
<EquitySwapData>
  <LegData>
    <LegType>Floating</LegType>
    <Payer>true</Payer>
    <DayCounter>ACT/365</DayCounter>
    ...
  </LegData>
  <LegData>
    <LegType>Equity</LegType>
    <Payer>false</Payer>
    <DayCounter>ACT/365</DayCounter>
    ...
    <EquityLegData>
    ...
    </EquityLegData>
  </LegData>
</EquitySwapData>
```

If the equity swap has a resetting notional, typically the Funding leg's notional will be aligned with the equity leg's notional. To achieve this, `Indexings` on the floating leg can be used, see 19.3.8. In the context of equity swaps the indexings can be defined in a simplified way by adding an `Indexings` node with a subnode `FromAssetLeg` set to *true* to the Funding leg's `LegData` node. The `Notionals` node is not required in the Funding leg's LegData in this case. An example is shown in listing 64.

Listing 64: Equity Swap Data with notional reset and FX indexing

```
<EquitySwapData>
  <LegData>
    <LegType>Floating</LegType>
    <Currency>USD</Currency>
    ...
    <!-- Notionals node is not required, set to 1 internally -->
    ...
    <Indexings>
      <!-- derive the indexing information (equity price, FX) from the Equity leg -->
      <FromAssetLeg>true</FromAssetLeg>
    </Indexings>
  </LegData>
  <LegData>
    <LegType>Equity</LegType>
      <Currency>USD</Currency>
      ...
      <EquityLegData>
        <Quantity>1000</Quantity>
            <Underlying>
             <Type>Equity</Type>
             <Name>.STOXX50E</Name>
             <IdentifierType>RIC</IdentifierType>
            </Underlying>
        <InitialPrice>2937.36</InitialPrice>
        <NotionalReset>true</NotionalReset>
        <FXTerms>
          <EquityCurrency>EUR</EquityCurrency>
          <FXIndex>FX-ECB-EUR-USD</FXIndex>
        </FXTerms>
      </EquityLegData>
      ...
  </LegData>
</EquitySwapData>
```

19.2.26 Dividend Swap

An Dividend Swap uses its the trade type *EquitySwap*, shown above 19.2.25, and is set up using a `EquitySwapData` node with one leg of type *Equity*, with *ReturnType* equal to *Dividend* and one more leg that can be either *Fixed* or *Floating*. Listing 65 shows an example.

An example is shown in listing 64.

Listing 65: Dividend Swap Data

```
<EquitySwapData>
    <LegData>
        ...
    </LegData>
    <LegData>
        <Payer>false</Payer>
        <LegType>Equity</LegType>
        <Currency>EUR</Currency>
        <PaymentConvention>Following</PaymentConvention>
        <DayCounter>A360</DayCounter>
        <EquityLegData>
            <ReturnType>Dividend</ReturnType>
            <Underlying>
             <Type>Equity</Type>
             <Name>.STOXX50E</Name>
             <IdentifierType>RIC</IdentifierType>
            </Underlying>
            <Quantity>10000</Quantity>
        </EquityLegData>
        <ScheduleData>
            <Rules>
                <StartDate>2018-12-31</StartDate>
                <EndDate>2020-12-31</EndDate>
                <Tenor>6M</Tenor>
                <Calendar>EUR</Calendar>
                <Convention>ModifiedFollowing</Convention>
                <Rule>Forward</Rule>
            </Rules>
        </ScheduleData>
    </LegData>
</EquitySwapData>
```

19.2.27 Equity Asian Option

An Equity Asian Option is a path-dependent option whose payoff depends upon the averaged price of an Equity underlying over a pre-set period of time.

The `EquityAsianOptionData` node is the trade data container for the *EquityAsianOption* trade type. The `EquityAsianOptionData` node includes one `OptionData` trade component sub-node plus elements specific to the Equity Asian Option.

The structure of an example `EquityAsianOptionData` node for an Equity Asian Option is shown in Listing 66.

Listing 66: Equity Asian Option data

```
<Trade id="EquityAsianOption">
        <TradeType>EquityAsianOption</TradeType>
        <Envelope>
                <CounterParty>CPTY_A</CounterParty>
                <NettingSetId>CPTY_A</NettingSetId>
                <AdditionalFields />
        </Envelope>
        <EquityAsianOptionData>
                <Quantity>100</Quantity>
                <Currency>USD</Currency>
                <StrikeData>
                        <Value>3100</Value>
                        <Currency>USD</Currency>
                </StrikeData>
                <Underlying>
                        <Type>Equity</Type>
                        <Name>RIC:.SPX</Name>
                        <Currency>USD</Currency>
                </Underlying>
                <OptionData>
                        <LongShort>Long</LongShort>
                        <OptionType>Call</OptionType>
                        <PayoffType>Asian</PayoffType>
                        <PayoffType2>Arithmetic</PayoffType2>
                        <ExerciseDates>
                                <ExerciseDate>2020-07-15</ExerciseDate>
                         </ExerciseDates>
                         <Premiums> ... </Premiums>
                </OptionData>
                <Settlement>2020-07-20</Settlement>
                <ObservationDates>
                        <Rules>
                                <StartDate>2019-12-27</StartDate>
                                <EndDate>2020-07-06</EndDate>
                                <Tenor>1D</Tenor>
                                <Calendar>US</Calendar>
                                <Convention>F</Convention>
                                <TermConvention>F</TermConvention>
                                <Rule>Forward</Rule>
                        </Rules>
                </ObservationDates>
        </EquityAsianOptionData>
</Trade>
```

In the above example, the holder of the EquityAsianOption has a call option that gives the right but not obligation to pay 310,000 USD (Strike*Quantity) and receive [the averaged equity spot price during the Asian period] USD multiplied by the `Quantity`.

If OptionType would be changed to Put, the holder of the option would have the right to receive 310,000 USD (Strike*Quantity) and pay [the averaged equity spot price during the Asian period] USD multiplied by the `Quantity`.

The payoff is:

$$Payoff = Quantity \cdot MAX(\omega \cdot (A(0,T) - K), 0)$$

where:

- $A(0,T)$: the arithmetic average of underlying euqity spot price over the Asian observation period from start 0 to end T, quoted in `Currency`
- K: equity strike price, quoted in `Currency`
- ω: 1 for a call option (ie receiving averaged equity spot price and paying strike), -1 for a put option

The meanings and allowable values of the elements in the `EquityAsianOptionData` node follow below.

- StrikeData: A node containing the strike in `Value` and the currency in which both the underlying and the strike are quoted in `Currency`. Allowable values: See `Currency` in Table 20. The strike may be any positive real number. The currency provided in this node may be quoted as corresponding minor currency to the underlying major currency.
- Quantity: The quantity of the underlying equities. See payoff formula above.
  Allowable values: all positive real numbers
- Underlying: One (and only one) `Underlying` node where `Type` must be set to *Equity*.
  Allowable values: See 19.3.29. Note that the equity must be quoted in the `Currency` above.
- OptionData: The relevant fields in the `OptionData` node for an EquityAsianOption are the `LongShort` flag, the `OptionType` (*call/put*), the `PayoffType` which must be set to *Asian* or *AverageStrike* to identify a fixed or floating strike asian payoff, and the `ExerciseDates` node where exactly one ExerciseDate date element must be given. `PayoffType2` can be optionally set to *Arithmetic* or *Geometric* and defaults to *Arithmetic* if not given. Furthermore, a *Premiums* node can be added to represent deterministic option premia to be paid by the option holder.
  Allowable values: See 19.3.1 for the general structure of the option data node
- Settlement[Optional]: The settlement date.
  Allowable values: See `Date` in Table 20. Defaults to the ExerciseDate if left blank or omitted.
- ObservationDates: The observation dates for the asian period, given as a rules-based or dates-based schedule, analogous to a `ScheduleData` node but called `ObservationDates`.
  Allowable values: See the definition in 19.3.4

19.2.28 Equity Barrier Option

European exercise, American barrier.

An Equity Barrier Option is a path-dependent option whose existence depends upon an Equity underlying spot price reaching a pre-set barrier level. Exercise is European.

This product has a continuously monitored single barrier (American Barrier style) with a Vanilla European Equity Option Underlying.

The `EquityBarrierOptionData` node is the trade data container for the *EquityBarrierOption* trade type. The barrier level of an Equity Barrier Option should be

quoted in the currency of the underlying Equity spot price. The `EquityBarrierOptionData` node includes one `OptionData` trade component sub-node and one `BarrierData` trade component sub-node plus elements specific to the Equity Barrier Option.

The structure of an example `EquityBarrierOptionData` node for an Equity Barrier Option is shown in Listing 67.

Listing 67: Equity Barrier Option data

```
<EquityBarrierOptionData>
    <OptionData>
        ...
    </OptionData>
    <BarrierData>
        ...
    </BarrierData>
    <StartDate>2025-01-25</StartDate>
    <Calendar>TARGET</Calendar>
    <EQIndex>EQ-RIC:.SPX</EQIndex>
    <Name>RIC:.SPX</Name>
    <StrikeData>
    <StrikePrice>
     <Value>3200.00</Value>
     <Currency>USD</Currency>
    </StrikePrice>
    </StrikeData>
    <Quantity>1000</Quantity>
    <Currency>USD</Currency>
</EquityBarrierOptionData>
```

The meanings and allowable values of the elements in the `EquityBarrierOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. Note that the Equity Barrier Option type allows for *European* option style only.
- BarrierData: This is a trade component sub-node outlined in section 19.3.31. Level specified in BarrierData should be quoted in the same currency with the underlying Equity spot price. Changing the option from Call to Put or vice versa does not require switching the barrier level.
- StartDate[Optional]: The start date for checking if a barrier has been breached prior to today's date. If omitted or left blank no check is made and it is assumed no barrier has been breached in the past. Has no impact if set to today's date or a date in the future.

  Allowable values: See `Date` in Table 20.
- Calendar[Optional]: The calendar associated with the Equity Index. Required if StartDate is set to a date prior to today's date, otherwise optional.

  Allowable values: See Table 24 Calendar.
- EQIndex[Optional]: A reference to an Equity Index source to check if the barrier has been breached. Required if StartDate is set to a date prior to today's date, otherwise optional and can be omitted but not left blank.

Allowable values: The format of the Equity Index is"EQ-RIC:Code".

- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29.
- StrikeData: A node containing the strike in `Value` and the currency in which both the underlying and the strike are quoted in `Currency`. I.e. compo options with strike currency not equal to underlying equity currency are not supported for this trade type.

  Allowable values: Only supports `StrikePrice` as described in Section 19.3.30.
- Quantity: The number of units of the underlying covered by the transaction.

  Allowable values: Any positive real number.
- Currency: The payment currency of the trade.

  Allowable values: See `Currency` and `Minor Currencies` in Table 20. This should be equal to the underlying equity except for the major / minor distinction. I.e. quanto payoffs that are usually identified by setting the payment currency to a different currency than the underlying equity currency, are not allowed for this trade type.

19.2.29 Equity Digital Option

The `EquityDigitalOptionData` node is the trade data container for the *EquityDigitalOption* trade type. The `EquityDigitalOptionData` node includes one `OptionData` trade component sub-node plus elements specific to the Equity Digital Option. The structure of an example `EquityDigitalOptionData` node for an Equity Digital Option is shown in Listing 68.

Listing 68: Equity Digital Option data

```
<EquityDigitalOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <OptionType>Call</OptionType>
        <Style>European</Style>
        <ExerciseDates>
            <ExerciseDate>2027-02-26</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <Strike>3300</Strike>
    <PayoffCurrency>USD</PayoffCurrency>
    <PayoffAmount>1000</PayoffAmount>
    <Name>RIC:.SPX</Name>
    <Quantity>1000</Quantity>
</EquityDigitalOptionData>
```

The meanings and allowable values of the elements in the `EquityDigitalOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the `OptionData` node for an EquityDigitalOption are:

  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*. *Call* means that the option is in the money when the underlying equity price is above the strike, and *Put* means that the option is in the money when the underlying equity price is below the strike.
  - `Style` The allowable value is *European*. Note that the Equity Digital Option type allows for *European* option style only.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - Premiums [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2

- Strike: The option strike price per one unit of the underlying, expressed in the currency of the underlying equity .

  Allowable values: Any positive real number.

- PayoffCurrency: The payoff currency of the Equity Digital Option is the currency of the payoff amount. Must be consistent with the currency of the underlying Equity spot price.

  Allowable values: See Table 22 `Currency`.

- PayoffAmount: The fixed payoff amount per unit of underlying expressed in payoff currency. It is cash-or-nothing payoff that depends on the option being in or out of the money.

  Allowable values: Any positive real number.

- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29.

- Quantity: The number of units of the underlying covered by the transaction.

  Allowable values: Any positive real number.

19.2.30 Equity Double Barrier Option

The `EquityDoubleBarrierOptionData` node is the trade data container for the *EquityDoubleBarrierOption* trade type.

An Equity Double Barrier Option is a path-dependent option whose existence depends upon an Equity spot rate reaching one of the two pre-set barrier levels. Exercise is European, and barriers are American (continuously monitored).

Equity Double Barrier options can be knock-in or knock-out:

- A knock-in option is a barrier option that only comes into existence/becomes active when the Equity spot rate reaches the one of the barrier level at any point in the option's

life. Once a barrier is knocked-in, the option will not cease to exist until the option expires and effectively it becomes a Vanilla Equity Option.

- A knock-out option starts its life active, but ceases to exist/becomes inactive, if the one of the barriers is reached during the life of the option.

The barrier levels of an Equity Double Barrier Option should be quoted in the currency of the underlying Equity spot price. The `EquityDoubleBarrierOptionData` node includes one `OptionData` trade component sub-node and one `BarrierData` trade component sub-node plus elements specific to the Equity Double Barrier Option. The structure of an example `EquityDoubleBarrierOptionData` node for a Equity Double Barrier Option is shown in Listing 69.

Listing 69: Equity Double Barrier Option data

```
<EquityDoubleBarrierOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <OptionType>Call</OptionType>
        <Style>European</Style>
        <Settlement>Cash</Settlement>
        <ExerciseDates>
            <ExerciseDate>2021-01-29</ExerciseDate>
        </ExerciseDates>
    </OptionData>
    <BarrierData>
        <Type>KnockOut</Type>
        <Levels>
            <Level>3000.00</Level>
            <Level>3500.00</Level>
        </Levels>
    </BarrierData>
    <Name>RIC:.SPX</Name>
    <Currency>USD</Currency>
            <StrikeData>
                    <StrikePrice>
                            <Value>3200.00</Value>
                            <Currency>USD</Currency>
                    </StrikePrice>
            </StrikeData>
    <Quantity>1000</Quantity>
    <StartDate>2019-12-27</StartDate>
    <Calendar>US-NYSE</Calendar>
    <EQIndex>EQ-RIC:.SPX</EQIndex>
</EquityDoubleBarrierOptionData>
```

The meanings and allowable values of the elements in the `EquityDoubleBarrierOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the `OptionData` node for an EquityDoubleBarrierOption are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*.

  - `Style` The Equity Double Barrier Option type allows for *European* option exercise style only.
  - `Settlement` The allowable values are *Cash* or *Physical*.
  - An `ExerciseDates` node where exactly one `ExerciseDate` date element must be given.
  - Optional `PremiumAmount`, `PremiumCurrency`, and `PremiumPayDate` fields to specify the Equity Double Barrier Option premium.

- BarrierData: This is a trade component sub-node outlined in section 19.3.31. Two levels in ascending order should be defined in `Levels`. `Type` should be *KnockOut* or *KnockIn*.

  Allowable values: See Table 24 Calendar.

- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29.

- Currency: The currency of the equity option.

  Allowable values: See Table 22 `Currency`.

- StrikeData: A node containing the strike in `Value` and the currency in which both the underlying and the strike are quoted in `Currency`. Allowable values: Only supports `StrikePrice` as described in Section 19.3.30.

- Quantity: The number of units of the underlying covered by the transaction.

  Allowable values: Any positive real number.

- StartDate [Optional]: The start date for checking if a barrier has been breached prior to today's date. If omitted or left blank no check is made and it is assumed no barrier has been breached in the past. Has no impact if set to today's date or a date in the future.

  Allowable values: See `Date` in Table 20.

- Calendar [Optional]: The calendar associated with the Equity Index. Required if StartDate is set to a date prior to today's date, otherwise optional.

- EQIndex [Optional]: A reference to an Equity Index source to check if the barrier has been breached. Required if StartDate is set to a date prior to today's date, otherwise optional and can be omitted but not left blank.

  Allowable values: The format of the Equity Index is"EQ-RIC:Code".

19.2.31 Equity Double Touch Option

An Equity Double Touch Option pays a given cash amount (PayoffAmount) at expiry or at hit if the underlying equity price or index has hit either of the barriers (KnockIn) resp. has not hit any of barriers (KnockOut) using continuous monitoring between start and expiry date. No rebates are supported.

The `EquityDoubleTouchOptionData` node is the trade data container for the *EquityDoubleTouchOption* trade type. The `EquityDoubleTouchOptionData` node includes

one OptionData trade component sub-node and one BarrierData trade component sub-node plus elements specific to the Equity Double Touch Option.

The structure of an example EquityDoubleTouchOptionData node for an Equity Double Touch Option is shown in Listing 70.

Listing 70: Equity Double Touch Option data

```
<EquityDoubleTouchOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <PayOffAtExpiry>true</PayOffAtExpiry>
        <ExerciseDates>
         <ExerciseDate>2021-12-14</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <BarrierData>
        ...
        <Type>KnockIn</Type> <!-- KnockOut or KnockIn -->
        <Levels>
            <Level>3000</Level>
            <Level>4500</Level>
        </Levels>
        ...
    </BarrierData>
    <PayoffCurrency>USD</PayoffCurrency>
    <PayoffAmount>1000000</PayoffAmount>
    <Name>RIC:.SPX</Name>
    <StartDate>2021-03-01</StartDate>
    <Calendar>USD</Calendar>
    <EQIndex>EQ-RIC:.SPX</EQIndex>
</EquityDoubleTouchOptionData>
```

The meanings and allowable values of the elements in the EquityDoubleTouchOptionData node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the OptionData node for an EquityDoubleTouchOption are as below. Note that the OptionType can be omitted.
  - LongShort The allowable values are *Long* or *Short*.
  - PayOffAtExpiry [Optional] *true* for payoff at expiry and *false* for payoff at hit. Currently, for both *KnockOut* and *KnockIn* barriers, only payoff at expiry (i.e. *true*) is supported. Defaults to *true* if left blank or omitted.
  - An ExerciseDates node where exactly one ExerciseDate date element must be given.
  - Premiums [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2

- BarrierData: This is a trade component sub-node outlined in section 19.3.31. Two levels in ascending order should be defined in *Levels*. *Type* should be *KnockOut* or *KnockIn*. Levels specified in BarrierData should be quoted in the same currency as the underlying Equity spot prices.
- PayoffCurrency: The payoff currency of the Equity Double Touch Option is the currency of the payoff amount. Must be consistent with the currency of the underlying Equity spot prices.

  Allowable values: See Table 22 `Currency`.
- PayoffAmount: The fixed payoff amount expressed in payoff currency. It is cash-or-nothing payoff that depends on the option being in or out of the money, and whether the barrier has been touched.

  Allowable values: Any positive real number.
- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29.
- StartDate[Optional]: The start date for checking if a barrier has been breached prior to today's date. If omitted or left blank no check is made and it is assumed no barrier has been breached in the past. Has no impact if set to today's date or a date in the future.

  Allowable values: See `Date` in Table 20.
- Calendar[Optional]: The calendar associated with the Equity Index. Required if StartDate is set to a date prior to today's date, otherwise optional.

  Allowable values: See Table 24 Calendar.
- EQIndex[Optional]: A reference to an Equity Index source to check if the barrier has been breached. Required if StartDate is set to a date prior to today's date, otherwise optional and can be omitted but not left blank.

  Allowable values: The format of the Equity Index is"EQ-RICCode".

19.2.32 Equity European Barrier Option

European exercise, European barrier.

An Equity European Barrier Option gives the buyer the right, but not the obligation, to buy a set number of shares of a single name equity or an equity index, at a predetermined strike price, at one predetermined time in the future. This right may be withdrawn depending upon an Eqity spot price or index reaching a predetermined barrier level at the predetermined time, the underlying is monitored only at expiry with a single barrier (European Barrier style).

The `EquityEuropeanBarrierOptionData` node is the trade data container for the *EquityEuropeanBarrierOption* trade type. The barrier level of an Equity European Barrier Option is quoted in the currency of the underlying Equity spot price. The `EquityEuropeanBarrierOptionData` node includes one `OptionData` trade component sub-node and one `BarrierData` trade component sub-node plus elements specific to the Equity European Barrier Option.

The structure of an example `EquityEuropeanBarrierOptionData` node for an Equity European Barrier Option is shown in Listing 71.

Listing 71: Equity European Barrier Option data

```
<EquityEuropeanBarrierOptionData>
    <OptionData>
        ...
    </OptionData>
    <BarrierData>
        ...
    </BarrierData>
    <Name>RIC:.SPX</Name>
    <StrikeData>
        <StrikePrice>
                                    <Value>3200.00</Value>
                                    <Currency>USD</Currency>
                            </StrikePrice>
    </StrikeData>
    <Quantity>1000</Quantity>>
</EquityEuropeanBarrierOptionData>
```

The meanings and allowable values of the elements in the `EquityEuropeanBarrierOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. Note that the Equity European Barrier Option type allows for *European* option style only.
- BarrierData: This is a trade component sub-node outlined in section 19.3.31. Level specified in BarrierData should be quoted in the same currency with the underlying Equity spot price. Changing the option from Call to Put or vice versa does not require switching the barrier level.
- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29.
- StrikeData: A node containing the strike in `Value` and the currency in which both the underlying and the strike are quoted in `Currency`. Allowable values: Only supports `StrikePrice` as described in Section 19.3.30.
- Quantity: The number of units of the underlying covered by the transaction.

  Allowable values: Any positive real number.

19.2.33 Equity Touch Option

An Equity Touch Option pays a given cash amount (PayoffAmount) at expiry or at hit if the underlying equity price or index has hit a barrier (UpAndIn, DownAndIn) resp. has not hit a barrier (UpAndOut, DownAndOut) using continuous monitoring between start and expiry date. No rebates are supported.

The `EquityTouchOptionData` node is the trade data container for the *EquityTouchOption* trade type. The `EquityTouchOptionData` node includes one `OptionData` trade component

sub-node and one `BarrierData` trade component sub-node plus elements specific to the Equity Touch Option.

The structure of an example `EquityTouchOptionData` node for an Equity Touch Option is shown in Listing 72.

Listing 72: Equity Touch Option data

```
<EquityTouchOptionData>
    <OptionData>
        <LongShort>Long</LongShort>
        <PayOffAtExpiry>true</PayOffAtExpiry>
        <Settlement>Cash</Settlement>
        <ExerciseDates>
         <ExerciseDate>2022-03-01</ExerciseDate>
        </ExerciseDates>
        ...
    </OptionData>
    <BarrierData>
       <Type>UpAndIn</Type>
       <Levels>
           <Level>3300</Level>
       </Levels>
    </BarrierData>
    <PayoffCurrency>USD</PayoffCurrency>
    <PayoffAmount>1000000</PayoffAmount>
    <Name>RIC:.SPX</Name>
    <StartDate>2019-12-27</StartDate>
    <Calendar>US-NYSE</Calendar>
    <EQIndex>EQ-RIC:.SPX</EQIndex>>
</EquityTouchOptionData>
```

The meanings and allowable values of the elements in the `EquityTouchOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The `OptionType` sub-node is not required and is inferred from the `BarrierData` type (i.e. *Call* for an Up barrier, and *Put* for a Down barrier). The relevant fields in the `OptionData` node for an EquityTouchOption are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `PayOffAtExpiry` [Optional] *true* for payoff at expiry and *false* for payoff at hit. For *UpAndOut* and *DownAndOut* barriers, only payoff at expiry (i.e. *true*) is supported. Defaults to *true* if left blank or omitted.
  - `Settlement` The allowable values are *Cash* or *Physical*.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - Premiums [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2

- BarrierData: This is a trade component sub-node outlined in section 19.3.31. Level specified in BarrierData should be quoted in the same currency as the underlying Equity spot price.
- PayoffCurrency: The payoff currency of the Equity Touch Option is the currency of the payoff amount. Must be consistent with the currency of the underlying Equity spot price.

  Allowable values: See `Currency` in Table 20.
- PayoffAmount: The fixed payoff amount expressed in payoff currency. It is cash-or-nothing payoff that depends on the option being in or out of the money, and whether the barrier has been touched.

  Allowable values: Any positive real number.
- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29.
- StartDate[Optional]: The start date for checking if a barrier has been breached prior to today's date. If omitted or left blank no check is made and it is assumed no barrier has been breached in the past. Has no impact if set to today's date or a date in the future.

  Allowable values: See `Date` in Table 20.
- Calendar[Optional]: The calendar associated with the Equity Index. Required if StartDate is set to a date prior to today's date, otherwise optional.

  Allowable values: See Table 24 Calendar.
- EQIndex[Optional]: A reference to an Equity Index source to check if the barrier has been breached. Required if StartDate is set to a date prior to today's date, otherwise optional and can be omitted but not left blank.

  Allowable values: The format of the Equity Index is"EQ-RICCode".

19.2.34 Equity Variance Swap

The `EqutiyVarianceSwapData` node is the trade data container for the *EquityVarianceSwap* trade type. Only vanilla variance swaps are supported. The structure of an example `EqutiyVarianceSwapData` node for an equity variance swap is shown in Listing 73.

Listing 73: Variance Swap data

```
<EquityVarianceSwapData>
    <StartDate>2016-01-29</StartDate>
    <EndDate>2016-05-05</EndDate>
    <Currency>USD</Currency>
    <Underlying>
      <Type>Equity</Type>
      <Name>.SPX</Name>
      <IdentifierType>RIC</IdentifierType>
    </Underlying>
    <LongShort>Long</LongShort>
    <Strike>0.20</Strike>
    <Notional>50000</Notional>
    <Calendar>US</Calendar>
    <MomentType>Variance</MomentType>
    <AddPastDividends>true</AddPastDividends>
</EqutiyVarianceSwapData>
```

The meanings and allowable values of the elements in the `EquityVarianceSwapData` node below.

- StartDate: The variance swap start date.
  Allowable values: See `Date` in Table 20.
- EndDate: The variance swap end date.
  Allowable values: See `Date` in Table 20.
- Currency: The bought currency of the variance swap.
  Allowable values: See `Currency` in Table 20.
- Name: The identifier of the underlying equity or equity index.
  Allowable values: See `Name` for equity trades in Table 30.
- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29.
- LongShort: Defines whether the trade is long in the equity variance. For the avoidance of doubt, a long variance swap has positive value if the realised variance exceeds the variance strike.
  Allowable values: *Long, Short*
- Strike: The volatility strike K_{vol} of the variance swap quoted absolutely (i.e. not as a percent). If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any positive real number.
- Notional: The vega notional of the variance swap. This is the notional in terms of volatility units (like the strike). If the swap was struck in terms of a variance notional N_{var}, the corresponding vega notional is given by $N_{vol} = N_{var} * 2 * 100 * K_{vol}$ (where K_{vol} is in absolute terms).
  Allowable values: Any non-negative real number.
- Calendar: The calendar determining the observation/fixing dates according to which

variance is accrued is the combination of the calendar(s) given here plus the calendar associated with the equity in the equity curve configuration. If no such calendar is given in the equity curve configuration the standard calendar for the equity currency (also defined in the curve config) is used instead.
Allowable values: See Table 24.

- MomentType[Optional]: A flag to distinguish if the swap is struck in terms of volatility or variance. The MomentType should be set to *Volatility* or *Variance* depending on the payoff. Note that MomentType does not necessarily need to be equivalent to the way the Strike is quoted which is always as a Volatility.
  Allowable values: *Volatility* or *Variance*. Defaults to *Variance* if left blank or omitted.
- AddPastDividends[Optional]: A flag to distinguish if past dividend payments should be added to the fixings when calculating accrued variance.
  Allowable values: *true* or *false*. Defaults to *false* if left blank or omitted.

19.2.35 Equity Cliquet Option

The `EquityCliquetOptionData` node is the trade data container for the *EquityCliquetOption* trade type. A cliquet option consists of a series of consecutive forward starting equity options, with each option being struck at a given moneyness (commonly at-the-money) when it becomes active.

The structure of an example `EquityCliquetOptionData` node for an equity cliquet option is shown in Listing 74.

Listing 74: Cliquet Option data

```
<EquityCliquetOptionData>
    <Underlying>
      <Type>Equity</Type>
      <Name>.SPX</Name>
      <IdentifierType>RIC</IdentifierType>
    </Underlying>
    <Currency>USD</Currency>
    <Notional>1000000.0</Notional>
    <LongShort>Short</LongShort>
    <OptionType>Call</OptionType>
    <Moneyness>1.0</Moneyness>
    <LocalCap>0.07</LocalCap>
    <LocalFloor>-0.06</LocalFloor>
    <GlobalCap>0.07</GlobalCap>
    <GlobalFloor>-0.07</GlobalFloor>
    <ScheduleData>
        <Dates>
            <Dates>
                <Date>20171231</Date>
                <Date>20181231</Date>
                <Date>20191231</Date>
                <Date>20201231</Date>
                <Date>20211231</Date>
                <Date>20221231</Date>
            </Dates>
            <Calendar>USD</Calendar>
            <Convention>F</Convention>
        </Dates>
    </ScheduleData>
    <SettlementDays>5</SettlementDays>
    <Premium>0.027</Premium>
    <PremiumPaymentDate>31-12-2017</PremiumPaymentDate>
    <PremiumCurrency>USD</PremiumCurrency>
</EquityCliquetOptionData>
```

The meanings and allowable values of the elements in the `CliquetOptionData` node below.

- Name: The identifier of the underlying equity or equity index.
  Allowable values: See `Name` for equity trades in Table 30.
- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29.
- Currency: The currency of the notional, and thus of the option.
  Allowable values: See `Currency` in Table 20. The Currency must be the same as the currency of the underlying equity.
- Notional: The notional of the cliquet option.
  Allowable values: Any positive real number.
- LongShort: Defines whether the trade is long or short the option.
  Allowable values: *Long, Short*
- OptionType: The type of the option.

Allowable values: *Call, Put*

- Moneyness: Adjustment of option return. The moneyness M each forward starting option is being struck at.
  Allowable values: Any real number. Expressed in decimal form where 1.0 is at-the-money, 1.1 is 110% of the at-the-money strike, 0.9 is 90% of the at-the-money strike, etc.
- LocalCap[Optional]: The local cap, cap_l, in each of the option return.
  Allowable values: Any real number. If omitted, no local cap is applied. Can't be left blank.
- LocalFloor[Optional]: The local floor, $floor_l$, in each of the option return.
  Allowable values: Any real number. If omitted, no local floor is applied. Can't be left blank.
- GlobalCap[Optional]: The global cap, cap_g, for the option return.
  Allowable values: Any real number. If omitted, no global cap is applied. Can't be left blank.
- GlobalFloor[Optional]: The global floor,$floor_g$, for the option return.
  Allowable values: Any real number. If omitted, no global floor is applied. Can't be left blank.
- ScheduleData: A schedule of dates that define the valuation dates of the consecutive forward starting options forming the Equity Cliquet Option. The first date in the schedule is the start date of the first consecutive option, the second date in the schedule is the end/valuation date of the first consecutive option, and also the start date of the second consecutive option, etc. The last date is the final valuation date, with payoff of the whole Cliquet option at this date plus `SettlementDays`.
  Allowable values: A node on the same form as `ScheduleData`, (see 19.3.4).
- SettlementDays[Optional]: Number of days from the last valuation date to the payoff being paid or received. The payoff date is determined with regards to calendar and term date convention of the schedule's calendar.
  Allowable values: Any positive integer. Defaults to zero if left blank or omitted.
- Premium[Optional]: The premium paid for the option.
  Allowable values: Any real number. Expressed in decimal form relative to notional.
- PremiumPaymentDate[Optional]: The date the premium is the paid.
  Allowable values: See `Date` in Table 20. Note that if a Premium is specified, a PremiumPaymentDate must also be specified.
- PremiumCurrency[Optional]: The currency the premium is to paid in.
  Allowable values: See `Currency` in Table 20. Defaults to the currency of the notional.

19.2.36 Equity Position

An equity position represents a position in a single equity - using a single `Underlying` node, or in a weighted basket of underlying equities - using multiple `Underlying` nodes.

An Equity Position can be used both as a stand alone trade type (TradeType: *EquityPosition*) or as a trade component (`EquityPositionData`) used within the *TotalReturnSwap* (Generic

TRS) trade type, to set up for example Equity Basket trades.

It is set up using an `EquityPositionData` block as shown in listing 75. The meanings and allowable values of the elements in the block are as follows:

- Quantity: The number of shares or units of the weighted basket held.
  Allowable values: Any positive real number
- Underlying: One or more underlying descriptions. If a basket of equities is defined, the `Weight` field should be populated for each underlyings. The weighted basket price is then given by

$$\text{Basket-Price} = \text{Quantity} \times \sum_i \text{Weight}_i \times S_i \times \text{FX}_i$$

  where

  - S_i is the price of the ith share in the basket
  - FX_i is the FX Spot converting from the ith equity currency to the first equity currency which is by definition the currency in which the npv of the basket is expressed.

  Allowable values: See 19.3.29 for the definition of an underlying. Only equity underlyings are allowed.

Listing 75: Equity position data

```
<Trade id="EquityPosition">
  <TradeType>EquityPosition</TradeType>
  <Envelope>...</Envelope>
  <EquityPositionData>
    <Quantity>1000</Quantity>
      <Underlying>
        <Type>Equity</Type>
        <Name>BE0003565737</Name>
        <Weight>0.5</Weight>
        <IdentifierType>ISIN</IdentifierType>
        <Currency>EUR</Currency>
        <Exchange>XFRA</Exchange>
      </Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>GB00BH4HKS39</Name>
        <Weight>0.5</Weight>
        <IdentifierType>ISIN</IdentifierType>
        <Currency>GBP</Currency>
        <Exchange>XLON</Exchange>
      </Underlying>
  </EquityPositionData>
</Trade>
```

19.2.37 Equity Option Position

An equity option position represents a position in a single equity option - using a single `Underlying` node, or in a weighted basket of underlying equity options - using multiple `Underlying` nodes.

An Equity Option Position can be used both as a stand alone trade type (TradeType: *EquityOptionPosition*) or as a trade component (`EquityOptionPositionData`) used within the *TotalReturnSwap* (Generic TRS) trade type, to set up for example Equity Option Basket trades.

It is set up using an `EquityOptionPositionData` block as shown in listing 76. The meanings and allowable values of the elements in the block are as follows:

- Quantity: The number of options written on one underlying share resp. the number of units of the option basket held.
  Allowable values: Any positive real number
- Underlying: One or more underlying descriptions, each comprising an `Underlying` block, an `Optiondata` block and a `Strike` element, in that order:
  - Underlying: an underlying description, see 19.3.29, only equity underlying are allowed
  - OptionData: the option description, see 19.3.1, the relevant / allowed data is
    - LongShort: the type of the position,*long* and *Short* positions are allowed. Note that negative weights are allowed. A *long* position with a negative weight results in a *short* position, and a *short* position with a negative weight results in a *long* position.
    - OptionType: *Call* or *Put*
    - Style: *European* or *American*
    - Settlement: *Cash* or *Physical*
    - ExerciseDates: exactly one exercise must be given representing the European exercise date or the last American exercise date
  - Strike: the strike of the option. Allowable values are non-negative real numbers.

If a basket of equities is defined, the `Weight` field should be populated for each underlying. The weighted basket price is then given by

$$\text{Basket-Price} = \text{Quantity} \times \sum_i \text{Weight}_i \times p_i \times \text{FX}_i$$

where

- p_i is the price of the ith option in the basket, written on one underlying share
- FX_i is the FX Spot converting from the ith equity currency to the first equity currency which is by definition the currency in which the npv of the basket is expressed.

Listing 76: Equity Option position data

```
<Trade id="EquityOptionPositionTrade">
  <TradeType>EquityOptionPosition</TradeType>
  <EquityOptionPositionData>
    <!-- basket price = quantity x sum_i ( weight_i x equityOptionPrice_i x fx_i ) -->
    <Quantity>1000</Quantity>
    <!-- option #1 -->
    <Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>.SPX</Name>
        <Weight>0.5</Weight>
        <IdentifierType>RIC</IdentifierType>
      </Underlying>
      <OptionData>
        <LongShort>Long</LongShort>
        <OptionType>Call</OptionType>
        <Style>European</Style>
        <Settlement>Cash</Settlement>
        <ExerciseDates>
          <ExerciseDate>2021-01-29</ExerciseDate>
        </ExerciseDates>
      </OptionData>
      <Strike>3300</Strike>
    </Underlying>
    <!-- option #2 -->
    <Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>.SPX</Name>
        <Weight>0.5</Weight>
        <IdentifierType>RIC</IdentifierType>
      </Underlying>
      <OptionData>
        <LongShort>Long</LongShort>
        <OptionType>Call</OptionType>
        <Style>European</Style>
        <Settlement>Cash</Settlement>
        <ExerciseDates>
          <ExerciseDate>2021-01-29</ExerciseDate>
        </ExerciseDates>
      </OptionData>
      <Strike>3400</Strike>
    </Underlying>
    <!-- option #3 -->
    <!-- ... -->
  </EquityOptionPositionData>
</Trade>
```

19.2.38 CPI Swap

A CPI inflation swap can be set up using the *InflationSwap* trade type, with one leg of type `CPI`. and the other leg(s) can be of any leg type. Listing 77 shows an example. The CPI leg contains an additional `CPILegData` block. See 19.3.17 for details on the CPI leg specification.

Note that Cross Currency Inflation Swaps are supported, as the currencies on the legs of an *InflationSwap* do not need to be the same.

Listing 77: CPI Swap Data (using InflationSwap *trade type)*

```
<InflationSwapData>
  <LegData>
    <LegType>Floating</LegType>
    <Payer>true</Payer>
    ...
  </LegData>
  <LegData>
    <LegType>CPI</LegType>
    <Payer>false</Payer>
    ...
    <CPILegData>
    ...
    </CPILegData>
  </LegData>
</InflationSwapData>
```

Alternatively, a CPI swap can be set up as a swap with trade type *Swap*, with one leg of type `CPI`, see listing 78.

Listing 78: CPI Swap Data (using Swap *trade type)*

```
<SwapData>
  <LegData>
    <LegType>Floating</LegType>
    <Payer>true</Payer>
    ...
  </LegData>
  <LegData>
    <LegType>CPI</LegType>
    <Payer>false</Payer>
    ...
    <CPILegData>
    ...
    </CPILegData>
  </LegData>
</SwapData>
```

19.2.39 Year on Year Inflation Swap

A Year on Year inflation swap can be set up with trade type *Swap*, with one leg of type `YY`. Listing 79 shows an example. The YY leg contains an additional `YYLegData` block. See 19.3.18 for details on the YY leg specification.

Listing 79: Year on Year Swap Data (using Swap *trade type)*

```
<SwapData>
  <LegData>
    <LegType>Floating</LegType>
    <Payer>true</Payer>
    ...
  </LegData>
  <LegData>
    <LegType>YY</LegType>
    <Payer>false</Payer>
    ...
    <YYLegData>
    ...
    </YYLegData>
  </LegData>
</SwapData>
```

Alternatively, a Year on Year inflation swap can be set up using the *InflationSwap* trade type, see Listing 80. The structure of the `InflationSwapData` container is the same as for `SwapData` above.

Listing 80: Year on Year Swap Data (using InflationSwap *trade type)*

```
<InflationSwapData>
  <LegData>
    <LegType>Floating</LegType>
    <Payer>true</Payer>
    ...
  </LegData>
  <LegData>
    <LegType>YY</LegType>
    <Payer>false</Payer>
    ...
    <YYLegData>
    ...
    </YYLegData>
  </LegData>
</InflationSwapData>
```

19.2.40 Bond

A Bond is set up using a `BondData` block, and can be both a stand-alone instrument with trade type *Bond*, or a trade component used by multiple bond derivative instruments.

A Bond can be set up in a short version referencing an underlying bond static, or in a long version where the underlying bond details are specified explicitly, including a full LegData block. The short version is shown in listing 81. The details of the bond are read from the reference data in this case using the SecurityId as a key. The bond trade is fully specified by

- SecurityId: The id identifying the bond.

  Allowable Values: A valid bond identifier, typically the ISIN of the reference bond with the ISIN: prefix, e.g.: `ISIN:XXNNNNNNNNN`

- BondNotional: The notional of the position in the reference bond, expressed in the currency of the bond.

  Allowable Values: Any non-negative real number

- CreditRisk [Optional] Boolean flag indicating whether to show Credit Risk on the Bond product. If set to *false*, the product class will not be set to *Credit*, and there will be no credit sensitivities. However, if the underlying bond reference is set up without a CreditCurveId - typically for some highly rated government bonds - the CreditRisk flag will have no impact on the product class and no credit sensitivities will be shown even if CreditRisk is set to *true*.

  Allowable Values: *true* or *false* Defaults to *true* if left blank or omitted.

in this case.

Listing 81: Bond Data

```
<BondData>
  <SecurityId>ISIN:XS0982710740</SecurityId>
  <BondNotional>100000000.0</BondNotional>
  <CreditRisk>true</CreditRisk>
</BondData>
```

For the long version, the bond details are inlined in the trade as shown in listing 82. The bond specific elements are

- IssuerId [Optional]: A text description of the issuer of the bond. This is for informational purposes and not used for pricing.

  Allowable values: Any string. If left blank or omitted, the bond will not have any issuer description.

- CreditCurveId [Optional]: The unique identifier of the bond. This is used for pricing, and is required for bonds for which a credit - related margin component should be generated, and otherwise left blank. If left blank, the bond (and any bond derivatives using the bond as a trade component) will be plain IR rather than a IR/CR.

  Allowable values: A valid bond identifier, typically the ISIN of the reference bond with the ISIN: prefix, e.g.: `ISIN:XXNNNNNNNNN`

- SecurityId: The unique identifier of the bond. This defines the security specific spread to be used for pricing.

  Allowable values: A valid bond identifier, typically the ISIN of the reference bond with the ISIN: prefix, e.g.: `ISIN:XXNNNNNNNNN`

- ReferenceCurveId: The benchmark curve to be used for pricing. This is typically the main ibor index for the currency of the bond, and if no ibor index is available for the currency in question, a currency-specific benchmark curve can be used.

  Allowable values: For currencies with available ibor indices:
  An alphanumeric string of the form [CCY]-[INDEX]-[TERM]. CCY, INDEX and TERM must be separated by dashes (-). CCY and INDEX must be among the

supported currency and index combinations. TERM must be an integer followed by D, W, M or Y. See Table 26.

For currencies without available ibor indices:
An alphanumeric string, matching a benchmark curve set up in the market data configuration in `todaysmarket.xml` Yield curves section.

Examples: IDRBENCHMARK-IDR-3M, EGPBENCHMARK-EGP-3M, UAHBENCHMARK-UAH-3M, NGNBENCHMARK-NGN-3M

- SettlementDays: The settlement lag in number of business days applicable to the security.

  Allowable values: A non-negative integer.

- Calendar: The calendar associated to the settlement lag.

  Allowable values: See Table 24 Calendar.

- IssueDate: The issue date of the security.

  See `Date` in Table 20.

- PriceQuoteMethod [Optional]: The quote method of the bond. Bond price quotes and historical bond prices (stored as "fixings") follow this method. Also, the initial price for bond total return swaps follows this method. Defaults to PerentageOfPar.

  Allowable values: PercentageOfPar or CurrencyPerUnit

- PriceQuoteBaseValue [Optional]: The base value for quote method = CurrencyPerUnit. Bond price quotes, historical bond prices stored as fixings and initial prices in bond total return swaps are divided by this value. Defaults to 1.0.

  Allowable values: Any real number.

A LegData block then defines the cashflow structure of the bond, this can be of type fixed, floating etc. Note that a LegData block should only be included in the long version.

Listing 82: Bond Data

```
<BondData>
    <IssuerId>Ineos Group Holdings SA</IssuerId>
    <CreditCurveId>ISIN:XS0982710740</CreditCurveId>
    <SecurityId>ISIN:XS0982710740</SecurityId>
    <ReferenceCurveId>EUR-EURIBOR-6M</ReferenceCurveId>
    <SettlementDays>2</SettlementDays>
    <Calendar>TARGET</Calendar>
    <IssueDate>20160203</IssueDate>
    <PriceQuoteMethod>PercentageOfPar</PriceQuoteMethod>
    <PriceQuoteBaseValue>1.0</PriceQuoteBaseValue>
    <LegData>
        <LegType>Fixed</LegType>
        <Payer>false</Payer>
        ...
    </LegData>
</BondData>
```

The bond trade type supports perpetual schedules, i.e. perpetual bonds can be represented by omitting the EndDate in the leg data schedule definition. Only rule based schedules can be used to indicate perpetual schedules.

19.2.41 Bond Position

A bond position represents a position in a weighted basket of underlying bonds.

A bond position can be used both as a stand alone trade type (TradeType: *BondPosition*) or as a trade component (`BondBasketData`) used within the *TotalReturnSwap* (Generic TRS) trade type.

It is set up using an `BondBasketData` block as shown in listing 83. The meanings and allowable values of the elements in the block are as follows:

- Quantity: The number of units of the weighted basket held.
  Allowable values: Any positive real number
- Identifier[Optional]: The identifier of the weighted basket. The Underlying data can be retrieved from the reference data via this identifier, if not given in the trade itself. If the bond basket data is set up in the trade itself in Underlying blocks as in in listing 83, no Identifier is required.
  Allowable values: A string that matches the reference data.
- Underlying[Optional]: One or more underlying descriptions. If bond basket data is set up in the reference data for the given identifier, the underlying data will be populated from there and does not need to be provided in the trade. The weighted basket price is then given by

  $$\text{Basket-Price} = \text{Quantity} \times \sum_i \text{Weight}_i \times B_i \times \text{FX}_i$$

  where

  – B_i is the price of the ith Bond in the basket

  – FX_i is the FX Spot converting from the currency of the ith Bond to the return currency if the BondPosition is in a TotalReturnSwap, otherwise to the currency of the first Bond in the basket.

  Allowable values: See 19.3.29 for the definition of an underlying. Only underlyings of Type *Bond* are allowed.

Listing 83: Bond position data

```
<Trade id="BondPosition">
  <TradeType>BondPosition</TradeType>
  <Envelope>...</Envelope>
  <BondBasketData>
    <Quantity>1000</Quantity>
    <Identifier>ISIN:GB00B4KT9Q30</Identifier>
    <Underlying>
      <Type>Bond</Type>
      <Name>US69007TAB08</Name>
      <IdentifierType>ISIN</IdentifierType>
      <Weight>0.5</Weight>
      <BidAskAdjustment>-0.0025</BidAskAdjustment>
    </Underlying>
    <Underlying>
      <Type>Bond</Type>
      <Name>US750236AW16</Name>
      <IdentifierType>ISIN</IdentifierType>
      <Weight>0.5</Weight>
      <BidAskAdjustment>-0.005</BidAskAdjustment>
    </Underlying>
  </BondBasketData>
</Trade>
```

19.2.42 Forward Bond

A Forward Bond (or Bond Forward) is a contract that establishes an agreement to buy or sell (determined by `LongInForward`) an underlying bond at a future point in time (the `ForwardMaturityDate`) at an agreed price (the settlement `Amount`).

A T-Lock is a Forward Bond with a US Treasury Bond as underlying, whereas a J-Lock is a Forward Bond with a Japanese Government Bond as underlying. T-Locks can be specified in terms of a lock-in yield rather then a settlement amount. The cash settlement amount is given by (bond yield at maturity - lock rate) x DV01 in this case.

Listing 84 shows an example for a physically settled forward bond. Listing 85 shows an example for a cash settled T-Lock transaction specified by a lock-in yield.

A Forward Bond is set up using a `ForwardBondData` block as shown below and the trade type is *ForwardBond*. The specific elements are

- BondData: A `BondData` block specifying the underlying bond as described in section 19.2.40. A long position must be taken in the bond, i.e. (`Payer`) flag must be set to (`true`). The bond data block contains additional fields for forward bonds
  - IncomeCurveId: The benchmark curve to be used for compounding, this must match a name of a curve in the yield curves or index curve block in `todaysmarket.xml`. It is optional to provide this curve. If left out the market reference yield curve from `todaysmarket.xml` is used for compounding.
- SettlementData: The entity defining the terms of settlement:
  - ForwardMaturityDate: The date of maturity of the forward contract. Allowable values: See `Date` in Table 20.

  - Settlement [Optional]: Cash or Physical. Option, defaults to Physcial, except in case the settlement is defined by LockRate, in which case it defaults to Cash.
    Allowable values: Cash, Physical
  - Amount [Optional]: The settlement amount (also called strike) transferred at forward maturity in return for the bond (physical delivery) or a cash amount equal to the dirty price of the bond (cash settlement). This is transferred from the party that is long to the party that is short (determined by `LongInForward`) and cannot be a negative amount. It is assumed to be in the same currency as the underlying bond. Exactly one of the fields Amount, LockRate must be given.
    Allowable values: Any non-negative real number.
  - LockRate [Optional]: The payoff is given by (yield at forward maturity - LockRate) x DV01 (LongInForward = true). Exactly one of the fields Amount, LockRate must be given. In case the LockRate is given, the Settlement must be set to Cash. If Settlement is not given, it defaults to Cash in this case.
    Allowable values: Any non-negative real number.
  - LockRateDayCounter [Optional]: The day counter w.r.t. which the lock rate is expressed. Optional, defaults to A360.
    Allowable values: see table 25
  - SettlementDirty [Optional]: A flag that determines whether the settlement amount (`Amount`) reflects a clean (*false*) or dirty (*true*) price. In either case, the dirty amount is actually paid on the forward maturity date, i.e. if SettlementDirty = *false*, the (forward) accruals are computed internally and added to the given amount to get the actual settlement amount. Optional, defaults to true.
    Allowable values: *true*, *false*

- PremiumData: The entity defining the terms of a potential premium payment. This node is optional. If left out it is assumed that no premium is paid.
  - Date: The date when a premium is paid.
    Allowable values: See `Date` in Table 20.
  - Amount: The amount transferred as a premium. This is transferred from the party that is long to the party that is short (determined by `LongInForward`) and cannot be a negative amount. It is assumed to be in the same currency as the underlying bond.
    Allowable values: Any non-negative real number.

- LongInForward: A flag that determines whether the forward contract is entered in long (*true*) or short (*false*) position.
  Allowable values: *true*, *false*

Listing 84: Forward Bond Data

```
<ForwardBondData>
  <BondData>
   ...
   <IncomeCurveId>BENCHMARKINCOME-EUR<IncomeCurveId>
  </BondData>
  <SettlementData>
    <ForwardMaturityDate>20160808</ForwardMaturityDate>
    <Settlement>Physcial</Settlement>
    <ForwardSettlementDate>20160810</ForwardSettlementDate>
    <Amount>1000000.00</Amount>
    <SettlementDirty>true</SettlementDirty>
  </SettlementData>
  <PremiumData>
    <Amount>1000.00</Amount>
    <Date>20160808</Date>
  </PremiumData>
  <LongInForward>true</LongInForward>
</ForwardBondData>
```

Listing 85: Forward Bond Date (T-Lock)

```
<ForwardBondData>
  <BondData>
   ...
  </BondData>
  <SettlementData>
    <ForwardMaturityDate>20160808</ForwardMaturityDate>
    <ForwardSettlementDate>20160810</ForwardSettlementDate>
    <LockRate>0.02365</LockRate>
  </SettlementData>
  <LongInForward>true</LongInForward>
</ForwardBondData>
```

As for the ordinary bond the forward bond pricing requires a recovery rate that can be specified in VRE per SecurityId.

Forward Bond - Pricing Engine configuration

The configuration for the pricing engine of the forward bond is identical to the ordinary bond.The pricing engine called by forward bond products is the `DiscountingForwardBondEngine`, see below for a configuration example.

```
<Product type="ForwardBond">
<Model>DiscountedCashflows</Model>
<ModelParameters></ModelParameters>
<Engine>DiscountingForwardBondEngine</Engine>
<EngineParameters>
```

```
 <Parameter name="TimestepPeriod">3M</Parameter>
</EngineParameters>
</Product>
```

19.2.43 Bond Forward / T-Lock / J-Lock (using ref. data)

A Forward Bond (or Bond Forward) is a contract that establishes an agreement to buy or sell (determined by `LongInForward`) an underlying bond at a future point in time (the `ForwardMaturityDate`) at an agreed price (the settlement `Amount`).

A T-Lock is a Forward Bond with a US Treasury Bond as underlying, whereas a J-Lock is a Forward Bond with a Japanese Government Bond as underlying. T-Locks can be specified in terms of a lock-in yield rather then a settlement amount. The cash settlement amount is given by (bond yield at maturity - lock rate) x DV01 in this case.

Listing 86 shows an example for a physically settled forward bond. Listing 87 shows an example for a cash settled T-Lock transaction specified by a lock-in yield.

A Forward Bond is set up using a `ForwardBondData` block as shown below and the trade type is *ForwardBond*. The specific elements are

- The `BondData` block specifies the underlying bond, see below for more details.
  - SecurityId: The underlying security identifier
    Allowable values: Typically the ISIN of the underlying bond, with the ISIN: prefix.
  - BondNotional: The notional of the underlying bond on which the forward is written expressed in the currency of the bond
    Allowable values: Any positive real number.
  - CreditRisk [Optional] Boolean flag indicating whether to show Credit Risk on the Bond product. If set to *false*, the product class will be set to *RatesFX* instead of *Credit*, and there will be no credit sensitivities. Note that if the underlying bond reference is set up without a CreditCurveId - typically for some highly rated government bonds - the CreditRisk flag will have no impact on the product class and no credit sensitivities will be shown even if CreditRisk is set to *true*.
    Allowable Values: *true* or *false* Defaults to *true* if left blank or omitted.
- SettlementData: The entity defining the terms of settlement:
  - ForwardMaturityDate: The date of maturity of the forward contract.
    Allowable values: See `Date` in Table 20.
  - ForwardSettlementDate [Optional]: Settlement date for forward bond or cash settlement payment date.
    Allowable values: See `Date` in Table 20.
  - Settlement [Optional]: Cash or Physical. Option, defaults to Physcial, except in case the settlement is defined by LockRate, in which case it defaults to Cash.
    Allowable values: Cash, Physical
  - Amount [Optional]: The settlement amount (also called strike) transferred at forward maturity in return for either:

(a) the bond (physical delivery) or
(b) a cash amount equal to the dirty price of the bond (cash settlement).
This is transferred from the party that is long to the party that is short (determined by `LongInForward`) and cannot be a negative amount. It is assumed to be in the same currency as the underlying bond. Exactly one of the fields Amount, LockRate must be given.
Allowable values: Any non-negative real number.

- LockRate [Optional]: The payoff is given by (yield at forward maturity - LockRate) x DV01 (LongInForward = true). Exactly one of the fields Amount, LockRate must be given. In case the LockRate is given, the Settlement must be set to Cash. If Settlement is not given, it defaults to Cash in this case.
  Allowable values: Any non-negative real number. The LockRate is expressed in decimal form, eg 0.05 is a rate of 5%
- dv01 [Optional]: When the LockRate is given, it is possible to implement a contractual DV01 instead of deriving it from the bond price.
  Allowable values: Any positive real number. E.G If the dPdY is given then dv01=10000*dPdY/N.
- LockRateDayCounter [Optional]: The day counter w.r.t. which the lock rate is expressed. Optional, defaults to A360.
  Allowable values: see table 25
- SettlementDirty [Optional]: A flag that determines whether the settlement amount (`Amount`) reflects a clean (*false*) or dirty (*true*) price. In either case, the dirty amount is actually paid on the forward maturity date, i.e. if SettlementDirty = *false*, the (forward) accruals are computed internally and added to the given amount to get the actual settlement amount. Optional, defaults to true.
  Allowable values: *true*, *false*

- PremiumData: The entity defining the terms of a potential premium payment. This node is optional. If left out it is assumed that no premium is paid.
  - Date: The date when a premium is paid.
    Allowable values: See `Date` in Table 20.
  - Amount: The amount transferred as a premium. This is transferred from the party that is long to the party that is short (determined by `LongInForward`) and cannot be a negative amount. It is assumed to be in the same currency as the underlying bond.
    Allowable values: Any non-negative real number.
- LongInForward: A flag that determines whether the forward contract is entered in long (*true*) or short (*false*) position.
  Allowable values: *true*, *false*

Listing 86: Forward Bond Data

```
<ForwardBondData>
  <BondData>
    <SecurityId>ISIN:XS1234567890</SecurityId>
    <BondNotional>100000</BondNotional>
  <BondData>
  <SettlementData>
    <ForwardMaturityDate>20160808</ForwardMaturityDate>
    <Settlement>Physcial</Settlement>
    <ForwardSettlementDate>20160810</ForwardSettlementDate>
    <Amount>1000000.00</Amount>
    <SettlementDirty>true</SettlementDirty>
  </SettlementData>
  <PremiumData>
    <Amount>1000.00</Amount>
    <Date>20160808</Date>
  </PremiumData>
  <LongInForward>true</LongInForward>
</ForwardBondData>
```

Listing 87: Forward Bond Date (T-Lock)

```
<ForwardBondData>
  <BondData>
    <SecurityId>ISIN:XS1234567890</SecurityId>
    <BondNotional>100000</BondNotional>
  </BondData>
  <SettlementData>
    <ForwardMaturityDate>20160808</ForwardMaturityDate>
    <ForwardSettlementDate>20160810</ForwardSettlementDate>
    <LockRate>0.02365</LockRate>
  </SettlementData>
  <LongInForward>true</LongInForward>
</ForwardBondData>
```

Listing 88: Forward Bond Date (T-Lock) with DV01

```
<ForwardBondData>
<BondData>
  <SecurityId>ISIN:XS1234567890</SecurityId>
  <BondNotional>100000</BondNotional>
</BondData>
<SettlementData>
  <ForwardMaturityDate>20160808</ForwardMaturityDate>
  <ForwardSettlementDate>20160810</ForwardSettlementDate>
  <LockRate>0.02365</LockRate>
  <dv01>0.8</dv01>
</SettlementData>
<LongInForward>true</LongInForward>
</ForwardBondData>
```

19.2.44 Bond Repo

In a bond repo transaction one party A receives a cash amount from a party B for a specified period. At the maturity of the trade party A pays back the cash amount plus accrued interest to party B. Intermediate interest payments are also possible. Party A delivers a bond to party B as a collateral for the received cash amount for the duration of the trade. In exchange the interest to be paid by party A will be lower than for an uncollateralised borrowing transaction.

A bond repo trade is set up using the trade type `BondRepo` and a `BondRepoData` block as shown in listing 89. The block contains two nodes

- `BondData`, which specifies the underlying bond and its quantity, and
- `RepoData`, which specifies the cash leg of the repo

The `BondData` block contains the following fields

- SecurityId: The identified of the underlying security.
  Allowable values: A valid key, usually of the form "ISIN::XY012345679"
- BondNotional: The notional of the underlying bond. This is the effective notional used as collateral, i.e. it should include hair cuts. Usually the number Bond Notional x Bond Dirty Price x (1 - Haircut) will correspond to the nominal on the cash leg at trade inception.
  Allowable values: Any positive real number.
- CreditRisk [Optional] Boolean flag indicating whether to show Credit Risk on the Bond product.
  Allowable Values: *true* or *false* Defaults to *true* if left blank or omitted.

In this case the details of the underlying bond is read from the reference data. It is also possible to inline the details in the trade, see 19.2.40 for more details on this.

The `RepoData` block contains exactly one `LegData` subnode that describes the payments on the cash leg of the repo, see 19.3.3 for details on how to set this up. The `Payer` leg determines whether interest is paid (regular repo) or received (reversed repo).

Listing 89: Bond Repo Data

```
<BondRepoData>
  <BondData>
    <SecurityId>ISIN:US912828X703</SecurityId>
    <BondNotional>27807597.777444</BondNotional>
  </BondData>
  <RepoData>
    <LegData>
      <LegType>Fixed</LegType>
      <Payer>true</Payer>
      <Currency>USD</Currency>
      <Notionals>
        <Notional>28371510.00</Notional>
      </Notionals>
      <ScheduleData>
        <Rules>
          <StartDate>2020-01-06</StartDate>
          <EndDate>2020-04-07</EndDate>
          <Tenor>1Y</Tenor>
          <Calendar>US</Calendar>
          <Convention>MF</Convention>
          <TermConvention>MF</TermConvention>
          <Rule>Forward</Rule>
          <EndOfMonth/>
          <FirstDate/>
          <LastDate/>
        </Rules>
      </ScheduleData>
      <DayCounter>A360</DayCounter>
      <PaymentConvention>F</PaymentConvention>
      <FixedLegData>
        <Rates>
          <Rate>0.0178</Rate>
        </Rates>
      </FixedLegData>
    </LegData>
  </RepoData>
</BondRepoData>
```

19.2.45 Bond Option

A bond option provides the buyer with the right, but not the obligation, to buy or sell a given bond at a fixed price either at or before a specific date. Options are written on government bonds and are traded on an OTC basis.

The structure of a trade node representing a *BondOption* is shown in listing 90:

- The `BondOptionData` node is the trade data container for the option part of a bond option trade type. Vanilla bond options are supported, the exercise style must be *European*. The `BondOptionData` node includes one and only one `OptionData` trade component sub-node plus elements specific to the bond option.
- The latter also includes the underlying Bond description in the `BondData` node, see section 19.2.40, listing 82 for details

Listing 90: Bond Option data

```
<Trade id="...">
  <TradeType>BondOption</TradeType>
  <Envelope>
      ...
  </Envelope>
  <BondOptionData>
    <OptionData>
        ...
    </OptionData>
    <StrikeData>
      <StrikePrice>
        <Value>11809123.56</Value>
        <Currency>EUR</Currency>
      </StrikePrice>
    </StrikeData>
    <Redemption>100.00</Redemption>
    <PriceType>Dirty</PriceType>
    <KnocksOut>false</KnocksOut>
    <BondData>
       <VolatilityCurveId>YieldVols-EUR</VolatilityCurveId>
        ...
    <BondData>
  </BondOptionData>
</Trade>
```

The meanings and allowable values of the elements in the `BondOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1 Option Data. Note that the bond option type allows for *European* option style only.
- StrikeData: A node containing the strike information. Allowable values: Supports `StrikePrice` and `StrikeYield` as described in Section 19.3.30.
- Redemption: Redemption ratio in percent
- PriceType: This node defines which strike should be used for the pricing. If the node takes the value Dirty, the strike price should be set equal to the value of the Strike node. If the node takes the value Clean, the strike price should be set equal to the value of the Strike node plus accrued interest at the expiration date of the option.
  Allowable values: Dirty or Clean.
- KnocksOut: If true the option knocks out if the underlying defaults before the option expiry, if false the option is written on the recovery value in case of a default of the bond before the option expiry

The meanings and allowable values of the elements in the `BondData` are:

- VolatilityCurveId: The yield volatility curve to use for the valuation of this bond option.

19.2.46 Bond Option (using bond reference data)

The structure of a trade node representing a *BondOption* is shown in listing 91:

- The `BondOptionData` node is the trade data container for the option part of a bond option trade type. Vanilla bond options are supported, the exercise style must be *European*. The `BondOptionData` node includes one and only one `OptionData` trade component sub-node plus elements specific to the bond option.
- The latter also includes the underlying Bond description in the `BondData` node, see below for details

Note that only par redemption vanilla bonds are supported.

Listing 91: Bond Option data using bond reference data

```
<Trade id="...">
  <TradeType>BondOption</TradeType>
  <Envelope>
      ...
  </Envelope>
  <BondOptionData>
    <OptionData>
     <LongShort>Long</LongShort>
     <OptionType>Call</OptionType>
     <Style>European</Style>
     <ExerciseDates>
      <ExerciseDate>20210203</ExerciseDate>
     </ExerciseDates>
        ...
    </OptionData>
    <StrikeData>
      <StrikePrice>
        <Value>1.23</Value>
      </StrikePrice>
    </StrikeData>
    <PriceType>Dirty</PriceType>
    <KnocksOut>false</KnocksOut>
    <BondData>
       <SecurityId>ISIN:XS1234567890</SecurityId>
       <BondNotional>100000</BondNotional>
    <BondData>
  </BondOptionData>
</Trade>
```

The meanings and allowable values of the elements in the `BondOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1 Option Data.

  The relevant fields in the `OptionData` node for a BondOption are:

  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*. For option type *Call*, the Bond Option holder has the right to buy the underlying Bond at the strike price. For option type *Put*, the Bond Option holder has the right to sell the underlying Bond at the strike price.

  - `Style` The allowable value is *European* only.
  - `Settlement` [Optional] The allowable values are *Cash* or *Physical*, but this field is currently ignored.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - Premiums [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2

- `StrikeData`: A `StrikeData` node is used as described in Section 19.3.30 to represent the Bond Option strike price or strike yield. If StrikePrice is used, the strike price (`Value` field) is expressed per unit notional, i.e. a strike of 101% of the bond notional is expressed as 1.01. If StrikeYield is used, the `Yield` is quoted in decimal form, e.g. 5% should be entered as 0.05.
- PriceType [Mandatory for StrikePrice, no impact for StrikeYield]:
  The payoff for a bond option is

  max(B - X, 0)

  where B is always the dirty NPV of the underlying bond on the exercise settlement date. If `PriceType` is *Clean*, X is (Strike + Underlying Bond Accruals) x BondNotional If `PriceType` is *Dirty*, X is Strike x BondNotional

  Allowable values: *Dirty* or *Clean*. If the `StrikeData` node uses StrikeYield, `PriceType` can be omitted as it is not relevant in the yield case.
- KnocksOut: If *true* the option knocks out if the underlying defaults before the option expiry, if *false* the option is written on the recovery value in case of a default of the bond before the option expiry.

  Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36.

The meanings and allowable values of the elements in the `BondData` are:

- SecurityId: The underlying security identifier

  Allowable values: Typically the ISIN of the underlying bond, with the ISIN: prefix.
- BondNotional: The notional of the underlying bond on which the option is written expressed in the currency of the bond.

  Allowable values: Any positive real number.
- CreditRisk [Optional] Boolean flag indicating whether to show Credit Risk on the Bond product.

  Allowable Values: *true* or *false* Defaults to *true* if left blank or omitted.

19.2.47 Bond Total Return Swap

A vanilla Bond Total Return Swap (Trade type: *BondTRS*) is set up using a `BondTRSData` block as shown in listing 93. The block is comprised of three sub-blocks, which are

`BondData`, `TotalReturnData` and `FundingData`.

- The `BondData` block specifies the underlying bond, usually by specifiying the security id and the quantity / bond notional and relying on reference data:
  - SecurityId: The underlying security identifier
    Allowable values: Typically the ISIN of the underlying bond, with the ISIN: prefix.
  - BondNotional: The quantity or number of bonds that is relevant for the TRS, with the convention that 1 bond always corresponds to a face value of 1 unit of bond currency.
    Allowable values: Any positive real number.
  - CreditRisk [Optional] Boolean flag indicating whether to show Credit Risk on the Bond product. If set to *true*, the product class will be set to *Credit* instead of *RatesFX*, and there will be credit sensitivities. Note that if the underlying bond reference is set up without a CreditCurveId - typically for some highly rated government bonds - the CreditRisk flag will have no impact on the product class and no credit sensitivities will be shown even if CreditRisk is set to *true*.
    Allowable Values: *true* or *false* Defaults to *true* if left blank or omitted.

  Alternatively, the BondData block can be specified fully explicit, as outlined in 19.2.40
- The `TotalReturnData` block specifies
  - Payer: Indicates whether the total return leg is paid.
    Allowable values: *true* or *false*
  - InitialPrice [Optional]: Should be filled if the bond price on the first date of the total return schedule is contractually given, in which case the price must correspond to the price type of the total return leg, i.e. if the price type is *Dirty* then the InitialPrice must also be a dirty price (as it is usually given in the term sheet in this case). The price must given in percent, e.g. 101.20.[4] If not given, the bond price for the first date of the total return schedule is read from the price history. Notice that if a bond is quoted in Currency per Unit the initial price should be given in this format too: If e.g. one unit is 50.0 USD an initial price of 51.0 would correspond a dirty amount of 51.0 USD for one unit of the bond.
    Allowable values: Any positive real number.
  - PriceType: The price type on which these payments are based
    Allowable values: *Dirty* or *Clean*
  - ObservationLag [Optional]: The lag between the valuation date and the reference schedule period start date.

    Allowable values: Any valid period, i.e. a non-negative whole number, followed by *D* (days), *W* (weeks), *M* (months), *Y* (years). Defaults to *0D* if left blank or omitted.
  - ObservationConvention [Optional]: The roll convention to be used when applying the observation lag.

[4] as opposed to the bond price in the fixing history, where it must be given as 1.0120 and is always a clean quotation

Allowable values: A valid roll convention (*F, MF, P, MP, U, NEAREST*), see Table 21 Roll Convention. Defaults to *U* if left blank or omitted.

– ObservationCalendar [Optional]: The calendar to be used when applying the observation lag.

  Allowable values: Any valid calendar, see Table 24 Calendar. Defaults to the *NullCalendar* (no holidays) if left blank or omitted.

– PaymentLag [Optional]: The lag between the reference schedule period end date and the payment date.

  Allowable values: Any valid period, i.e. a non-negative whole number, optionally followed by *D* (days), *W* (weeks), *M* (months), *Y* (years). Defaults to *0D* if left blank or omitted. If a whole number is given and no letter, it is assumed that it is a number of *D* (days).

– PaymentConvention [Optional]: The business day convention to be used when applying the payment lag.

  Allowable values: A valid roll convention (*F, MF, P, MP, U, NEAREST*), see Table 21 Roll Convention. Defaults to *U* if left blank or omitted.

– PaymentCalendar [Optional]: The calendar to be used when applying the payment lag.

  Allowable values: Any valid calendar, see Table 24 Calendar. Defaults to the *NullCalendar* (no holidays) if left blank or omitted.

– PaymentDates [Optional]: This node allows for the specification of a list of explicit payment dates, using `PaymentDate` elements. The list must contain exactly $n - 1$ dates where n is the number of dates in the reference schedule given in the ScheduleData node. See Listing 92 for an example with an assumed ScheduleData with 4 dates.

Listing 92: Payment dates

```
<PaymentDates>
      <PaymentDate>2020-01-15</PaymentDate>
      <PaymentDate>2021-01-15</PaymentDate>
      <PaymentDate>2022-01-17</PaymentDate>
</PaymentDates>
```

– FXTerms [Mandatory when underlying bond and BondTRS currencies differ]: Required if the bond currency is different from the return currency, which is always assumed to be equal to the funding leg currency. This kind of trade is also known as a "composite trs". The subnode for the FXTerms node is:

  * FXIndex: The fx index to use for the conversion, this must contain the bond currency and the funding leg currency (in the order defined in table 28, i.e. it does not matter which one is the bond currency and which is the funding currency)

    Allowable values: See Table 28

- ScheduleData: The reference schedule for the return leg, where the valuation dates are derived from this schedule using the ObservationLag, ObservationConvention and ObservationCalendar fields. The payment dates are derived from this schedule using the PaymentLag, PaymentConvention and PaymentCalendar fields. The payment dates can also be given as an explicit list in the PaymentDates node. Allowable values: A `ScheduleData` block as defined in section 19.3.4
- PayBondCashFlowsImmediately [Optional]: If true, bond cashflows like coupon or amortisation payments are paid when they occur. If false, these cashflows are paid together with the next return payment. If omitted, the default value is false.

  Allowable values: *true* (immediate payment of bond cashflows) or *false* (bond cashflows are paid on the next return payment date)

- The `FundingData` block specifies the funding leg, which can be of any leg type. The `FundingData` contains exactly one `Leg`. The currency of this leg also defines the currency in which the return is paid. Usually the funding leg's notional will be aligned with the return leg's notional. To achieve this, indexings on the floating leg can be used, see 19.3.8. In the context of bond total return swaps, the indexings can be defined in a simplified way by adding an Indexings node with a subnode FromAssetLeg set to true to the funding leg's LegData node. The notionals node is not required either in the funding leg's LegData in this case. An example for this setup is shown in 93.

Listing 93: Bond Total Return Swap Data with indexed funding leg

```
<BondTRSData>
  <BondData>
    <SecurityId>ISIN:NZIIBDT005C5</SecurityId>
    <BondNotional>100000</BondNotional>
  </BondData>
  <TotalReturnData>
    <Payer>false</Payer>
    <InitialPrice>102.0</InitialPrice>
    <PriceType>Clean</PriceType>
    <ObservationLag>0D</ObservationLag>
    <ObservationConvention>P</ObservationConvention>
    <ObservationCalendar>USD</ObservationCalendar>
    <PaymentLag>2D</PaymentLag>
    <PaymentConvention>F</PaymentConvention>
    <PaymentCalendar>TARGET</PaymentCalendar>
    <!-- <PaymentDates> -->
    <!--   <PaymentDate> ... </PaymentDate> -->
    <!--   <PaymentDate> ... </PaymentDate> -->
    <!-- </PaymentDates> -->
    <FXTerms>
      <FXIndex>FX-TR20H-NZD-USD</FXIndex>
    </FXTerms>
    <ScheduleData>
    ...
    </ScheduleData>
    <PayBondCashFlowsImmediately>false</PayBondCashFlowsImmediately>
  </TotalReturnData>
  <FundingData>
    <LegData>
      <Payer>true</Payer>
      <LegType>Floating</LegType>
      <Currency>USD</Currency>
      ...
      <!-- Notionals node is not required, set to 1 internally -->
      ...
      <Indexings>
      <!-- derive the indexing information (bond price, FX) from the total return leg -->
      <FromAssetLeg>true</FromAssetLeg>
      </Indexings>
      ...
    </LegData>
  </FundingData>
</BondTRSData>
```

19.2.48 Convertible Bond

A convertible bond is set up in VRE using a `ConvertibleBondData` block as shown in listing 94. The bond details are read from reference data in this case.

A convertible bond is a bond, that can be converted into a prespecified number of shares, given by:

$$NumberOfShares = \frac{BondNotional}{ConversionRatio}$$

Where the Conversion Ratio is specified in the underlying bond reference data.

The shares are usually from the bond issuer, but it is also possible that the shares are from a different issuer (exchangeables). In addition, the share currency can be different from the bond currency in both cases (cross-currency convertibles).

The bond might be callable by the issuer (typically in American style) and / or puttable by the investor (typically in Bermudan style). The issuer calls can be "hard calls", which are call rights in the traditional sense, as opposed to "soft calls" which can only the exercised if the equity price observed on the exercise date is above a prespecified threshold given by TriggerRatios. If a soft call is exercised, the investor has the right to convert the bond into shares instead of accepting the payment from the issuer call ("forced conversion").

The meanings and allowable values of the elements in the `ConvertibleBondData` block are as follows:

- SecurityId: The underlying security identifier
  Allowable values: Typically the ISIN of the underlying bond, with the ISIN: prefix.
- BondNotional: The notional of the underlying bond expressed in the currency of the bond.
  Allowable values: Any positive real number.
- CreditRisk [Optional] Boolean flag indicating whether to show Credit Risk on the Bond product.
  Allowable Values: *true* or *false* Defaults to *true* if left blank or omitted.

Listing 94: Convertible bond set up using reference data

```
<Trade id="ConvertibleBond">
  <TradeType>ConvertibleBond</TradeType>
  <Envelope>...</Envelope>
  <ConvertibleBondData>
    <BondData>
      <SecurityId>ISIN:XS0451905367</SecurityId>
      <BondNotional>1000000.00</BondNotional>
    </BondData>
  </ConvertibleBondData>
</Trade>
```

Alternatively the bond can be set up with further explicit details using the blocks as shown in listing 95. All fields that are not given in the trade XML are filled up with the information from the reference data if available in the reference data. In other words, if reference data is given, the trade xml can still be used to overwrite the information partially, if this seems appropriate. The meanings and allowable values of the elements in the block are as follows:

- BondData: The vanilla part of the bond, see 19.2.40.
- CallData: The call terms of the bond, as described below. Optional, if not given, no calls are present.
- PutData: The put terms of the bond, as described below. Optional, if not given, no puts are present.
- ConversionData: The conversion terms of the bond, as described below. This node must always be given, even if no conversion rights are present (in which case an empty

conversion date list can be used).

- DividendProtectionData: The dividend protection terms of the bond, as described below. Optional, if not given, no dividend prtection is present.
- Detachable: If true, the trade represents the embedded optionality, i.e. the difference between the full convertible bond and the bond floor. Optional, defaults to false. Allowable values: true, false

The convertible bond trade type supports perpetual schedules, i.e. perpetual convertible bonds can be represented by omitting the EndDate in the following schedules to indicate perpetual schedules. Only rule based schedules can be used to indicate perpetual schedules.

- BondData / LegData: Omitting the EndDate in this schedule indicates that the underlying bond runs perpetually.
- CallData: Omitting the EndDate in this schedule indicates perpetual call dates. For American call dates, where only two dates have to be specified (start and end date of the american call window), a rule based schedule with Tenor = 0D, Rule = Zero and without EndDate can be used to indicate an end date infinitely far away in the future.
- PutData: Same as CallData.
- ConversionData: Omitting the EndDate in this schedule indicates perpetual conversion rights. For American rights, the same comment as under CallData applies.
- ConversionData / ConversionResets: Omitting the EndDate in this schedule indicates perpetual conversion resets.
- DividendProtectionData: Omitting the EndDate in this schedule indicates a perpetual dividend protection schedule.

Listing 95: Convertible bond set up using the detail blocks

```
<Trade id="ConvertibleBond">
  <TradeType>ConvertibleBond</TradeType>
  <Envelope>...</Envelope>
  <ConvertibleBondData>
    <BondData> ... </BondData>
    <CallData> ... </CallData>
    <PutData> ... </PutData>
    <ConversionData> ... </ConversionData>
    <DividendProtectionData> ... </DividendProtectionData>
    <Detachable>false</Detachable>
  </ConvertibleBondData>
</Trade>
```

Specification of CallData / PutData:

All lists specified in subnodes (except the date list itself of course) can be specified as either an explicit list of values corresponding to the schedule dates list or using the attribute `startDate`. An explicit value list can be shorter than the list of dates, in which case the last value from the list is associated to the remaining dates.

See listings 96,97,98,99,100,101,102 for examples of exercise schedules.

- Styles: A list of the exercise styles. Notice that Bermudan is used to define European exercises as well, namely as a Bermudan exercise with a single exercise date. The attribute `startDate` can be used to specify the list.
  Allowable values: American, Bermudan
- ScheduleData: A schedule of exercise dates (for Bermudan exercises) or start / end dates (for American exercises)
  Allowable values: see 19.3.4.
- Prices: A list of exercise prices in relative terms, i.e. if the price is 1.02 then the amount paid on the exercise is this price times the current notional of the bond (plus accrued interest, if the price type is clean, see below). The attribute `startDate` can be used to specify the list.
  Allowable values: Any positive real number.
- PriceType: A list of the flavour in which the exercise prices are given. The attribute `startDate` can be used to specify the list.
  Allowable values: Clean, Dirty.
- IncludeAccrual: A list of flags specifying whether accruals have to be paid on exercise. This is independent of the quoting style of the exercise prices (PriceType).
  Allowable values: true, false
- Soft: A list of flags specifying whether the call is soft (true) or hard (false). The attribute `startDate` can be used to specify the list. Optional, defaults to false. Only applicable to Calls, not to Puts. Optional, if not given, false is assumed, i.e. hard calls. If soft calls are specified, at least one conversion exercise date with corresponding conversion rate must be defined under ConversionData.
  Allowable values: true, false
- TriggerRatios: A list of trigger ratios T for soft calls. A soft call can be executed only if the equity price on the exercise date is above the Conversion Price (defined below) times the trigger ratio, i.e. $S_t > C_t^P T$. Only applicable to Calls, not to Puts. Required for soft calls, can be omitted otherwise.

  $$ConversionPrice, C_t^P = \frac{1}{ConversionRatio}$$

  For cross-currency trades the conversion price is usually quoted in equity ccy, i.e.

  $$ConversionPrice, C_t^P = \frac{1}{ConversionRatio \cdot X_t}$$

  where X_t converts one equity ccy unit to bond ccy

  Allowable values: Any positive real number.
- NOfMTriggers: A list of n-of-m trigger specifications for calls, i.e. the soft-call trigger defined by TriggerRatios must be observed on n of the m calendar days in the period before (and including) a call date. Only applicable to Calls, not to Puts. Optional, defaults to "1-of-1"
  Allowable values: x-of-y with x, y non-negative integers, "1-of-1" corresponds to a vanilla call specification

- MakeWhole: A list of make whole conditions. Optional. Possible subnodes are:
  - ConversionRatioIncrease: In case of a call exercise, the conversion ratio (applicable in case of a forced conversion) is adjusted upwards. The adjustment is additive, i.e. if the current conversion ratio is CR the conversion ratio applicable in case of a forced conversion will be $CR + d$ where d is interpolated from a matrix of effective dates (rows) and stock prices (columns). The conversion rate adjustment might be capped by a prespecified rate. If the exercise date / stock price lies outside the matrix, d is zero, i.e. no adjustment is made. Notice that a soft call trigger is checked w.r.t. CR, i.e. the unadjusted conversion ratio.
    - Cap: An upper bound for the adjusted conversion ratio. Optional, if not given, no cap will be applied.
      Allowable values: Any non-negative real number.
    - StockPrices: A comma separated list of stock prices defining the interpolation grid's x values. At least two stock prices must be given.
      Allowable values: A list of non-negative real numbers.
    - CrIncreases: A node that contains at least two subnodes CrIncrease. Each subnode must have an attribute startDate defining the effective date of the adjustment and a list of conversion ratio adjustments d. The number of adjustments must match the number of prices given in the StockPrices node.
      Allowable values: A list of non-negative real numbers.

Listing 96: Convertible bond call data example 1

```
<!-- Bermudan issuer call on three dates at a clean price of 100 (hard calls),
     accruals are paid on exercise -->
<CallData>
  <Styles>
    <Style>Bermudan</Style>
  </Styles>
  <ScheduleData>
    <Dates>
      <Dates>
        <Date>2016-08-03</Date>
        <Date>2017-08-03</Date>
        <Date>2018-08-03</Date>
      </Dates>
    </Dates>
  </ScheduleData>
  <Prices>
    <Price>1.00</Price>
  </Prices>
  <PriceTypes>
    <PriceType>Clean</PriceType>
  </PriceTypes>
  <IncludeAccruals>
    <IncludeAccrual>true</IncludeAccrual>
  </IncludeAccruals>
  <Soft>
    <Soft>false</Soft>
  </Soft>
  <TriggerRatios/>
  <NOfMTriggers>
    <NOfMTrigger>20-of-30</NOfMTrigger>
  </NOfMTriggers>
</CallData>
```

Listing 97: Convertible bond call data example 2

```
<!-- Bermudan issuer call on three dates at a clean price of 101, 102 and 103,
     soft calls with trigger ratio of 0.8, 0.85, 0.9,
     accrual are _not_ paid on exercise -->
<CallData>
  <Styles>
    <Style>Bermudan</Style>
  </Styles>
  <ScheduleData>
    <Dates>
      <Dates>
        <Date>2016-08-03</Date>
        <Date>2017-08-03</Date>
        <Date>2018-08-03</Date>
      </Dates>
    </Dates>
  </ScheduleData>
  <Prices>
    <Price>1.01</Price>
    <Price>1.02</Price>
    <Price>1.03</Price>
  </Prices>
  <PriceTypes>
    <PriceType>Clean</PriceType>
  </PriceTypes>
  <IncludeAccruals>
    <IncludeAccrual>false</IncludeAccrual>
  </IncludeAccruals>
  <Soft>
    <Soft>true</Soft>
  </Soft>
  <TriggerRatios>
    <TriggerRatio>0.8</TriggerRatio>
    <TriggerRatio>0.85</TriggerRatio>
    <TriggerRatio>0.9</TriggerRatio>
  </TriggerRatios>
</CallData>
```

Listing 98: Convertible bond call data example 3

```
<!-- American issuer call between 2016-08-03 and 2018-08-03
     at a clean price of 100 (hard calls) -->
<CallData>
  <Styles>
    <Style>American</Style>
  </Styles>
  <ScheduleData>
    <Dates>
      <Dates>
        <Date>2016-08-03</Date>
        <Date>2018-08-03</Date>
      </Dates>
    </Dates>
  </ScheduleData>
  <Prices>
    <Price>1.00</Price>
  </Prices>
  <PriceTypes>
    <PriceType>Clean</PriceType>
  </PriceTypes>
  <IncludeAccruals>
    <IncludeAccrual>true</IncludeAccrual>
  </IncludeAccruals>
  <Soft>
    <Soft>false</Soft>
  </Soft>
  <TriggerRatios/>
</CallData>
```

Listing 99: Convertible bond call data example 4

```
<!-- American issuer call between 2016-08-03 and 2020-08-03 (excl),
     hard calls at 100 between 2016-08-03 and 2018-08-03 (excl),
     soft calls at 102 between 2018-08-03 and 2019-08-03 (excl),
     soft calls at 103 between 2019-08-03 and 2020-08-03 -->
<CallData>
  <Styles>
    <Style>American</Style>
  </Styles>
  <ScheduleData>
    <Dates>
      <Dates>
        <Date>2016-08-03</Date>
        <Date>2018-08-03</Date>
        <Date>2019-08-03</Date>
        <Date>2020-08-03</Date>
      </Dates>
    </Dates>
  </ScheduleData>
  <Prices>
    <Price>1.00</Price>
    <Price startDate="2018-08-03">1.02</Price>
    <Price startDate="2019-08-03">1.03</Price>
  </Prices>
  <PriceTypes>
    <PriceType>Clean</PriceType>
  </PriceTypes>
  <IncludeAccruals>
    <IncludeAccrual>true</IncludeAccrual>
  </IncludeAccruals>
  <Soft>
    <Soft>false</Soft>
    <Soft startDate="2018-03-03">true</Soft>
  </Soft>
  <TriggerRatios>
    <TriggerRatio>0.8</TriggerRatio>
    <TriggerRatio startDate="2019-08-03">0.9</TriggerRatio>
  </TriggerRatios>
</CallData>
```

Listing 100: Convertible bond call data example 5

```
<!-- Bermudan (hard) calls at 100 at 3 dates from 2016 to 2018,
     followed by American (soft) calls at 102 between 2018 and 2020 -->
<CallData>
  <Styles>
    <Style>Bermudan</Style>
    <Style startDate="2018-08-03">American</Style>
  </Styles>
  <ScheduleData>
    <Dates>
      <Dates>
        <Date>2016-08-03</Date>
        <Date>2017-08-03</Date>
        <Date>2018-08-03</Date>
        <Date>2020-08-03</Date>
      </Dates>
    </Dates>
  </ScheduleData>
  <Prices>
    <Price>1.00</Price>
    <Price startDate="2018-08-03">1.02</Price>
  </Prices>
  <PriceTypes>
    <PriceType>Clean</PriceType>
  </PriceTypes>
  <IncludeAccruals>
    <IncludeAccrual>true</IncludeAccrual>
  </IncludeAccruals>
  <Soft>
    <Soft>false</Soft>
    <Soft startDate="2018-08-03">true</Soft>
  </Soft>
  <TriggerRatios>
    <TriggerRatio>0.8</TriggerRatio>
  </TriggerRatios>
</CallData>
```

Listing 101: Convertible bond put data example 6

```
<!-- Bermudan puts calls at 100, 101, 102 at 3 dates from 2016 to 2018 -->
<PutData>
  <Styles>
    <Style>Bermudan</Style>
  </Styles>
  <ScheduleData>
    <Dates>
      <Dates>
        <Date>2016-08-03</Date>
        <Date>2017-08-03</Date>
        <Date>2018-08-03</Date>
      </Dates>
    </Dates>
  </ScheduleData>
  <Prices>
    <Price>1.00</Price>
    <Price>1.01</Price>
    <Price>1.02</Price>
  </Prices>
  <PriceTypes>
    <PriceType>Clean</PriceType>
  </PriceTypes>
  <IncludeAccruals>
    <IncludeAccrual>true</IncludeAccrual>
  </IncludeAccruals>
</PutData>
```

Listing 102: Convertible bond make whole data (conversion ratio increase)

```
<CallData>
...
  <MakeWhole>
    <ConversionRatioIncrease>
      <Cap>0.0740740</Cap>
      <StockPrices>13.50,15.00,16.20,18.00</StockPrices>
      <CrIncreases>
        <CrIncrease startDate="2020-06-25">0.0123456,0.0107487,0.0097173,0.0084567</CrIncrease>
        <CrIncrease startDate="2021-07-01">0.0123456,0.0096880,0.0086963,0.0075294</CrIncrease>
        <CrIncrease startDate="2022-07-01">0.0123456,0.0083927,0.0074222,0.0063383</CrIncrease>
        <CrIncrease startDate="2023-07-01">0.0123456,0.0069360,0.0058790,0.0048322</CrIncrease>
        <CrIncrease startDate="2024-07-01">0.0123456,0.0054453,0.0040025,0.0028833</CrIncrease>
        <CrIncrease startDate="2025-07-01">0.0123456,0.0049380,0.0000000,0.0000000</CrIncrease>
      </CrIncreases>
    </ConversionRatioIncrease>
  </MakeWhole>
</CallData>
```

Specification of ConversionData:

As in the case of the CallData, all lists can be specified as either an explicit list of values corresponding to the schedule dates list or using the attribute `startDate`. The ConversionRatios element is an expcetion, the given start dates are interpreted independently of these schedule dates.

See listings 103, 104,105,106, 107,108 for examples of conversion schedules.

- Styles: The styles of the conversion rights. Notice that Bermudan is used to define European conversion rights as well, namely as a Bermudan conversion right with a single date. The attribute `startDate` can be used to specify the list. Can be omitted, if no conversion dates are given.
  Allwoable values: American, Bermudan
- ScheduleData: The dates defining when the bond is convertible. For Bermudan exercises, the conversion can be executed on the single dates given in the list. For American exercises, the conversion can be executed between a given start and end date. Can be omitted, if no conversion rights are present.
  Allowable values: see 19.3.4.
- ConversionRatios: A list of conversion ratios C^R. The attribute `startDate` can be used to specify a date from which the ratio is valid. Notice that this date is always interpreted "as is", i.e. it is not mapped onto the next date in the defined schedule. If no startDate is given for a ratio, this ratio is interpreted as the initial ratio.
  Allowable values: Any non-negative real number.
- FixedConversionAmounts: If this node is given, the conversion is specified to be conversion to fixed cash amounts instead of equity. If the cash amount currency is different from the bond currency, the FXIndex node must be given. See 108 for an example. As for ConversionRatios the attribute `startDate` can be used to specify a date from which the amount is valid and this date is interpreted "as is", i.e. not mapped onto the next date in the defined schedule. The nodes
  - ConversionRatios
  - ContingentConversion
  - MandatoryConversion
  - ConversionResets
  - Underlying
  - Exchangeable

  must *not* be given, if this node is present. Furthermore, the following nodes from other sections are not applicable if the conversion is specified to be fixed cash amounts, and must therefore not be given:
  - CallData/Soft
  - CallData/TriggerRatios
  - CallData/NoMTriggers
  - CallData/MakeWhole
  - DividendProtectionData (including all subnodes)
- ContingentConversion: This adds a condition $C_t^R S_t > B$ on the convertibility for the periods defined by the conversion dates. Optional.
  - Observations: A list of observation modes.

Allowable values: Spot (trigger is checked on the conversion date), StartOfPeriod (trigger is checked on the start of the conversion period defined by the dates list, for American style conversion only)

  - Barriers: A list of barriers B associated to the conversion dates.
    Allowable values: Positive real number or zero (conversion is not made contingent for this date).

- MandatoryConversion: This adds a mandatory conversion obligation at a date greater than all other conversion dates (if any). Optional.
  - Date: The mandatory conversion date.
    Allowable values: Any date not earlier than the last otherwise specified conversion date.
  - Type: The type of the mandatory conversion.
    Allowable values: PEPS
  - PepsData: Details of mandatory conversion type PEPS.
    * UpperBarrier: upper barrier for PEPS payoff.
      Allowable values: A real number.
    * LowerBarrier: lower barrier for PEPS payoff.
      Allowable values: A real number.
    * UpperConversionRatio: conversion ratio for upper barrier in PEPS payoff.
      Allowable values: A real number.
    * LowerConversionRatio: conversion ratio for lower barrier in PEPS payoff.
      Allowable values: A real number.
- ConversionResets: This adds a reset schedule for the conversion rate. If a reset feature is defined, only an initial ConversionRatio can be defined, the future conversion ratios are determined by the resets. The startDate attribute can be used to define references, thresholds, gearings, floors, global floors. Optional.
  - ScheduleData: The conversion reset dates.
    Allowable values: see 19.3.4.
  - References: Whether the initial conversion price C_0^P or the current conversion price C_t^P is the reference for the reset.
    Allowable values: InitialConversionPrice, CurrentConversionPrice
  - Thresholds: The threshold T that triggers a reset ($S_t < TC_0^P$ or $S_t < TC_t^P$, depending on Reference)
    Allowable values: positive number or zero (disables the reset on this date effectively)
  - Gearings: The gearings g for the conversion rate adjustment. Option, defaults to 0 (= no gearing applicable)
    Allowable values: positive number or zero (no gearing applicable on this date).
  - Floors: The floors f for the conversion rate adjustment. Optional, defaults to 0 (= no floor applicable)
    Allowable values: positive number or zero (no floor applicable on this date)

  - GlobalFloors: The global floors for the conversion rate adjustment. Option, defaults to 0 (= no global floor applicable)
    Allowable values: positive number or zero (no global floor applicable on this date)

- Underlying: The equity underlying.
  Allwoable values: See 19.3.29, the underlying type must be equity.
- FXIndex: If equity ccy is different from bond ccy, an fx index for the two involved ccy is required.
  Allowable values: The format of the FX Index is"FX-SOURCE-CCY1-CCY2" as described in table 28.
- Exchangeable: Node with data for exchangeables. Option, if omitted, the structure is considered non-exchangeable. Subnodes are:
  - IsExchangeable: indicates whether the convertible bond is exchangeable
    Allowable values: true, false
  - EquityCreditCurve: the credit curve modeling the equity issuer default, required if IsExchangeable is true.
    Allowable values: A valid credit curve identifier, e.g the ISIN of a reference bond with the ISIN: prefix: `ISIN:XXNNNNNNNNN`
  - Secured: Indicates whether the convertible is secured with pledged shares or not. Optional, defaults to false.
    Allowable values: true, false.

Listing 103: Convertible bond conversion example 1

```
<!-- Three conversion dates (Bermudan), conversion ratio is 0.5 -->
  <ConversionData>
    <Styles>
      <Style>Bermudan</Style>
    </Styles>
    <ScheduleData>
      <Dates>
        <Dates>
          <Date>2016-08-03</Date>
          <Date>2017-08-03</Date>
          <Date>2018-08-03</Date>
        </Dates>
      </Dates>
    </ScheduleData>
    <ConversionRatios>
      <ConversionRatio>0.05</ConversionRatio>
    </ConversionRatios>
    <Underlying>
      <Type>Equity</Type>
      <Name>RIC:.ABCD</Name>
    </Underlying>
    <FXIndex>FX-ECB-EUR-USD</FXIndex>
    <Exchangeable>
      <IsExchangeable>true</IsExchangeable>
      <EquityCreditCurve>ISIN:XS0982710740</EquityCreditCurve>
      <Secured>true</Secured>
    </Exchangeable>
  </ConversionData>
```

Listing 104: Convertible bond conversion example 2

```
<!-- American conversion between 2016-08-03 and 2020-08-03, with
     conversion ratio 0.5 for 2016-08-03 through 2018-08-03 (excl) and
     conversion ratio 0.6 for 2018-08-03 through 2020-08-03 -->
  <ConversionData>
    <Styles>
      <Style>American</Style>
    </Styles>
    <ScheduleData>
      <Dates>
        <Dates>
          <Date>2016-08-03</Date>
          <Date>2018-08-03</Date>
          <Date>2020-08-03</Date>
        </Dates>
      </Dates>
    </ScheduleData>
    <ConversionRatios>
      <ConversionRatio>0.05</ConversionRatio>
      <ConversionRatio startDate="2018-08-03">0.06</ConversionRatio>
    </ConversionRatios>
    <Underlying>
      <Type>Equity</Type>
      <Name>RIC:.ABCD</Name>
    </Underlying>
  </ConversionData>
```

Listing 105: Convertible bond conversion example 3

```
<!-- American conversion between 2016-08-03 and 2018-08-03, with conversion
     ratio 0.5, the conversion is contingent on the parity being above 1.3
     on 2016-08-03 for the conversion between 2016-08-03 and 2017-08-03 (excl)
     on 2017-08-03 for the conversion between 2017-08-03 and 2018-08-03 -->
  <ConversionData>
    <Styles>
      <Style>American</Style>
    </Styles>
    <ScheduleData>
      <Dates>
        <Dates>
          <Date>2016-08-03</Date>
          <Date>2017-08-03</Date>
          <Date>2018-08-03</Date>
        </Dates>
      </Dates>
    </ScheduleData>
    <ConversionRatios>
      <ConversionRatio>0.05</ConversionRatio>
    </ConversionRatios>
    <ContingentConversion>
      <Observations>
        <Observation>StartOfPeriod</Observation>
      </Observations>
      <Barriers>
        <Barrier>1.3</Barrier>
      </Barriers>
    </ContingentConversion>
    <Underlying>
      <Type>Equity</Type>
      <Name>RIC:.ABCD</Name>
    </Underlying>
  </ConversionData>
```

Listing 106: Convertible bond conversion example 4

```
<!-- American converion between 2016-08-03 and 2018-08-03 with CR 0.5.
     Mandatory conversion on 2020-08-03:
     LowerConversionRatio applies if stock price < LowerBarrier,
     UpperConversionRatio applies if stock price > UpperBarrier -->
  <ConversionData>
    <Styles>
      <Style>American</Style>
    </Styles>
    <ScheduleData>
      <Dates>
        <Dates>
          <Date>2016-08-03</Date>
          <Date>2018-08-03</Date>
        </Dates>
      </Dates>
    </ScheduleData>
    <ConversionRatios>
      <ConversionRatio>0.05</ConversionRatio>
    </ConversionRatios>
    <MandatoryConversion>
      <Date>2020-08-03</Date>
      <Type>PEPS</Type>
      <PepsData>
        <UpperBarrier>32.5</UpperBarrier>
        <LowerBarrier>20.5</LowerBarrier>
        <UpperConversionRatio>0.08</UpperConversionRatio>
        <LowerConversionRatio>0.03</LowerConversionRatio>
      </PepsData>
    </MandatoryConversion>
    <Underlying>
      <Type>Equity</Type>
      <Name>RIC:.ABCD</Name>
    </Underlying>
  </ConversionData>
```

Listing 107: Convertible bond conversion example 5

```
<!-- American conversion between 2016-08-03 and 2018-08-03 with CR 0.5.
     The conversion ratio is reset on 2016-11-03, 2017-02-03, 2018-05-03
     using T = 0.9, g = 0.8, f = 0.6, F = 0.6. -->
  <ConversionData>
    <Styles>
      <Style>American</Style>
    </Styles>
    <ScheduleData>
      <Dates>
        <Dates>
          <Date>2016-08-03</Date>
          <Date>2018-08-03</Date>
        </Dates>
      </Dates>
    </ScheduleData>
    <ConversionRatios>
      <ConversionRatio>0.05</ConversionRatio>
    </ConversionRatios>
    <ConversionResets>
      <ScheduleData>
        <Dates>
          <Dates>
            <Date>2016-11-03</Date>
            <Date>2017-02-03</Date>
            <Date>2018-05-03</Date>
          </Dates>
        </Dates>
      </ScheduleData>
      <References>
        <Reference>InitialConversionPrice</Reference>
      </References>
      <Thresholds>
        <Threshold>0.9</Threshold>
      </Thresholds>
      <Gearings>
        <Gearing>0.8</Gearing>
      </Gearings>
      <Floors>
        <Floor>0.7</Floor>
      </Floors>
      <GlobalFloors>
        <GlobalFloor>15</GlobalFloor>
      </GlobalFloors>
    </ConversionResets>
    <Underlying>
      <Type>Equity</Type>
      <Name>RIC:.ABCD</Name>
    </Underlying>
  </ConversionData>
```

Listing 108: Convertible bond conversion example 6

```
<!-- American conversion between 2024-08-24 and 2027-05-13, with
     conversion to 0.87 GBP cash for 2024-08-24 through 2024-11-23 (excl) and
     conversion to 0.75 GBP cash for 2024-11-23 through 2027-05-13 -->
  <ConversionData>
    <Styles>
      <Style>American</Style>
    </Styles>
    <ScheduleData>
      <Dates>
        <Dates>
          <Date>2024-08-24</Date>
          <Date>2024-11-23</Date>
          <Date>2027-05-13</Date>
        </Dates>
      </Dates>
    </ScheduleData>
    <FixedAmountConversion>
      <Currency>GBP</Currency>
      <Amounts>
        <Amount>0.87</Amount>
        <Amount startDate="2024-11-24">0.75</Amount>
      </Amounts>
    </FixedAmountConversion>
  </ConversionData>
```

Specification of DividendProtectionData:

As for the CallData, all lists can be specified as either an explicit list of values corresponding to the schedule dates list or using the attribute `startDate`.

See listings 109, 110 for examples of dividend protection schedules.

- ScheduleData: The dates of the dividend protection schedule. The first date marks the date when the dividend protection becomes effective, i.e. dividend payments from this date on are taken into account in conversion ratio adjustments or passthroughs. The second date is then the first date on which the accumulated dividends between the first and second date trigger a conversion ratio reset or passthrough, and similar for all subsequent dates. The last given date is the last date with a conversion ratio reset or passthrough.
  Allowable values: see 19.3.4.
- AdjustmentStyles: Whether the dividend excessing the threshold is passed through or the conversion ratio is adjusted. In both cases, the adjustment can be upwards only or up and down.
  Allwoable values: CrUpOnly, CrUpDown, CrUpOnly2, CrUpDown2, PassThroughUpOnly, PassThroughUpDown
- DividendTypes: Whether the conversion ratio adjustment is calculated in terms of absolute or relative dividends. Does not have an effect for pass through dividends (should be set to Aboslute in this case).
  Allwoable values: Absolute, Relative
- Thresholds: The threshold H. Notice that the threshold applies to each single period of

the dividend protection schedule. If the threshold is e.g. provided on an annual basis in the terms of the convertible bond, but the dividend protection schedule is quarterly, then the threshold in the trade xml should be the annual threshold divided by 4. Allwoable values: Any non-negativee number.

Listing 109: Convertible bond dividend protection example 1

```
<!-- Divdend protection based on aboslute dividend amounts via adjustment
     of the conversion rate, up-only adjustment. -->
<DividendProtectionData>
  <ScheduleData>
    <Dates>
      <Dates>
        <Date>2016-08-03</Date>
        <Date>2017-08-03</Date>
        <Date>2018-08-03</Date>
        <Date>2019-08-03</Date>
      </Dates>
    </Dates>
  </ScheduleData>
  <AdjustmentStyles>
    <AdjustmentStyle>CrUpOnly</AdjustmentStyle>
  </AdjustmentStyles>
  <DividendTypes>
    <DividendType>Absolute</DividendType>
  </DividendTypes>
  <Thresholds>
    <Threshold>1.2</Threshold>
  </Thresholds>
</DividendProtectionData>
```

Listing 110: Convertible bond dividend protection example 2

```
<!-- Dividend protection based on relative dividend amounts via adjustment
     of the conversion rate, up-only adjustment. -->
<DividendProtectionData>
  <ScheduleData>
    <Dates>
      <Dates>
        <Date>2016-08-03</Date>
        <Date>2017-08-03</Date>
        <Date>2018-08-03</Date>
        <Date>2019-08-03</Date>
      </Dates>
    </Dates>
  </ScheduleData>
  <AdjustmentStyles>
    <AdjustmentStyle>CrUpOnly</AdjustmentStyle>
  </AdjustmentStyles>
  <DividendTypes>
    <DividendType>Relative</DividendType>
  </DividendTypes>
  <Thresholds>
    <Threshold>0.01</Threshold>
  </Thresholds>
</DividendProtectionData>
```

19.2.49 Ascot

Payoff

An Ascot or a Convertible Bond Option is an American style option to buy back a convertible bond. The buyer of a Call Ascot can exercise the deal and get the underlying bond in exchange for paying the strike.

The payout formula for a Call Ascot is:

$$Payout = \max(0, convertiblePrice - Strike)$$

And for a Put Ascot:

$$Payout = \max(0, Strike - convertiblePrice)$$

where:

$$Strike = bondQuantity \cdot (upfrontPayment + assetLeg - redemptionLeg) - fundingLeg$$

Input

An Ascot is set up using an `AscotData` block as shown in listing 111. The bond details are read from reference data in this case.

Listing 111: Ascot set up using reference data

```
<Trade id="Ascot">
  <TradeType>Ascot</TradeType>
  <Envelope>...</Envelope>
  <AscotData>
    <ConvertibleBondData>
      <BondData>
        <SecurityId>ISIN:XY1000000000</SecurityId>
        <BondNotional>1000000.00</BondNotional>
      </BondData>
    </ConvertibleBondData>
    <OptionData>
     <LongShort>Long</LongShort>
     <OptionType>Call</OptionType>
     <Style>American</Style>
     <Settlement>Physical</Settlement>
     <ExerciseDates>
       <ExerciseDate>2029-02-03</ExerciseDate>
     </ExerciseDates>
    </OptionData>
    <ReferenceSwapData>
      <LegData>
        <LegType>Floating</LegType>
        <Payer>false</Payer>
        ...
      </LegData>
    </ReferenceSwapData>
  <AscotData>
</Trade>
```

The meanings and allowable values of the elements in the block are as follows:

- ConvertibleBondData: This describes the underlying convertible bond, see 19.2.48.
- OptionData: This is a trade component sub-node outlined in section 19.3.1 Option Data. The relevant fields in the `OptionData` node for an Ascot are:
  - `LongShort` The allowable values are *Long* or *Short*. The LongShort flag multiplies the option price with +1 / -1. Call and Put payout formulas above are from the long perspective
  - `OptionType` The allowable values are *Call* or *Put*. See payout formulas above.
  - `Style` The Ascot type allows for *American* option exercise style only.
  - `Settlement` The allowable values are *Cash* or *Physical*.
  - An `ExerciseDates` node where exactly one `ExerciseDate` date element must be given.
  - `Premiums` [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2
- ReferenceSwapData: Contains a single `LegData` node that describes the trade's reference swap funding leg. The asset leg is implied from the bond data. Payer should always be *false* i.e. the swap is entered from the viewpoint of the asset swap buyer.

19.2.50 Collateral Bond Obligation CBO

A Cashflow CDO or Collateral Bond Obligation CBO (trade type *CBO*) can be set up in a short version referencing the underlying CBO structure in a static CBO reference datum or a long version, where the CBO structure is specified explicitly.

The main building block is the `CBOData` block as shown in listing 112. The `CBOData` requires the two components `CBOInvestment` and `CBOStructure`. Where the latter represents the general structure, the former specfies the actual investment. For the short version, the CBO is fully specified using the component `CBOInvestment` only, the component `CBOStructure` can be omitted.

Listing 112 exhibits the long version:

Listing 112: CBO Data

```
<CBOData>
  <CBOInvestment>
    <TrancheName>JuniorNote</TrancheName>
    <Notional>4000000.00</Notional>
    <StructureId>Constellation</StructureId>
  </CBOInvestment>
  <CBOStructure>
    <DayCounter>ACT/ACT</DayCounter>
    <PaymentConvention>F</PaymentConvention>
    <Currency>EUR</Currency>
    <ReinvestmentEndDate>2019-12-31</ReinvestmentEndDate>
    <SeniorFee>0.01</SeniorFee>
    <FeeDayCounter>A365</FeeDayCounter>
    <SubordinatedFee>0.02</SubordinatedFee>
    <EquityKicker>0.25</EquityKicker>
    <BondBasketData>
      ...
    </BondBasketData>
    <CBOTranches>
      ...
    </CBOTranches>
    <ScheduleData>
      ...
    </ScheduleData>
  </CBOStructure>
</CBOData>
```

The meanings of the elements of the `CBOData` node follow below:

- TrancheName: Specifies of which tranche, results are shown in the report files (NPV, Sensitivity, ...). The name needs to match one the names specified in `CBOTranches`.
- Notional: Is the invested amount into the tranche specified above. The value is used to scale the NPV from the general tranche NPV, so it may be different to the face amount specified in `CBOTranches`.
- StructureId: if details of the cbo are read from the reference data, StructureId is used as a key.
- DayCounter: The day count convention of the tranches. Allowable values: See table 25.
- PaymentConvention: The payment convention of the tranches. Allowable values: See Table 21 Roll Convention.
- Currency: Defines the currency of the trade, i.e. the currency of the tranches. Allowable values: See Table 22 `Currency`.
- ReinvestmentEndDate: Defines the end of the reinvestment period. During the reinvestment period, principal proceeds are used to reinvest in eliglible assets rather than to redeem CBO notes. Currently the model cannot handle underlying bonds with full amortisation within the reinvestment period. In case the underlying bonds amortise only parts of their full notional (during that period), the model will leave outstanding balance constant until the end of the reinvestment period. Therafter the underyling bonds amortises at a higher speed.

- SeniorFee: The fee, expressed as rate, paid before all other obligations, top of the waterfall.
- FeeDayCounter: The day count convention for the fees. Allowable values: See table 25.
- SubordinatedFee: The fee, expressed as rate, paid after all other obligations.
- EquityKicker: Fraction x of the residual payment, that will be split among the senior fee receiver (x) and the equity piece (1-x).
- BondBasketData: All specifications of the underlying bond basket. Uses the sub node BondBasketData as described in section 19.3.33.
- CBOTranches: All required instrument data for the tranches of the CBO. Uses the sub node CBOTranches as described in section 19.3.34.
- ScheduleData: This is a trade component sub-node outlined in section 19.3.4 Schedule Data and Dates.

Listing 113 exhibits the reference data in conjunction with short version of the `CBOData` in listing 114. The element meanings are the same as in the long version.

Listing 113: CboReferenceData

```
<ReferenceDatum id="Constellation">
    <Type>CBO</Type>
    <CboReferenceData>
        <Currency>USD</Currency>
        <DayCounter>A365</DayCounter>
        <PaymentConvention>F</PaymentConvention>
        <SeniorFee>0.001</SeniorFee>
        <FeeDayCounter>A365</FeeDayCounter>
        <SubordinatedFee>0.005</SubordinatedFee>
        <EquityKicker>0.01</EquityKicker>
        <CBOTranches>
            ...
        </CBOTranches>
        <ScheduleData>
            ...
        </ScheduleData>
        <BondBasketData>
            ...
        </BondBasketData>
    </CboReferenceData>
</ReferenceDatum>
```

Listing 114: CBOInvestment

```
<CBOData>
  <CBOInvestment>
    <TrancheName>JuniorNote</TrancheName>
    <Notional>4000000.00</Notional>
    <StructureId>Constellation</StructureId>
  </CBOInvestment>
</CBOData>
```

19.2.51 Composite Trade

A composite trade is a hybrid position consisting of multiple component trades. As such it inherits the characteristics of the trades defined within it. Examples of Composite Trades include combinations of vanilla options like straddles.

The `CompositeTradeData` node is the trade data container for the *CompositeTrade* trade type. A composite trade is a hybrid position consisting of multiple component trades. The structure of an example `CompositeTradeData` node for a commodity option is shown in Listing 115.

Listing 115: Composite trade data

```
<CompositeTradeData>
  <Currency>USD</Currency>
  <NotionalCalculation>Sum</NotionalCalculation>
  <Components>
    <Trade id="">
      <!-- A valid trade xml -->
    </Trade>
    <Trade id="">
      <!-- A valid trade xml -->
    </Trade>
  </Components>
</CompositeTradeData>
```

Listing 116: Composite trade data with Portfolio Reference Data

```
<CompositeTradeData>
  <Currency>USD</Currency>
  <NotionalCalculation>Sum</NotionalCalculation>
  <PortfolioBasket>true</PortfolioBasket>
  <BasketName>NAME</BasketName>
<IndexQuantity>100</IndexQuantity>
</CompositeTradeData>
```

The meanings and allowable values of the elements in the `CompositeTradeData` node follow below.

- Currency: Defines the currency the NPV of the composite trade will be represented in.
  Allowable values: See Table 22 `Currency`.
- NotionalCalculation [Optional]: The method by which the notional of the composite trade will be calculated.
  Allowable values:

  Sum: The notional will be calculated as the sum of the notionals of the constituent trades. This is the default behaviour if the field is omitted (unless an override is provided).

  Mean or *Average*: The notional will be calculated as the mean of the notionals of the constituent trades.

  First: The notional of the first constituent trade will be used.

Last: The notional of the first constituent trade will be used.

Min: The notional will be calculated as the minimum of the notionals of the constituent trades.

Max: The notional will be calculated as the minimum of the notionals of the constituent trades.

Override: the notional will be read directly from the notional override field.

- NotionalOverride [Optional]: The notional which will be used for the trade, overriding any calculation method specified.
  Allowable values: Any non-negative real number.
- Components: The portfolio of trades that make up the composite trade.
  Allowable values: These trades should be valid xmls that could otherwise be entered into the portfolio, with the exception that they can have empty ids.
- PortfolioBasket [Optional]: Indicate if the Component represent a portfolio basket.
  Allowable values: Boolean true or false.
- BasketName [Optional]: The portfolio Id.
  Allowable values: Any string. Note that if PortfolioBasket is True then there must be a BasketName. We look up the Basket within the reference data.
- IndexQuantity [Optional]: Number of shares of the index.

19.2.52 Credit Default Swap / Quanto Credit Default Swap

A credit default swap, trade type `CreditDefaultSwap`, is set up using a `CreditDefaultSwapData` block as shown in listing 117 or 118. The `CreditDefaultSwapData` block must include either a `CreditCurveId` element or a `ReferenceInformation` node.

The `LegData` sub-node must be a fixed leg, and represents the recurring premium payments. The direction of the fixed leg payments define if the CDS is for bought (`Payer`: *true*) or sold (`Payer`: *false*) protection.

The elements have the following meaning:

- IssuerId [Optional]: An identifier for the reference entity of the CDS. For informational purposes and not used for pricing.
- CreditCurveId: The identifier of the reference entity defining the default curve used for pricing. For the allowable values, see `CreditCurveId` for credit trades - single name in Table 30. A `ReferenceInformation` node may be used in place of this `CreditCurveId` node.
- ReferenceInformation: This node may be used as an alternative to the `CreditCurveId` node to specify the reference entity, tier, currency and documentation clause for the CDS. This in turn defines the credit curve used for pricing. The `ReferenceInformation` node is described in further detail in Section 19.3.27.
- SettlesAccrual [Optional]: Whether or not the accrued coupon is due in the event of a default. This defaults to `true` if not provided.

Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36.

- ProtectionPaymentTime [Optional]: Controls the payment time of protection and premium accrual payments in case of a default event. Defaults to `atDefault`.

  Allowable values: `atDefault`, `atPeriodEnd`, `atMaturity`. Overrides the `PaysAtDefaultTime` node

- PaysAtDefaultTime [Deprecated]: *true* is equivalent to ProtectionPaymentTime = atDefault, *false* to ProtectionPaymentTime = atPeriodEnd. Overridden by the `ProtectionPaymentTime` node if set

  Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36.

- ProtectionStart [Optional]: The first date where a credit event will trigger the contract. This defaults to the first date in the schedule if it is not provided. Must be set to a date before or on the first date in the schedule if the `LegData` has a rule that is not one of `CDS` or `CDS2015`. In general, for standard CDS traded after the CDS Big Bang in 2009, the protection start date is equal to the trade date. Therefore, typically the `ProtectionStart` should be set to the trade date of the CDS.

- UpfrontDate [Optional]: Settlement date for the UpfrontFee if an UpfrontFee is provided. If an UpfrontFee is provided and it is non-zero, `UpfrontDate` is required. The `UpfrontDate`, if provided, must be on or after the ProtectionStart date. Typically, it is 3 business days after the CDS contract trade date.

- UpfrontFee [Optional]: The upfront payment, expressed as a percentage in decimal form, to be multiplied by notional amount. If an UpfrontDate is provided, an UpfrontFee must also be provided. The UpfrontFee can be omitted but cannot be left blank. The UpfrontFee can be negative. The UpfrontFee is treated as an amount payable by the protection buyer to the protection seller. A negative value for the UpfrontFee indicates that the UpfrontFee is being paid by the protection seller to the protection buyer.

  Allowable values: Any real number, expressed in decimal form as a percentage of the notional. E.g. an UpfrontFee of *0.045* and a notional of 10M, would imply an upfront fee amount of 450K.

- FixedRecoveryRate [Optional]: This node holds the fixed recovery rate if the CDS is a fixed recovery CDS. For a standard CDS, this field should be omitted.

- TradeDate [Optional]: The CDS trade date. If omitted, the trade date is deduced from the protection start date. If the schedule provided in the `LegData` has a rule that is either `CDS` or `CDS2015`, the trade date is set equal to the protection start date. This is the standard for CDS traded after the CDS Big Bang in 2009. Otherwise, the trade date is set equal to the protection start date minus 1 day as it was standard before the CDS Big Bang to have protection starting on the day after the trade date.

- CashSettlementDays [Optional]: The number of business days between the trade date and the cash settlement date. For standard CDS, this is 3 business days. If omitted, this defaults to 3.

- RebatesAccrual [Optional]: The protection seller pays the accrued scheduled current coupon at the start of the contract. The rebate date is not provided but computed to be two days after protection start. This defaults to `true` if not provided.

  Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36.

The `LegData` block then defines the CDS premium leg structure. This premium leg must be be of type `Fixed` as described in Section 19.3.5.

Listing 117: CreditDefaultSwap Data

```
<CreditDefaultSwapData>
  <IssuerId>CPTY_A</IssuerId>
  <CreditCurveId>RED:008CA0|SNRFOR|USD|MR14</CreditCurveId>
  <SettlesAccrual>Y</SettlesAccrual>
  <ProtectionPaymentTime>atDefault</ProtectionPaymentTime>
  <ProtectionStart>20160206</ProtectionStart>
  <UpfrontDate>20160208</UpfrontDate>
  <UpfrontFee>0.0</UpfrontFee>
  <LegData>
        <LegType>Fixed</LegType>
        <Payer>false</Payer>
        ...
  </LegData>
</CreditDefaultSwapData>
```

Listing 118: `CreditDefaultSwapData` with `ReferenceInformation`

```
<CreditDefaultSwapData>
  <ReferenceInformation>
    <ReferenceEntityId>RED:008CA0</ReferenceEntityId>
    <Tier>SNRFOR</Tier>
    <Currency>USD</Currency>
    <DocClause>MR14</DocClause>
  </ReferenceInformation>
  <LegData>
    ...
  </LegData>
</CreditDefaultSwapData>
```

A quanto credit default swap is a credit default swap with different denomination and settlement currencies. Listing 119 shows an Example: The trade has a notional of 50 million BRL and pays a 6% premium. The premuim amounts are converted using the FX-TR20H-USD-BRL fixing two days before they are settled in USD. The hypothetical protection amounts computed for pricing purposes are converted to USD in a similar fashion.

Listing 119: Quanto CDS CreditDefaultSwap Data

```
<LegData>
  <LegType>Fixed</LegType>
  <Payer>true</Payer>
  <!-- This is the settlement currency -->
  <Currency>USD</Currency>
  <!-- This is the BRL notional -->
  <Notionals>
    <Notional>50000000</Notional>
  </Notionals>
  <!-- The FX index used to convert BRL amounts to the settlement ccy USD -->
  <Indexings>
    <Indexing>
      <Index>FX-TR20H-USD-BRL</Index>
      <FixingDays>2</FixingDays>
      <FixingCalendar>USD,BRL</FixingCalendar>
      <IsInArrears>true</IsInArrears>
    </Indexing>
  </Indexings>
  ...
  <FixedLegData>
    <Rates>
      <Rate>0.06</Rate>
    </Rates>
  </FixedLegData>
 ...
</LegData>
```

19.2.53 Index Credit Default Swap

An index credit default swap (trade type *IndexCreditDefaultSwap*) is set up using an `IndexCreditDefaultSwapData` block as shown in listing 120 and includes `LegData` and `BasketData` trade component sub-nodes.

The `LegData` sub-node must be a fixed leg, and represents the recurring premium payments. The direction of the fixed leg payments define if the Index CDS is for bought (`Payer`: *true*) or sold (`Payer`: *false*) protection. The notional on the fixed leg is the "unfactored notional", i.e. the notional excluding any defaults. This is opposed to the "trade date notional" which is reduced by defaults since the series inception until the trade date and the "current notional" or "factored notional" which is reduced by defaults between the series inception and the current evaluation date of the trade.

The `BasketData` sub-node (see section 19.3.28) is optional and specifies the constituent reference entities of the index. This sub-node is intended for non-standard indices, that require a bespoke basket. When `BasketData` is omitted, the index constituents are derived from the `CreditCurveId` element in the `IndexCreditDefaultSwapData` block.

Listing 120: Index CreditDefaultSwap Data

```
<IndexCreditDefaultSwapData>
  <CreditCurveId>RED:2I65BRHH6</CreditCurveId>
  <SettlesAccrual>Y</SettlesAccrual>
  <ProtectionPaymentTime>atDefault</ProtectionPaymentTime>
  <ProtectionStart>20160206</ProtectionStart>
  <UpfrontDate>20160208</UpfrontDate>
  <UpfrontFee>0.0</UpfrontFee>
  <LegData>
        <LegType>Fixed</LegType>
        <Payer>false</Payer>
        ...
  </LegData>
   <BasketData>
    <Name>
      <IssuerId>CPTY_1</IssuerId>
      <CreditCurveId>RED:</CreditCurveId>
      <Notional>100000.0</Notional>
      <Currency>USD</Currency>
    </Name>
    <Name>
      <IssuerId>CPTY_2</IssuerId>
      <CreditCurveId>RED:</CreditCurveId>
      <Notional>100000.0</Notional>
      <Currency>USD</Currency>
    </Name>
    <Name>
      <IssuerId>CPTY_3</IssuerId>
      <CreditCurveId>RED:</CreditCurveId>
      <Notional>100000.0</Notional>
      <Currency>USD</Currency>
    </Name>
    <!-- ... -->
  </BasketData>
</IndexCreditDefaultSwapData>
```

The meanings of the elements of the `IndexCreditDefaultSwapData` node follow below:

- CreditCurveId: The identifier of the index defining the default curve used for pricing. The pricing can be set up to either use the index curve id, or use the curve id:s of the individual index components defined in `BasketData`.

  Allowable values: See `CreditCurveId` for credit trades - index in Table 30. Note that the `CreditCurveId` cannot be a redcode or other identifier for an ABX or CMBX. For these underlyings, trade type *AssetBackedCreditDefaultSwap* is used instead.

- SettlesAccrual [Optional]: Whether or not the accrued coupon is due in the event of a default. This defaults to `true` if not provided.

  Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36.

- ProtectionPaymentTime [Optional]: Controls the payment time of protection and premium accrual payments in case of a default event. Defaults to `atDefault`.

Allowable values: `atDefault`, `atPeriodEnd`, `atMaturity`. Overrides the `PaysAtDefaultTime` node

- PaysAtDefaultTime [Deprecated]: `true` is equivalent to ProtectionPaymentTime = atDefault, `false` to ProtectionPaymentTime = atPeriodEnd. Overridden by the `ProtectionPaymentTime` node if set

  Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36.

- ProtectionStart [Optional]: The first date where a credit event will trigger the contract. This defaults to the first date in the schedule if it is not provided. Must be set to a date before or on the first date in the schedule if the `LegData` has a rule that is not one of `CDS` or `CDS2015`. In general, for standard index CDS, the protection start date is equal to the trade date. Therefore, typically the `ProtectionStart` should be set to the trade date of the index CDS.

  Allowable values: See `Date` in Table 20.

- UpfrontDate [Optional]: Settlement date for the UpfrontFee if an UpfrontFee is provided. If an UpfrontFee is provided and it is non-zero, UpfrontDate is required.

  Allowable values: See `Date` in Table 20. The UpfrontDate, if provided, must be on or after the ProtectionStart date.

- UpfrontFee [Optional]: The upfront payment, expressed in decimal form as a percentage of the notional. If an UpfrontDate is provided, an UpfrontFee must also be provided. The UpfrontFee can be omitted but cannot be left blank. The UpfrontFee can be negative. The UpfrontFee is treated as an amount payable by the protection buyer to the protection seller. A negative value for the UpfrontFee indicates that the UpfrontFee is being paid by the protection seller to the protection buyer.

  Allowable values: Any real number, expressed in decimal form as a percentage of the notional. E.g. an UpfrontFee of *0.045* and a notional of 10M, would imply an upfront fee amount of 450K.

- TradeDate [Optional]: The index CDS trade date. If omitted, the trade date is deduced from the protection start date. If the schedule provided in the `LegData` has a rule that is either `CDS` or `CDS2015`, the trade date is set equal to the protection start date. Otherwise, the trade date is set equal to the protection start date minus 1 day.

  Allowable values: See `Date` in Table 20.

- CashSettlementDays [Optional]: The number of business days between the trade date and the cash settlement date. For standard index CDS, this is generally 3 business days. If omitted, this defaults to 3.

  Allowable values: Any non-negative integer.

- RebatesAccrual [Optional]: The protection seller pays the accrued scheduled current coupon at the start of the contract. The rebate date is not provided but computed to be two days after protection start. This defaults to `true` if not provided.

  Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36.

The LegData block then defines the Index CDS premium leg structure. This premium leg must be be of type Fixed as described in Section 19.3.5.

19.2.54 Index Credit Default Swap Option

An index CDS option, trade type IndexCreditDefaultSwapOption, is an option to enter into an index CDS at a specified strike spread or strike price. The Index CDS Option is set up using an IndexCreditDefaultSwapOptionData node as shown in Listing 121. Its child nodes have the following meanings:

- IndexTerm [Optional]: An optional node giving the term of the underlying index CDS e.g. 3Y, 5Y, 7Y, 10Y etc. The main function of this node is to allow for different index CDS option volatility structures for different terms of the same index series e.g. a CDX HY Series 34 5Y volatility structure and a CDX HY Series 34 10Y volatility structure. If this node is omitted, the market is searched for a CDS volatility surface with ID equal to the value of the CreditCurveId node under IndexCreditDefaultSwapData. There will generally be one CreditCurveId for each index CDS series e.g. CDXHYS34V1 for CDX HY Series 34 Version 1. Consequently, there can only be one CDS volatility surface for this index CDS series. When IndexTerm is populated with the underlying index term, the market is searched for a CDS volatility surface with ID equal to the value of the CreditCurveId node with suffix -[IndexTerm]. For example, if the CreditCurveId node on an index CDS option trade is CDXHYS34V1 and the IndexTerm node is populated with 5Y, the market will be searched for a CDS volatility surface with ID CDXHYS34V1-5Y and this will be used in the trade valuation. In this way, different volatility surfaces can be used to value different terms of the same CDS index series.

  Allowable values: A string that can be parsed as a term that is a valid term for the underlying CDS index e.g. *5Y*, *10Y*, etc. Defaults to *5Y* if omitted.

- OptionData: A node defining the option details as described in Section 19.3.1. The relevant fields in the OptionData node for an IndexCDSOption are:
  - LongShort The allowable values are *Long* or *Short*. *Long* meaning that the holder has the option to enter into the underlying index CDS.
  - OptionType [Optional] *Put/Call* is optional and not used. The Payer field in the underlying Index CDS leg determines if the option is to buy or sell protection. The Payer field is from the perspective of the party that is long.
  - Style Must be set to *European* as this is the only supported exercise for IndexCreditDefaultSwapOption.
  - Settlement The allowable values are *Cash* or *Physical*.
  - PayOffAtExpiry Must be set to *false* as only payoff at exercise is supported.
  - An ExerciseDates node where exactly one ExerciseDate date element must be given.
  - Premiums [Optional]: Option premium amounts paid by the option buyer (*Long*) to the option seller (*Short*). See section 19.3.2

- `IndexCreditDefaultSwapData`: A node defining the underlying index CDS as described in Section 19.2.53. Note that the `StartDate` in the `Scheduledata` in the premium leg in the `IndexCreditDefaultSwapData` should be the date on which the underlying CDS is entered into if the option is exercised (as opposed to the inception date of the underlying index CDS series). Under standard terms, the `StartDate` would be equal to the `ExerciseDate` but it can also be on a date after the `ExerciseDate`, but not on a date before the `ExerciseDate`, unless Rule is *CDS2015* or *CDS* and `StartDate` is set at the start of the full IMM period that the `ExerciseDate` falls into.

  The `TradeDate` and `ProtectionStart` on the underlying CDS do not need to be populated. If omitted, which is recommended, the `TradeDate` and `ProtectionStart` on the underlying CDS default as follows:

  `TradeDate` = max (option `ExerciseDate`, underlying schedule `StartDate`)
  `ProtectionStart` = max (option `ExerciseDate`, underl. schedule `StartDate`)

  Note that the cash settlement date for the underlying swap upfront premium is set to the underlying `TradeDate` with defaults as above, plus 3 business days.

  Also note that for schedules with IMM rules (e.g. *CDS2015*), if the underlying schedule `StartDate` is not falling on an IMM date, it is adjusted to the previous quarterly IMM date.

  Finally, the notional is - as in the case of an Index Credit Default Swap - the "unfactored notional", i.e. the notional excluding any defaults between the series inception and the trade or evaluation date of the trade.

- `Strike` [Optional]: A real number defining the option strike level. If this is an empty string or omitted the strike will be determined according to table 88.

  Note that if a strike is given, the UpfrontFee on the underlying IndexCDS must be zero or omitted. The UpfrontFee is interpreted as a price strike.

  Allowable values: Any real number. Note that the `Strike` is expressed in decimal form when `StrikeType` is *Spread*, and in decimal form as percentage of notional when `StrikeType` is *Price*. I.e. a `Strike` of 1.05 is 105% of the notional when `StrikeType` is *Price*.

- `StrikeType` [Optional]: Determines the strike type. If *Spread* is given, the `Strike` is interpreted as a strike *spread*. If *Price* is given, the `Strike` is interpreted as a strike *price*. If omitted or left blank, it will be determined according to table 88.

  Allowable values: *Spread* or *Price*. Note that *Spread* is only supported when the underlying market data is set up with spread strikes, and *Price* is only supported when the market data is set up with price strikes. Typically the market data convention for Index CDS Options is spread strikes, with the exception of CDX North America High Yield (CDX NA HY) names, where the convention is to use price strikes.

- `TradeDate` [Optional]: The trade date. If not given defaults to the valuation date. In case of an underlying default the trade date is used to determine whether the underlying notional before default should be considered part of the outstanding notional (TradeDate < AuctionDate) or not (TradeDate ≥ AuctionDate).

  Allowable values: See `Date` in Table 20. Can not be later than the valuation date.

- `FrontEndProtectionStartDate` [Optional]: The date on which the front end protection kicks in. If not given, it defaults to the TradeDate. In case of an underlying default this date is used to determine whether the underlying contributes to the realised front end protection amount (FrontEndProtectionStartDate < AuctionDate) or not (FrontEndProtectionStartDate ≥ AcutionDate).

  Allowable values: See `Date` in Table 20. Can not be later than the trade date.

- `FixedRecoveryRate` [Optional]: If provided, this recovery rate will be used in place of the market quoted recovery rate of the underlying.

  Allowable values: Any real number in the range [0,1]. If omitted, the market quoted recovery rate of the underlying is used.

Listing 121: Example Structure of `IndexCreditDefaultSwapOptionData` node.

```
<IndexCreditDefaultSwapOptionData>
  <IndexTerm>5Y</IndexTerm>
  <OptionData>
      <LongShort>Long</LongShort>
      <Style>European</Style>
      <Settlement>Cash</Settlement>
      <PayOffAtExpiry>false</PayOffAtExpiry>
      <ExerciseDates>
        <ExerciseDate>2023-05-09</ExerciseDate>
      </ExerciseDates>
  </OptionData>
  <IndexCreditDefaultSwapData>
    ...
  </IndexCreditDefaultSwapData>
  <Strike>1.063</Strike>
  <StrikeType>Price</StrikeType>
</IndexCreditDefaultSwapOptionData>
```

Strike	StrikeType	UpfrontFee	Effective Strike	Effective StrikeType
na	na	na	RunningCoupon	Spread
na	Spread	na	RunningCoupon	Spread
na	Price	na	1.0	Price
K	na	na	K	Spread
K	Spread	na	K	Spread
K	Price	na	K	Price
na	na	U	1.0 - U	Price
na	Spread	U (= 0)	RunningCoupon	Spread
na	Spread	U (≠ 0)	(not allowed)	(not allowed)
na	Price	U	1.0 - U	Price
K	na	U (= 0)	K	Spread
K	na	U (≠ 0)	(not allowed)	(not allowed)
K	Spread	U (= 0)	K	Spread
K	Spread	U (≠ 0)	(not allowed)	(not allowed)
K	Price	U (= 0)	K	Price
K	Price	U (≠ 0)	(not allowed)	(not allowed)

Table 88: Effective strike and strike type to be used in an Index CDS Option dependent on the Strike, StrikeType and UpfrontFee in the underlying Index CDS

19.2.55 Synthetic CDO

Payoff

A synthetic CDO is a basket credit derivative, where the protection seller receives a premium cash flow in exchange for providing (notional) protection against portfolio losses due to defaults in a specific tranche characterized by the attachment point A and detachment point D. Attachment point A is the fraction of the portfolio loss-given-default (LGD) where the protection starts, and the detachment point is the corresponding fraction where protection stops. The LGD is given by the sum of defaulted asset notional amounts N_i, reduced by their respective recovery rate R_i,i.e. $LGD_i = (1 - R_i)\, N_i$.

CDOs can refer to the constituents of an index such as CDX or iTraxx, or be bespoke, i.e. refer to a bespoke basket of underlying credit names, using the `BasketData` sub-node, (see section 19.3.28)

Input

A Synthetic Collateralized Debt Obligation (CDO), uses trade type *SyntheticCDO* and is set up using a `CdoData` block as shown in listing 122.

Listing 122: CDO Data

```
<CdoData>
  <Qualifier>RED:2I65BRHH6</Qualifier>
  <AttachmentPoint>0.12</AttachmentPoint>
  <DetachmentPoint>0.22</DetachmentPoint>
  <ProtectionStart> 20140425 </ProtectionStart>
  <UpfrontDate/>
  <UpfrontFee/>
  <SettlesAccrual>Y</SettlesAccrual>
  <ProtectionPaymentTime>atDefault</ProtectionPaymentTime>
  <!-- Premium leg -->
  <LegData>
   <LegType>Fixed</LegType>
   <Payer>true</Payer>
      ...
  </LegData>
  <BasketData>
    ...
  </BasketData>
</CdoData>
```

The meanings of the elements of the `CdoData` node follow below:

- Qualifier: The identifier of the credit index defining the default and base correlation curves used for pricing. In the case of a bespoke basket, i.e. when the `BasketData` sub-node is used, the Qualifer should be set to the credit index most closely matching the bespoke basket.

  Allowable values: See `CreditCurveId` for credit trades - index in Table 30.

- AttachmentPoint: Losses where protection starts, expressed as a fraction of the basket notional. Note that Attachment- and DetachmentPoints (AP, DP) are defined as fractions of the current basket notional.

  Allowable values: A number between 0 and 1, below the DetachmentPoint.

- DetachmentPoint: Losses where protection end, expressed as a fraction of the basket notional

  Note that Attachment- and DetachmentPoints (AP, DP) are defined as fractions of the current basket notional.

  Allowable values: A number between 0 and 1, above the AttachmentPoint.

- SettlesAccrual: Whether or not the accrued coupon is due in the event of a default.

  Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36.

- ProtectionPaymentTime [Optional]: Controls the payment time of protection and premium accrual payments in case of a default event. Defaults to `atDefault`.

  Allowable values: *atDefault*, *atPeriodEnd*, *!atMaturity*. Overrides the `PaysAtDefaultTime` node

- PaysAtDefaultTime [Deprecated]: *true* is equivalent to ProtectionPaymentTime = atDefault, *false* to ProtectionPaymentTime = atPeriodEnd. Overridden by the `ProtectionPaymentTime` node if set

  Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36.

- ProtectionStart: The first date where a default event will trigger the contract

  Allowable values: See `Date` in Table 20. Must be set to a date before or on the first date in the premium leg schedule.

- UpfrontDate[Optional]: Settlement date for the UpfrontFee if an UpfrontFee is provided. If an UpfrontFee is provided and it is non-zero, UpfrontDate is required.

  Allowable values: See `Date` in Table 20. The UpfrontDate, if provided, must be on or after the ProtectionStart date.

- UpfrontFee[Optional]: The upfront payment, expressed as a rate, to be multiplied by the tranche `Notional` amount. Note that a positive amount indicates that the UpfrontFee is paid by the protection buyer to the protection seller, and a negative amount indicates that the UpfrontFee is paid by the protection seller to the protection buyer. The UpfrontFee cannot be left blank.

  Allowable values: Any real number

- FixedRecoveryRate[Optional]: If provided, this recovery rate will be used in palce of the market quoted recovery rates of the underlying basket or index constituents, to work out the portfolio loss distribution and expected tranche loss.

  Allowable values: Any real number in the range [0, 1]

- LegData: Premium leg description as in an Index CDS (see section 19.2.53) with `Notional` corresponding to the initial tranche notional.
  Note that the `Payer` field in `LegData` determines whereas protection is bought (*true*) or sold (*false*).
  The `StartDate` in `LegData` is the first accrual start date on the premium leg of the index tranche. If the date generation `Rule` is *CDS2015*, one can enter the index tranche trade date for StartDate and the correct accrual start date will be deduced, i.e. the first accrual start date before the trade date using the *CDS2015* date generation rules.

- BasketData[Optional]: Underlying basket description for bespoke baskets (see section 19.3.28). This is analogous to a bespoke basket in an Index CDS (see section 19.2.53). If omitted, CreditIndex static data, with *id* `Qualifier` element in `CdoData`, is extracted from ReferenceData.

  Note that the sum of notionals of the basket components must add up to the complete basket notional.

  $$SumofComponentNotionals = CompleteBasketNotional = InitialTrancheNotional/(DetachmentPoint - AttachmentPoint)$$

  If weights are used instead of notionals in the basket components, the sum of the weights must add up to 1.

19.2.56 Risk Participation Agreement (RPA)

A risk participation agreement is set up using the trade type `RiskParticipationAgreement` and a `RiskParticipationAgreementData` block as shown in listing 123. The block contains a `ProtectionFee` block that can include one or more legs representing the fees paid by the protection buyer and an `Underlying` block containing either the legs of the underlying swap or the Treasury-Lock data that the contract references.

If the underlying reference entity defaults, the protection buyer receives the PV of the underlying if this is positive. Here, the underlying PV is computed using the payer / receiver flags as set up for the legs under the underlying node. Whether the trade represents a protection buyer or seller position is indicated by the payer flag in the protection fee leg data: If true protection is bought (and the protection fee is paid), if false the protection is sold (and the protection fee is received).

Listing 123: Risk Participation Agreement Data

```
<RiskParticipationAgreementData>
  <ParticipationRate>0.8</ParticipationRate>
  <ProtectionStart>2018-10-01</ProtectionStart>
  <ProtectionEnd>2038-10-01</ProtectionEnd>
  <CreditCurveId>RED:008CA0|SNRFOR|USD|MR14</CreditCurveId>
  <IssuerId>CompanyXZY</IssuerId>
  <SettlesAccrual>true</SettlesAccrual>
  <FixedRecoveryRate>0.6</FixedRecoveryRate>
  <ProtectionFee>
    <LegData>
      <LegType>Cashflow</LegType>
      <Payer>true</Payer>
      <Currency>EUR</Currency>
      <CashflowData>
        <Cashflow>
          <Amount date="2018-10-03">91171.72</Amount>
        </Cashflow>
      </CashflowData>
    </LegData>
  </ProtectionFee>
  <Underlying>
    <!-- Alternatives:
         - Sequence of LegData, possibly with OptionData to represent callability
         - A single block of TreasuryLockData -->
    <OptionData> ... </OptionData>
    <NakedOption> ... </NakedOption>
    <LegData>
      <LegType>Floating</LegType>
      ...
    </LegData>
    <LegData>
      <LegType>Fixed</LegType>
      <Payer>false</Payer>
      ...
    </LegData>
  </Underlying>
</RiskParticipationAgreementData>
```

- ParticipationRate: The rate reflecting the participation amount relative to the swap volume.

  Allowable values: Any number between 0 and 1.

- ProtectionStart: The date on which the protection starts (inclusive).

  Allowable values: Any valid date, see See `Date` in Table 20.

- ProtectionEnd: The date on which the protection ends (exclusive).

  Allowable values. Any valid date greater than the protection start date.

- CreditCurveId: Typically the RED-code of the underlying swap reference entity defining the default curve used for pricing. Other identifiers may be used as well, provided they are supported in the market data configuration.

  Allowable values: Any valid credit curve identifier.

- IssuerId [Optional]: An identifier for the underlying swap reference entity. For informational purposes and not used for pricing. Defaults to an empty string.

  Allowable values: Any string.

- SettlesAccrual [Optional]: Whether or not the accrued coupon of the protection fee is due in the event of a default. This defaults to `true` if not provided. Only applies to coupon legs (i.e. not simple cashflows) within the protection fee block, otherwise it is ignored.

  Allowable values: `true` or `false`

- FixedRecoveryRate [Optional]: This node holds the fixed recovery rate if the RPA assumes a fixed recovery to calculate the settlement amount in case of a default event. If the field is omitted the recovery rate associated to the credit curve is used instead.

  Allowable values: Any number between 0 and 1.

- ProtectionFee: The fees that are paid (if protection is bought) or received (if protection is sold). The fees are given by one or more legs as described under 19.3.3 with identical Payer flags, typically this will be a single `Cashflow` leg holding zero or more fixed fee amounts or a `Fixed` leg representing a series of periodic fee payments. Fees are paid up to (but excluding) the default event. If the fees are given as coupons the accrued amount between the accrual start date and the default date is paid if and only if `SettlesAccrual` is set to `true`. The protection fees can be given in any arbitrary currency.

- Underlying: The reference underlying. There are several subtypes to distinguish, all of which have separate pricing engines attached. There is no need to specify the subtype in the trade xml, this is deduced automatically during the trade building:

  - Vanilla Swap: This is a vanilla swap given by two legs in the same currency, one receiver, one payer and one Fixed (or Cashflow), one Floating. For the floating part only Ibor coupons (no averaging) or (compounded, averaging) OIS coupons are allowed. Spreads and gearings are allowed, but no embedded caps/floors, no in arrears fixings for Ibor coupons. This type allows an analytic Black engine where the RPA Options are found via a representative swaption matching.

- Structured Swap: As vanilla, but an arbitrary number of legs of type Fixed, Floating, Cashflow is allowed. Embedded caps/floors/collars and in arrears fixing are allowed. For floating legs, Ibor (no averaging) and OIS (compounded, averaging) coupons are allowed. All legs must be in the same currency. Standalone caps, floors, collars are allowed as an underlying of the RPA, if specified by a floating leg with NakedOption set to true. See 19.3.6 for details on the floating leg specification, amd likewise 19.3.5 for the fixed leg and 19.3.3 for the cashflow leg. This type requires a numeric grid engine.
- Callable Swap / Swaption: As structured swap, but an additional OptionData block allows to specify callability of the swap. The relevant fields in OptionData are the same as for callable swaps, see 19.2.6. This type requires a numeric grid engine as the structured swap. If NakedOption is set to true, an option to exercise into the underlying swap is represented, i.e. a swaption.
- Cross Currency Swap: Underlying legs as in structured swap, but the legs can be in two different currencies. No optionality is allowed though. At most two different currencies are allowed. This type can be priced using an analytic Black engine which models the FX Risk and assumes deterministic interest rates.
- T-Lock. The underlying is a T-Lock, represented as shown in listing 124 and explained in more detail below. This type requires a numeric grid engine.

Treasury Lock Underlying Specification

Listing 124 shows the specification of a T-Lock underlying. The fields have the following meaning:

- Payer: Boolean, true if the fixed reference rate is paid, false otherwise. I.e. if the payer flag is true and the yield is lower than the reference rate, then the underlying T-Lock trade pays the amount $(r - y) \cdot d$ where r is the reference rate, y is the yield, both expressed in basis points, and $d > 0$ is the (absolute) price change of the treasury bond when the yield moves by 1 basis point. Likewise, if the yield is higher than the reference rate, the underlying T-Lock trade receives $(y - r) \cdot d$.
  Allowable values: *true* or *false*
- BondData: Reference to the underlying security, given in in the BondData sub node, minimum required data are notional and security ID
  Allowable values: See 19.2.40
- ReferenceRate: Fixed rate paid or received on the T-Lock underlying
  Allowable values: Any real number. The rate is expressed in decimal form, eg 0.05 is a rate of 5%
- DayCounter [Optional]: Reference rate day counter. Optional, defaults to the coupon day counter of the underlying bond.
  Allowable values: See Table 25
- TerminationDate: Date for the cash settlement amount calculation
  Allowable values: See `Date` in Table 20.
- PaymentGap [Optional]: Business day gap between termination and payment date. Optional, defaults to zero.
  Allowable values: Any non-negative integer

- PaymentCalendar: Calendar to determine the payment date.
  Allowable values: See Table 24.

Listing 124: Treasury-Lock Data

```
<Underlying>
  <TreasuryLockData>
  <Payer>true</Payer>
  <BondData>
  ...
  </BondData>
  <ReferenceRate>0.05</ReferenceRate>
  <DayCounter>A360</DayCounter>
  <TerminationDate>2022-01-05</TerminationDate>
  <PaymentGap>5</PaymentGap>
  <PaymentCalendar>US</PaymentCalendar>
  </TreasuryLockData>
</Underlying>
```

19.2.57 Credit Linked Swap

A credit linked swap, trade type `CreditLinkedSwap`, is set up using a `CreditLinkedSwapData` block as shown in listing 125. The elements have the following meaning:

- CreditCurveId: The referenced CDS credit curve.
  Allowable values: See `CreditCurveId` for credit trades - single name in Table 30. A `ReferenceInformation` node may be used in place of this `CreditCurveId` node.
- SettlesAccrual [Optional]: A flag indicating whether accrued coupon amounts are paid in case of a credit event. Optional, defaults to `true`. Applies to the payments specified under ContingentPayments.
  Allowable values: true, false
- FixedRecoveryRate [Optional]: A fixed (digital) recovery rate to apply. If not given, the market recovery rate is used. Applies to the payments specified under DefaultPayments and RecoveryPayments.
  Allowable values: Any non-negative real number.
- DefaultPaymentTime [Optional]: Controls the timing of the payments specified under DefaultPayments and RecoveryPayments. Defaults to `atDefault`.
  Allowable values: `atDefault`, `atPeriodEnd`, `atMaturity`.
- IndependentPayments [Optional]: The legs for which payments are made independent from credit events. The node contains one or more `LegData` subnodes representing these legs. Optional, can be omitted if no such payments are made.
  Allowable values: See 19.3.3 for the `LegData` subnode structure.
- ContingentPayments [Optional]: The legs for which payments are contingent on no credit event having occured until the payment date. If no such payments are made, the node can be omitted.
  Allowable values: See 19.3.3 for the `LegData` subnode structure.
- DefaultPayments [Optional]: The legs for which payments are contingent on a credit event having occured. If no such payments are made, the node can be omitted. If a

default happens at a date d, the associated payment is the earliest payment with date greater or equal to d.
Allowable values: See 19.3.3 for the `LegData` subnode structure.

- RecoveryPayments [Optiopnal]: The legs for which payments are contingent on a credit event having occured. The node works analogously to the DefaultPayments node, the only difference is that that payment amounts are weighted by RR instead of $1 - RR$.
Allowable values: See 19.3.3 for the `LegData` subnode structure.

All legs must be given in the same currency.

Listing 125: Credit Linked Swap Data

```
<CreditLinkedSwapData>
  <CreditCurveId>RED:46A844|SNRFOR|USD|XR14</CreditCurveId>
  <SettlesAccrual>false</SettlesAccrual>
  <FixedRecoveryRate>0.4</FixedRecoveryRate>
  <DefaultPaymentTime>atDefault</DefaultPaymentTime>
  <IndependentPayments>
    <LegData> ... </LegData>
    <LegData> ... </LegData>
    ...
  </IndependentPayments>
  <ContingentPayments>
    <LegData> ... </LegData>
    <LegData> ... </LegData>
    ...
  </ContingentPayments>
  <DefaultPayments>
    <LegData> ... </LegData>
    <LegData> ... </LegData>
    ...
  </DefaultPayments>
  <RecoveryPayments>
    <LegData> ... </LegData>
    <LegData> ... </LegData>
    ...
  </RecoveryPayments>
</CreditLinkedSwapData>
```

19.2.58 Commodity Forward

A Commodity Forward contract is an agreement between two counterparties to buy/sell a set amount of a commodity, at a predetermined price (the strike), at the end of the contract. A commodity forward does not involve any upfront payment.

The `CommodityForwardData` node is the trade data container for the `CommodityForward` trade type. The structure of an example `CommodityForwardData` node is shown in Listings 126 and 127.

Listing 126: Commodity Forward data

```
<CommodityForwardData>
  <Position>Long</Position>
  <Maturity>2029-06-30</Maturity>
  <Name>XCEC:GC</Name>
  <Currency>USD</Currency>
  <Strike>1355</Strike>
  <Quantity>1000</Quantity>
  <IsFuturePrice>...</IsFuturePrice>
  <FutureExpiryDate>...</FutureExpiryDate>
  <FutureExpiryOffset>...</FutureExpiryOffset>
  <FutureExpiryOffsetCalendar>...</FutureExpiryOffsetCalendar>
  <PhysicallySettled>...</PhysicallySettled>
  <PaymentDate>...</PaymentDate>
</CommodityForwardData>
```

Listing 127: `CommodityForwardData` for forward on LME Aluminium 3M future.

```
<CommodityForwardData>
  <Position>Long</Position>
  <Maturity>2029-08-16</Maturity>
  <Name>XLME:AH</Name>
  <Currency>USD</Currency>
  <Strike>2160</Strike>
  <Quantity>1000</Quantity>
  <IsFuturePrice>true</IsFuturePrice>
  <FutureExpiryDate>2021-11-16</FutureExpiryDate>
  <PhysicallySettled>true</PhysicallySettled>
</CommodityForwardData>
```

The meanings and allowable values of the elements in the `CommodityForwardData` node follow below.

- Position: Defines whether the underlying commodity will be bought (long) or sold (short).
  Allowable values: *Long, Short*
- Maturity: The maturity date of the forward contract, i.e. the date when the underlying commodity will be bought/sold.
  Allowable values: Any date string, see `Date` in Table 20.
- Name: The name of the underlying commodity.
  Allowable values: See `Name` for commodity trades in Table 32.
- Currency: The currency the underlying commodity is quoted in. The Strike and the Forward price (or Future price) of the underlying commodity are both considered to be in this currency.
  Allowable values: See `Currency` in Table 20.
- Strike: The agreed buy/sell price of the commodity forward.
  Allowable values: Any positive real number.
- Quantity: The number of units of the underlying commodity to be bought/sold.

Allowable values: Any positive real number.

- `IsFuturePrice` [Optional]: This should be set to `true` if the forward contract underlying is the settlement price of a commodity future contract. If omitted, it defaults to `false`.
  Allowable values: Any string that evaluates to true or false as outlined in Table 36.
- `FutureExpiryDate` [Optional]: If `IsFuturePrice` is set to `true`, this gives the expiration date of the underlying commodity future contract. If omitted, the expiration date of the underlying commodity future contract is set equal to the value in the `Maturity` node. If `FutureExpiryDate` is provided, it takes precedence over any value provided in the `Maturity`, `FutureExpiryOffset` or `FutureExpiryOffsetCalendar` fields.
  Allowable values: Any date string, see `Date` in Table 20.
- `FutureExpiryOffset` [Optional]: If `IsFuturePrice` is set to `true` and `FutureExpiryDate` is not explicitly specified, this gives the offset period that should be applied to the `Maturity` date to generate the underlying commodity future contract expiration date. If omitted, the expiration date of the underlying commodity future contract is set equal to the value in the `Maturity` node.
  Allowable values: Any string that can be parsed as a period e.g. `2D`, `3M`, etc.
- `FutureExpiryOffsetCalendar` [Optional]: If `FutureExpiryOffset` is provided and is being used, this gives the calendar that should be used when generating the underlying commodity future contract expiration date from the `Maturity` date. If omitted, all days are considered good business days when generating the commodity future contract expiration date which is generally not what is desired.
  Allowable values: Any calendar string, see `Calendar` in Table 24.
- `PhysicallySettled` [Optional]: A value of `true` indicates that the forward contract is physically settled e.g. if the underlying is a future contract, that future contract is entered into on the `Maturity` date. A value of `false` indicates that the forward contract is cash settled e.g. if the underlying is a future contract, that future contract settlement price is observed on the `Maturity` date (or the `FutureExpiryDate`, when given) and the net amount due is exchanged on the cash settlement date. If omitted, it defaults to *true*.
  Allowable values: Any string that evaluates to true or false as outlined in Table 36.
- `PaymentDate` [Optional]: If `PhysicallySettled` is set to *false*, this gives the cash settlement date. It must be greater than or equal to the `Maturity` date. If omitted and the forward is cash settled, the `Maturity` date is used.
  Allowable values: Any date string, see `Date` in Table 20.
- `SettlementData` [Optional]: This node is used to specify the settlement of the cash flows for cash settled forwards, and the payment flow for physically settled ones.

A `SettlementData` node is shown in Listing 128, and the meanings and allowable values of its elements follow below.

- `PayCurrency`: The settlement currency for the payment cashflow.
  Allowable values: See `Currency` in Table 20.
- `FXIndex`: The FX reference index for determining the FX fixing at the value date. The

Forward Price will be observed at maturity date (or future expiry date if it's a future), the NPV is converted to `PayCurrency` with the `FXIndex` using an FX fixing on `FixingDate` (settlement date) discounted from `PaymentDate`.
Allowable values: The format of the `FXIndex` is "FX-FixingSource-CCY1-CCY2" as described in Table 28.

- `FixingDate`: The date on which the *FXIndex* is observed. Allowable values: See `Date` in Table 20.

Listing 128: Example `SettlementData` node with `Rules` sub-node

```
<SettlementData>
  <PayCurrency>EUR</PayCurrency>
  <FXIndex>FX-ECB-EUR-USD</FXIndex>
  <FixingDate>2025-05-28</FixingDate>
</SettlementData>
```

Note that a Precious Metal Forward should be represented as an FX Forward using the appropriate commodity "currency" (XAU, XAG, XPT, XPD).

19.2.59 Commodity Swap and Basis Swap

The structure of a `CommoditySwap` trade node is shown in listing 129. This trade node can be used to represent commodity swaps and commodity basis swaps. It consists of the generic `Envelope` and the specific `SwapData` section.

The `SwapData` node may contain two or more `LegData` nodes. There must be at least one `LegData` node of a commodity `LegType`, i.e. `CommodityFixed` or `CommodityFloating`, but non-commodity leg types are also allowed. The commodity leg types are described in sections 19.3.20 and 19.3.22 respectively.

Listing 129: Commodity Swap

```
<Trade id="...">
  <TradeType>CommoditySwap</TradeType>
  <Envelope>
  </Envelope>
  <SwapData>
    <LegData>
      <LegType>CommodityFixed</LegType>
      ...
    </LegData>
    <LegData>
      <LegType>CommodityFloating</LegType>
      ...
    </LegData>
  </SwapData>
</Trade>
```

19.2.60 Commodity Swaption

The structure of a trade node representing a commodity swaption is shown in listing 130. It consists of the generic `Envelope` and the specific `CommoditySwaptionData` node.

The `CommoditySwaptionData` node contains an `OptionData` node described in 19.3.1. The relevant fields in the `OptionData` node for a CommoditySwaption are:

- `LongShort`: The allowable values are *Long* or *Short*. Note that the payer and receiver legs in the underlying swap are always from the perspective of the party that is *Long*. E.g. for a *Short* CommoditySwaption with a fixed leg where the Payer flag is set to *false*, it means that the counterparty receives the fixed flows.
- `OptionType`[Optional]: This flag is optional for CommoditySwaptions, and even if set, has no impact. The direction of flows is determined entirely by the Payer flags on the underlying legs (and the `LongShort` flag above).
- `Style`: The exercise style of the CommoditySwaption. Only exercise style *European* is supported.
- `NoticePeriod`[Optional]: The notice period defining the date (relative to the exercise date) on which the exercise decision has to be taken. If not given the notice period defaults to *0D*, i.e. the notice date is identical to the exercise date. Allowable values: A number followed by *D, W, M, or Y*
- `NoticeCalendar`[Optional]: The calendar used to compute the notice date from the exercise date. If not given defaults to the *NullCalendar* (no holidays, weekends are no holidays either). Allowable values: See Table 24 `Calendar`.
- `NoticeConvention`[Optional]: The roll convention used to compute the notice date from the exercise date. Defaults to *Unadjusted* if not given. Allowable values: See Table 21 Roll Convention.
- `Settlement`: Delivery Type. The allowable values are *Cash* or *Physical*.
- `ExerciseFees`[Optional]: This node contains child elements of type `ExerciseFee`. Similar to a list of notionals (see 19.3.3) the fees can be given either
  - as a list where each entry corresponds to an exercise date and the last entry is used for all remaining exercise dates if there are more exercise dates than exercise fee entries, or
  - using the `startDate` attribute to specify a change in a fee from a certain day on (w.r.t. the exercise date schedule)

  Fees can either be given as an absolute amount or relative to the current notional of the period immediately following the exercise date using the `type` attribute together with specifiers `Absolute` resp. `Percentage`. If not given, the type defaults to `Absolute`. `Percentage` fees are expressed in decimal form, e.g. 0.05 is a fee of 5% of notional.

  If a fee is given as a positive number the option holder has to pay a corresponding amount if they exercise the option. If the fee is negative on the other hand, the option holder receives an amount on the option exercise.
- `ExerciseFeeSettlementPeriod`[Optional]: The settlement lag for exercise fee payments. Defaults to 0D if not given. This lag is relative to the exercise date (as opposed to the notice date). Allowable values: A number followed by *D, W, M, or Y*
- `ExerciseFeeSettlementCalendar`[Optional]: The calendar used to compute the exercise fee settlement date from the exercise date. If not given defaults to the

NullCalendar (no holidays, weekends are no holidays either). Allowable values: See Table 24 Calendar.

- `ExerciseFeeSettlementConvention`[Optional]: The roll convention used to compute the exercise fee settlement date from the exercise date. Defaults to *Unadjusted* if not given. Allowable values: See Table 21 Roll Convention.
- An `ExerciseDates` node where exactly one `ExerciseDate` date element must be given for *European* style CommoditySwaptions. Allowable values: The `ExerciseDate` must be on or before the StartDate of the underlying legs, and be on or after the valuation date. For the format, see Date in Table 20.
- `Premiums` [Optional]: Option premium node with amounts paid by the option buyer to the option seller.

  Allowable values: See section 19.3.2

The `CommoditySwaptionData` node should contain exactly two `LegData` nodes. One `LegData` node should be of type `CommodityFixed` described in section 19.3.20 and one should be of type `CommodityFloating` described in section 19.3.22. Note that on the `CommodityFloating` leg, the Spread must be omitted or set to *0*, and the Gearing must be omitted or set to *1*.

Listing 130: Commodity swaption

```
<Trade id="...">
  <TradeType>CommoditySwaption</TradeType>
  <Envelope>
    ...
  </Envelope>
  <CommoditySwaptionData>
    <OptionData>
      <LongShort>Long</LongShort>
      <Style>European</Style>
      <Settlement>Cash</Settlement>
      <ExerciseDates>
        <ExerciseDate>2023-01-05</ExerciseDate>
      </ExerciseDates>
    </OptionData>
    <LegData>
      <LegType>CommodityFixed</LegType>
      ...
    </LegData>
    <LegData>
      <LegType>CommodityFloating</LegType>
      ...
    </LegData>
  </CommoditySwaptionData>
</Trade>
```

19.2.61 Commodity Option

A European Commodity Option gives the buyer the right, but not the obligation, to buy a set amount of a commodity, at a predetermined price (the strike), at the end of the contract. For

this right the buyer pays a premium to the seller. Settlement can be either cash or physical delivery.

The `CommodityOptionData` node is the trade data container for the *CommodityOption* trade type. Vanilla commodity options are supported. The exercise style may be *European* or *American*. The `CommodityOptionData` node includes exactly one `OptionData` trade component sub-node plus elements specific to the commodity option. The structure of a `CommodityOptionData` node for a commodity option is shown in Listing 131.

Listing 131: Commodity Option data

```
<CommodityOptionData>
  <OptionData>
   <LongShort>Short</LongShort>
   <OptionType>Put</OptionType>
   <Style>European</Style>
   <Settlement>Cash</Settlement>
   <PayOffAtExpiry>false</PayOffAtExpiry>
    <ExerciseDates>
      <ExerciseDate>2029-04-28</ExerciseDate>
     </ExerciseDates>
  </OptionData>
  <Name>NYMEX:CL</Name>
  <Currency>USD</Currency>
  <StrikeData>
    <StrikePrice>
      <Value>100</Value>
      <Currency>USD</Currency>
    </StrikePrice>
  </StrikeData>
  <Quantity>500000</Quantity>
  <IsFuturePrice>true<IsFuturePrice>
  <FutureExpiryDate>2029-04-28<FutureExpiryDate>
</CommodityOptionData>
```

The meanings and allowable values of the elements in the `CommodityOptionData` node follow below.

- The `CommodityOptionData` node contains an `OptionData` node described in 19.3.1. The relevant fields in the `OptionData` node for a CommodityOption are:
  - `LongShort`: The allowable values are *Long* or *Short*.
  - `OptionType`: The allowable values are *Call* or *Put*.
  - `Style`: The exercise style of the CommodityOption. The allowable values are *European* or *American*.
  - `PayOffAtExpiry`: This must be set to *false* as payoff at expiry is not currently supported.
  - An `ExerciseDates` node where exactly one `ExerciseDate` date element must be given for. Allowable values: See Date in Table 20.
  - `Premiums` [Optional]: Option premium node with amounts paid by the option buyer to the option seller. Allowable values: See section 19.3.2

- Name: The name of the underlying commodity.
  Allowable values: See `Name` for commodity trades in Table 32.
- Currency: The currency of the commodity option.
  Allowable values: See `Currency` in Table 20.
- StrikeData: The option strike price. It uses the price quotation outlined in the underlying contract specs for the commodity name in question.
  Allowable values: Only supports `StrikePrice` as described in Section 19.3.30.
- Quantity: The number of units of the underlying commodity covered by the transaction. The unit type is defined in the underlying contract specs for the commodity name in question. For avoidance of doubt, the Quantity is the number of units of the underlying commodity, not the number of contracts.
  Allowable values: Any positive real number.
- IsFuturePrice [Optional]: A boolean indicating if the underlying is a future contract settlement price, `true`, or a spot price, `false`.

  Allowable values: A boolean value given in Table 36. If not provided, the default value is `true`.
- FutureExpiryDate [Optional]: If `IsFuturePrice` is `true` and the underlying is a future contract settlement price, this node allows the user to specify the expiry date of the underlying future contract.

  Allowable values: This should be a valid date as outlined in Table 20. If not provided, it is assumed that the future contract's expiry date is equal to the option expiry date provided in the `OptionData` node.

19.2.62 Commodity Digital Option

A commodity digital option is represented with trade type *CommodityDigitalOption* and a corresponding `CommodityDigitalOptionData` node. The latter differs from the `CommodityOptionData` node in section 19.2.61 by replacing tag *Quantity* with tag *Payoff* which is the cash amount paid in the Currency of the option from the party that is short to the party that is long, when the underlying price exceeds the strike at expiry in case of a Call (or falls below the strike in case of a Put). The digital option is priced in VRE as a spread of vanilla Commodity options at two slightly different strikes. For option type *Call* and *Put*, respectively, the digital call/put is constructed as

$$\text{Digital Call} = \frac{\text{Payoff}}{\Delta} \times (\text{Call}(K - \Delta/2) - \text{Call}(K + \Delta/2))$$
$$\text{Digital Put} = \frac{\text{Payoff}}{\Delta} \times (\text{Put}(K + \Delta/2) - \text{Put}(K - \Delta/2))$$

so that the long digital option has positive value in both cases. The strike spread Δ used here is set to 1% of strike K. However, the default 1% can be overriden by setting the 'StrikeSpread' parameter in the pricing engine global parameters.

19.2.63 Commodity Spread Option

A commodity Spread Option is represented with trade type *CommoditySpreadOption* and a corresponding `CommoditySpreadOptionData` node.

The CommoditySpreadOptionData node is the trade data container for the *CommoditySpreadOption* trade type. The structure of a CommoditySpreadOptionData node for a commodity option is shown in Listing 132.

The CommoditySpreadOptionData include exactly two LegData nodes of type *CommodityFloating*. Details on these are described in 19.3.22. The resulting Legs must produce the same amount of cashflows (i.e. the number of *calculation period*s must be the same for the long and short positions). If the number of cashflows per leg is 1, this trade represents a single commodity spread option. If is greater than 1, it represents a multi-period commodity spread option. Exactly one payer and one receiver leg are required, the leg with isPayer setto *true* is the long (positive) position in the spread payoff.

Within the two LegData, the Quantity node has must be equal. If the underlying contracts are quoted using different units (e.g. barrels vs liters), the Gearing node must be used to account for this difference. The gearing could also be used for the heat rate factor in spark / heat rate options.

Other than the two legs, the following nodes complete the CommoditySpreadOptionData container:

- SpreadStrike: The strike value for the spread. Allowable values: Any real number.
- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the OptionData node for an CommoditySpreadOption are
  - LongShort The allowable values are *Long* or *Short*.
  - OptionType The allowable values are *Call* or *Put*.
  - Premiums [Optional]: Option premium amounts paid by the option buyer to the option seller. See section 19.3.2
  - ExerciseDates [Optional]: If given, there must be one exercise date for each option period. If omitted, the option expiry is derived from the cashflows and falls on the last pricing period of the commodity cashflow.
  - PaymentData [Optional]: This node is used to supply the date on which the option is cash settled if it is exercised. If omitted the settlement date is derived from the cashflow payment dates.
- OptionStripPaymentDates [Optional]: If the number of cashflows per leg is greater than 1, we can group options by their expiry date into strips. All option in a strip will have the same payment date as defined in this node. The payment date will be *lag* business days after the latest expiry date in the strip. The node has following sub-nodes:
  - OptionStripDefinition A schedule node 19.3.4 defining the option strips. The n dates in the schedule defining $n-1$ strips, each strip include the period's start date and excludes period's end date. All options with expiry within start and end of a period are falling in the same strip. The schedule has to cover all option expiries. The first date in the schedule has to be before or on the first expiry date of the options and the last date in the schedule has to be after last expiry date of the options.
  - PaymentCalendar Calendar defining valid business days for the payment date.

- PaymentLag number of business days.
- PaymentConvention business day convention for the option strip payment date.

Listing 132: Commodity Spread Option data

```
<CommoditySpreadOptionData>
  <LegData>
   <LegType>CommodityFloating</LegType>
   <IsPayer>true<IsPayer>
   ...
  </LegData>
  <LegData>
   <LegType>CommodityFloating</LegType>
   <IsPayer>false<IsPayer>
   ...
  </LegData>
  <OptionData>
    <LongShort>Long</LongShort>
    <OptionType>Call</OptionType>
     <Premiums>
       <Premium>
         <Amount>10900</Amount>
         <Currency>EUR</Currency>
         <PayDate>2020-03-01</PayDate>
       </Premium>
     </Premiums>
  </OptionData>
  <SpreadStrike>2.3</SpreadStrike>
  <OptionStripPaymentDates>
          <OptionStripDefinition>
          <Rules>
            <StartDate>2023-07-01</StartDate>
            <EndDate>2023-10-01</EndDate>
            <Tenor>1M</Tenor>
            <Calendar>NullCalendar</Calendar>
            <Convention>Unadjusted</Convention>
            <TermConvention>Unadjusted</TermConvention>
            <Rule>Backward</Rule>
          </Rules>
          </OptionStripDefinition>
    <PaymentCalendar>ICE_FuturesUS,US-NERC</PaymentCalendar>
    <PaymentLag>5</PaymentLag>
    <PaymentConvention>MF</PaymentConvention>
  </OptionStripPaymentDates>
</CommoditySpreadOptionData>
```

19.2.64 Commodity Average Price Option

The structure of a trade node representing a commodity average price option (APO) is shown in listing 133. It consists of the generic Envelope and the specific CommodityAveragePriceOptionData node. A strip of these options may be booked using the commodity option strip trade outlined in section 19.2.65. The meanings and allowable values of the elements in the CommodityAveragePriceOptionData are as follows:

- OptionData: This node is described in section 19.3.1. The relevant fields in the OptionData node for a CommodityAveragePriceOption are:

  - `LongShort`: The allowable values are *Long* or *Short*.
  - `OptionType`: The allowable values are *Call* or *Put*, where *Call* is an option for the party that is *Long* to buy the underlying commodity, and *Put* is an option to sell the underlying commodity.
  - `Style`: only `European` exercise style is supported.
  - the `ExerciseDates` node should contain exactly one `ExerciseDate`, and the single `ExerciseDate` should be on or before the payment date. Also, the single `ExerciseDate` should be on or after the final date in the calculation period i.e. the `EndDate` below.
  - `Premiums` [Optional]: Option premium node with amounts paid by the option buyer to the option seller. Allowable values: See section 19.3.2
- `BarrierData` [optional]: If given this node specifies the barrier terms of the option:
  - `Type`: One of *UpAndIn*, *DownAndIn*, *UpAndOut*, *DownAndOut*
  - `Style`: One of *European*, *American*. A European barrier is observed on the last relevant pricing date of the APO while an American barrier is observed on all pricing dates of the APO.
  - `LevelData`: The barrier level. Only single-barrier options are allowed, i.e. exactly one level must be given.
- `Name`: An identifier specifying the commodity being referenced. This is described in section 19.3.24.

  Allowable values: See `Name` for commodity trades in Table 32.
- `Currency`: The currency of the payoff which must be consistent with either currency of the market data set up for the commodity or the other currency specified in `FXIndex` (see below).

  Allowable values: See `Currency` in Table 20.
- `Quantity`: The number of units of the underlying commodity covered by the APO. The unit type is defined in the underlying contract specs for the commodity name in question. For avoidance of doubt, the Quantity is the number of units of the underlying commodity, not the number of contracts.

  The meaning of the Quantity is influenced by the `CommodityQuantityFrequency` value as described in section 19.3.24. If `CommodityQuantityFrequency` is set to `PerCalculationPeriod`, this quantity is used directly in the APO payoff. If `CommodityQuantityFrequency` is set to `PerCalendarDay`, this quantity is multiplied by the number of calendar days in the APO period to give the final quantity that is used in the APO payoff.

  Allowable values: Any positive real number.
- `StrikeData`: A `StrikeData` node is used as described in Section 19.3.30 to represent the APO strike price and Currency of the Strike.The APO strike price. The strike uses the price quotation outlined in the underlying contract specs for the commodity name in

question. Note that for CommodityAPOs only `StrikePrice` is supported within the `StrikeData` node, and not `StrikeYield`.

Allowable values: Only supports `StrikePrice` as described in Section 19.3.30.

- `PriceType`: The price type is `Spot` if the APO is referencing a commodity spot price, and it is `FutureSettlement` if the APO is referencing a commodity future contract settlement price.

  Allowable values: `Spot` or `FutureSettlement`.

- `StartDate`: The start date of the APO's calculation period.

  Allowable values: See `Date` in Table 20.

- `EndDate`: The end date of the APO's calculation period.

  Allowable values: See `Date` in Table 20.

- `PaymentCalendar`: The business calendar used to determine the valid payment date.

  Allowable values: See Table 24 `Calendar`.

- `PaymentLag`: The payment date is this number of business days after a given base date. The base date is determined by the value of the `CommodityPayRelativeTo` node below which is generally omitted for APOs and allowed to take its default value of `CalculationPeriodEndDate`.

  Allowable values: Any valid period, i.e. a non-negative whole number, optionally followed by *D* (days), *W* (weeks), *M* (months), *Y* (years). Defaults to *0D* if left blank or omitted. If a whole number is given and no letter, it is assumed that it is a number of *D* (days).

- `PaymentConvention`: The roll convention used to adjust the payment date.

  Allowable values: See Table 21 Roll Convention. Defaults to *Unadjusted* if left blank or omitted.

- `PricingCalendar`: The business calendar used for determining the *Pricing Dates* in the calculation period.

  Allowable values: See Table 24 `Calendar`. Defaults to *NullCalendar* (no holidays) if left blank or omitted.

- `PaymentDate` [Optional]: An explicit payment date for the APO if the combination of `PaymentCalendar`, `PaymentLag` and `PaymentConvention` is not sufficient. If `PaymentDate` is provided, it overrides the values provided in `PaymentCalendar`, `PaymentLag` and `PaymentConvention`.

- `Gearing` [Optional]: An optional gearing factor that the average price is multiplied by in the APO payoff. The default value is 1.0.

- `Spread` [Optional]: An optional spread that is added to the average price in the APO payoff. The default value is 0.0.

- `CommodityQuantityFrequency` [Optional]: This is as described in section 19.3.24.

- `CommodityPayRelativeTo` [Optional]: This is as described in section 19.3.24.

- `FutureMonthOffset` [Optional]: This is as described in section 19.3.24. Note that IsAveraged defaults to *false* as it cannot be used as a tag within the CommodityAveragePriceOptionData node. Thus, if e.g. FutureMonthOffset is set to *2*, the future contract month and year is taken as the second month following the base date's month and year; and so on for all positive values of FutureMonthOffset.
- `DeliveryRollDays` [Optional]: This is as described in section 19.3.24.
- `IncludePeriodEnd` [Optional]: This is as described in section 19.3.24.
- `FXIndex` [Optional]: This is an FX index used to apply currency conversion daily in the average. The currencies pair must include the currency used in the underlying commodity trade and the currency used for settlement.

  Allowable values: See Table 28 for supported fx indices.

19.2.65 Commodity Option Strip

The structure of a trade node representing a commodity option strip is shown in listing 134. This node can be used to represent a strip of commodity average price options as described in section 19.2.64 or a strip of European commodity options as described in section 19.2.61. It consists of the generic `Envelope` and the specific `CommodityOptionStripData` node.

The `CommodityOptionStripData` node has a `LegData` node with `LegType` set to `CommodityFloating`. This `LegData` node is described in detail in sections 19.3.22 and 19.3.24. Note that the `Payer` field in `CommodityFloatingLegData`, while mandatory, has no impact on flows. The node `IsAveraged` in `CommodityFloatingLegData` determines whether a strip of European commodity options or a strip of APOs are created:

- If `IsAveraged` is `false`, a strip of European commodity options is created. There is a European put and or European call option created for each calculation period. The exercise date of the option in the calculation period is given by the *Pricing Date* in the calculation period using the rules outlined in section 19.3.24. The quantity is given by the quantity in the calculation period using the rules outlined in section 19.3.24. If cash settled, the cash settlement date is given by the payment date for the calculation period using the rules outlined in section 19.3.24.
- If `IsAveraged` is `true`, a strip of commodity average price options is created. There is a put and or call option created for each calculation period. The exercise date of the option in the calculation period is given by the calculation period end date. The quantity is given by the quantity in the calculation period using the rules outlined in section 19.3.24.

Each calculation period may contain a put and a call that may be either bought or sold. The type of option, whether they are bought or sold and the strike price is determined by the `Calls` and `Puts` nodes. We describe here the settings for the `Calls` node with the understanding that analogous descriptions apply to the `Puts` node. If the `Calls` node is omitted, it is assumed that there are no call options in the strip.

The `LongShorts` node may contain one `LongShort` node or the same number of `LongShort` nodes as calculation periods. Each `LongShort` node has the allowable values `Long` or `Short`. If `LongShort` is `Long`, then the call option is bought and if `LongShort` is `Short` then the call option is sold. If a single `LongShort` node is provided, it is applied to all options in the strip.

Listing 133: Commodity average price option

```
<Trade id="...">
  <TradeType>CommodityAveragePriceOption</TradeType>
  <Envelope>
  ...
  </Envelope>
  <CommodityAveragePriceOptionData>
    <OptionData>
      <LongShort>Short</LongShort>
      <OptionType>Call</OptionType>
      <Style>European</Style>
      <ExerciseDates>
         <ExerciseDate>2020-01-31</ExerciseDate>
      </ExerciseDates>
    </OptionData>
    <BarrierData>
      <Type>UpAndIn</Type>
      <Style>American</Style>
      <LevelData>
        <Level>
          <Value>80</Value>
        </Level>
      </LevelData>
    </BarrierData>
    <Name>NYMEX:CL</Name>
    <Currency>USD</Currency>
    <Quantity>6000</Quantity>
    <StrikeData>
      <StrikePrice>
        <Value>80</Value>
        <Currency>USD</Currency>
      </StrikePrice>
    </StrikeData>
    <PriceType>FutureSettlement</PriceType>
    <StartDate>2022-01-01</StartDate>
    <EndDate>2023-01-31</EndDate>
    <PaymentCalendar>USD</PaymentCalendar>
    <PaymentLag>5</PaymentLag>
    <PaymentConvention>Following</PaymentConvention>
    <PricingCalendar>USD</PricingCalendar>
    <Gearing>1</Gearing>
    <Spread>0.0</Spread>
    <CommodityQuantityFrequency>PerCalculationPeriod</CommodityQuantityFrequency>
    <CommodityPayRelativeTo>CalculationPeriodEndDate</CommodityPayRelativeTo>
    <FutureMonthOffset>0</FutureMonthOffset>
    <DeliveryRollDays>0</DeliveryRollDays>
    <IncludePeriodEnd>true</IncludePeriodEnd>
    <FXIndex>FX-ECB-EUR-USD</FXIndex>
  </CommodityAveragePriceOptionData>
</Trade>
```

If the same number of `LongShort` nodes as calculation periods are provided, a `LongShort` node is applied to the option in the corresponding period. The optional `BarrierData` node specifies the barrier terms of the options. See section 19.2.64 for details on this. Call and put options can have different barrier terms, but all call (resp. put) options share the same terms.

In listing 134 only the call options have a barrier feature.

Similar to the `LongShorts` node, the `Strikes` node may contain one `Strike` node or the same number of `Strike` nodes as calculation periods. If only one is provided, this strike applies to all options in the strip. If the same number of `Strike` nodes as calculation periods are provided, a `Strike` node is applied to the option in the corresponding period. In this way, we support varying strikes across options in the strip. At least one of `Calls` or `Puts` needs to be provided for a valid option strip to be created.

The `Premiums` node allows for the addition of premiums. If the `PremiumAmount` is negative, it is paid and if it is positive, it is received. See 19.3.2.

The optional `Style` node can be set to `European` or `American` to change the exercise style for the strip of options. If not set, `European` is assumed. If the strip is a strip of APOs, `European` is assumed and a warning is issued if `Style` is not `European`.

The optional `Settlement` node can be set to `Cash` or `Physical` to change the settlement method for the strip of options. If not set, `Cash` is assumed. If the strip is a strip of APOs, `Cash` is assumed and a warning is issued if `Settlement` is not `Cash`.

The optional `IsDigital` node allows the creation of a strip of `CommodityDigitalOptions` (see 19.2.62). If set to `true` the node `PayoffPerUnit` needs to be set.

Node `PayoffPerUnit` [Optional] specifies the payoff per commodity unit, expressed in leg currency, in case a digital option is exercised. If the trade is a strip of digital options, this node must be set. It accepts real numbers as input.

19.2.66 Commodity Variance and Volatility Swap

A Commodity Variance or Volatility Swap has a payoff that depends on the volatility/variance of an underlying commodity instrument. See section 19.2.34 for the equivalent Equity product.

The `CommodityVarianceSwapData` node is the trade data container for the *CommodityVarianceSwap* trade type. The structure of an example `CommodityVarianceSwapData` node for a Commodity Variance Swap is the same as for an Equity Variance Swap in section 19.2.34, with the exception of the underlying node which is of type 'Commodity' here. See section 19.3.29 for additional optional elements of the underlying node and allowable values.

19.2.67 Commodity Position

An commodity position represents a position in a single commodity - using a single `Underlying` node, or in a weighted basket of underlying commodities - using multiple `Underlying` nodes.

An commodity Position can be used both as a stand alone trade type (TradeType: *CommodityPosition*) or as a trade component (`CommodityPositionData`) used within the *TotalReturnSwap* (Generic TRS) trade type, to set up for example Commodity Basket trades.

If the *PriceType* is set to *FutureSettlement* it will refer by default to today's prompt (lead) future. At the moment a generic TRS doesn't support rolling of the future contracts. Today's prompt future could be different from the prompt future at inception. If the initial price for the basket is not set, it will use the price of today's prompt future at trade inception as initial

Listing 134: Commodity option strip

```
<Trade id="...">
  <TradeType>CommodityOptionStrip</TradeType>
  <Envelope>
    ...
  </Envelope>
  <CommodityOptionStripData>
    <LegData>
      <LegType>CommodityFloating</LegType>
      ...
    </LegData>
    <Calls>
      <LongShorts>
        <LongShort>Short</LongShort>
      </LongShorts>
      <Strikes>
        <Strike>5.3</Strike>
      </Strikes>
      <BarrierData>
        <Type>UpAndIn</Type>
        <Style>American</Style>
        <LevelData>
          <Level>
            <Value>70.0</Value>
          </Level>
        </LevelData>
      </BarrierData>
    </Calls>
    <Puts>
      <LongShorts>
        <LongShort>Long</LongShort>
      </LongShorts>
      <Strikes>
        <Strike>8.17</Strike>
      </Strikes>
    </Puts>
    <Premiums> ... </Premiums>
    <Style>European</Style>
    <Settlement>Cash</Settlement>
  </CommodityOptionStripData>
</Trade>
```

price and the TRS will also ignore the roll yield caused by rolling from one prompt future to the next contract.

It is set up using an `CommodityPositionData` block as shown in listing 135. The meanings and allowable values of the elements in the block are as follows:

- Quantity: The number of shares or units of the weighted basket held.
  Allowable values: Any positive real number
- Underlying: One or more underlying descriptions. If a basket of commodities is defined, the `Weight` field should be populated for each underlyings. The weighted basket price is then given by

$$\text{Basket-Price} = \text{Quantity} \times \sum_i \text{Weight}_i \times S_i \times \text{FX}_i$$

where

- S_i is the i-th commodity prompt future or spot price in the basket
- FX_i is the FX Spot converting from the ith commodity currency to the first commodity currency which is by definition the currency in which the npv of the basket is expressed.

Allowable values: See 19.3.29 for the definition of an underlying. Only commodity underlyings are allowed.

Listing 135: Commodity position data

```
<Trade id="CommodityPosition">
  <TradeType>CommodityPosition</TradeType>
  <Envelope>...</Envelope>
  <CommodityPositionData>
    <Quantity>1000</Quantity>
    <Underlying>
      <Type>Commodity</Type>
      <Name>NYMEX:CL</Name>
      <Weight>0.5</Weight>
      <PriceType>FutureSettlement</PriceType>
      <FutureMonthOffset>0</FutureMonthOffset>
      <DeliveryRollDays>0</DeliveryRollDays>
      <DeliveryRollCalendar>TARGET</DeliveryRollCalendar>
    </Underlying>
    <Underlying>
      <Type>Commodity</Type>
      <Name>ICE:B</Name>
      <Weight>0.5</Weight>
      <PriceType>FutureSettlement</PriceType>
      <FutureMonthOffset>0</FutureMonthOffset>
      <DeliveryRollDays>0</DeliveryRollDays>
      <DeliveryRollCalendar>TARGET</DeliveryRollCalendar>
    </Underlying>
  </CommodityPositionData>
</Trade>
```

19.2.68 Generic Total Return Swap / Contract for Difference (CFD)

A generic total return swap / CFD (Trade type: *TotalReturnSwap* or *ContractForDifference*) is set up using a TotalReturnSwapData (or ContractForDifferenceData) block as shown in listing 138 and 144. Both trade types behave exactly the same.

Usually CFDs are traded without a funding component and captured with only two dates in the return schedule, namely the start date on which the initial price is fixed and a fictitious closing date usually set to "tomorrow" or another suitable future date. See listing 144 for the setup of a CFD on STOXX50E with initial price 3399.20 on 2019-09-28.

The generic total return swap is priced using the *accrual method* as opposed to a *full discounting method* as it is used for the *equity swap* trade type. The accrual method is

common practice when daily unwind rights are present in the trade terms or when the underlying valuation is too complex to allow for future projection.

The TotalReturnSwapData (ContractForDifferenceData) block is comprised of four sub-blocks, which are

- `UnderlyingData` containing one or more `Trade` subnodes describing the asset position of the TRS
- `ReturnData` describing the fixing and payment schedule of the return leg and specifying indices for FX conversion if applicable
- `FundingData` (optional) containing one or more funding legs of the TRS, whose notionals are based on either
  - "PeriodReset": the underlying price on the last valuation date before or on the accrual start date of the relevant funding coupon, this price is converted to the funding currency using the FX rate on this same valuation date for compo / cross currency swaps (see below)
  - "DailyReset": the underlying price on each day of the accrual period, again converted to the funding currency using the FX rate of the same date for compo / cross currency swaps. This notional type is only supported for fixed rate funding legs.
  - "Fixed": a fixed notional given explicitly in the funding leg
- `AdditionalCashflowData` (optional) a single leg of type Cashflow containing additional payments

The `ReturnData` and `FundingData` schedule periods often match, but this is not a strict requirement: In general, the funding notional is determined as described above dependent on the notional types "PeriodReset", "DailyReset", "Fixed".

Notice that in every case, the `UnderlyingData` schedule (if applicable to the underlying trade type as e.g. for a bond) is completely independent from the funding / return schedules: The underlying schedule defines the underlying flows to compute its NPV, and is not directly related to the return swap itself.

Generic TRS can be used to represent total return swaps on a wide range of underlying assets including e.g. single bonds or equities, CFDs on an underlying basket of EquityPositions, proprietary indices on equity options and equity or bond indices.

- The `UnderlyingData` block specifies one or more underlyings, which can be a trades of one of the following types (see the trade type specific sections), or structures with the Derivative or PortfolioIndexTradeData subnode.
  - Bond: See 19.2.40, the trade data is given in a BondData sub node for a single Bond.
  - ForwardBond: See 19.2.43, the trade data is given in a ForwardBondData sub node. To represent a TRS on a Bond Futures position a ForwardBond underlying is used.
  - CBO: See 19.2.50, the trade data is given in a CBOData sub node.

- CommodityPosition: See 19.2.67, the trade data is given in a CommodityPositionData sub node.
- ConvertibleBond: See 19.2.48, the trade data is given in a ConvertibleBondData sub node. When using reference data, a TRS on a convertible bond can also be captured as a TRS on a bond, i.e. there is no need to distinguish between a TRS on a Bond and a TRS on a convertible Bond in this case, the pricer will figure out which underlying to set up based on the type of reference data that is set up for the ISIN referenced in the security id field.
- EquityPosition: See 19.2.36, the trade data is given in a EquityPositionData sub node. Notice that the equities given in the basket must be available as quoted market data. To represent a TRS on an Equity Futures position, one or multiple EquityPosition underlyings are used, where the equity positions cover the underlying equity of the futures position.
- EquityOptionPosition: See 19.2.37, the trade data is given in a EquityOptionPositionData sub node.
- BondPosition: See 19.2.41, the trade data is given in a BondBasketData sub node for multiple underlying Bonds.
- CashPosition: See 19.2.7, the trade data is given in a CashPositionData sub node.
- Derivative: An arbitrary underlying derivative trade (of any type covered by VRE), allowing the set up of a so called Portfolio Swap with multiple underlying derivatives. The Derivative subnode has exactly two subnodes:
  - Id: A string with a unique identifier for the derivative position, typically starting with *DERIV:*.
  - Trade: The root node of a derivative trade.
- PortfolioIndexTradeData: This is a Portfolio Swap that references an underlying Basket via the `BasketName` identifier. The underlying basket can have an arbitrary number underlying derivatives of any supported TradeType. The PortfolioIndexTradeData subnode has one subnode:
  - BasketName: A string with a unique identifier for the portfolioIndex, matching the underlying reference basket.
  - IndexQuantity: Number of shares of the index.

Except for PortfolioIndexTradeData, each trade is specified by a `TradeType` and a trade type dependent data block as listed above. Listing 138 shows an example for a convertible bond underlying. Listing 139 shows an example for an equity basket underlying. Listing 140 shows an example for a bond basket underlying. Listing 141 shows an example for a Derivative underlying (with 3 underlying trades in this case). Listing 142 shows an example for a PortfolioIndexTradeData underlying.

- The `ReturnData` block specifies the details of the return leg.
  - Payer: Indicates whether the return leg is paid.

    Allowable values: *true, false*

– Currency: The currency in which the return is expressed. This can be different from the underlying currency ("composite" swap) and also from the funding leg currency ("cross currency" swap). The "composite" and "cross currency" features can occur alone or in combination.

  Allowable values: A valid currency code, see `Currency` in Table 20, provided it is the same as on the funding leg.

– ScheduleData: The reference schedule for the return leg, where the valuation dates are derived from this schedule using the ObservationLag, ObservationConvention and ObservationCalendar fields. The payment dates are derived from this schedule using the PaymentLag, PaymentConvention and PaymentCalendar fields. The payment dates can also be given as an explicit list in the PaymentDates node.

  Allowable values: A `ScheduleData` block as defined in section 19.3.4

– ObservationLag [Optional]: The lag between the valuation date and the reference schedule period start date.

  Allowable values: Any valid period, i.e. a non-negative whole number, followed by *D* (days), *W* (weeks), *M* (months), *Y* (years). Defaults to *0D* if left blank or omitted.

– ObservationConvention [Optional]: The roll convention to be used when applying the observation lag.

  Allowable values: A valid roll convention (*F, MF, P, MP, U, NEAREST*), see Table 21 Roll Convention. Defaults to *U* if left blank or omitted.

– ObservationCalendar [Optional]: The calendar to be used when applying the observation lag.

  Allowable values: Any valid calendar, see Table 24 Calendar. Defaults to the *NullCalendar* (no holidays) if left blank or omitted.

– PaymentLag [Optional]: The lag between the reference schedule period end date and the payment date.

  Allowable values: Any valid period, i.e. a non-negative whole number, optionally followed by *D* (days), *W* (weeks), *M* (months), *Y* (years). Defaults to *0D* if left blank or omitted. If a whole number is given and no letter, it is assumed that it is a number of *D* (days).

– PaymentConvention [Optional]: The business day convention to be used when applying the payment lag.

  Allowable values: A valid roll convention (*F, MF, P, MP, U, NEAREST*), see Table 21 Roll Convention. Defaults to *U* if left blank or omitted.

– PaymentCalendar [Optional]: The calendar to be used when applying the payment lag.

  Allowable values: Any valid calendar, see Table 24 Calendar. Defaults to the *NullCalendar* (no holidays) if left blank or omitted.

– PaymentDates [Optional]: This node allows for the specification of a list of explicit payment dates, using `PaymentDate` elements. The list must contain exactly $n - 1$ dates where n is the number of dates in the reference schedule given in the ScheduleData node. See Listing 136 for an example with an assumed ScheduleData with 4 dates.

Listing 136: Payment dates

```
<PaymentDates>
      <PaymentDate>2020-01-15</PaymentDate>
      <PaymentDate>2021-01-15</PaymentDate>
      <PaymentDate>2022-01-17</PaymentDate>
</PaymentDates>
```

– InitialPrice [Optional]: The equity (or bond) price of the underlying on the valuation date associated with the start date. Commonly contractually given. The price can be given in the underlying currency or the return currency as specified by the InitialPriceCurrency field and is given as

  * a (dirty) price for Bond, ForwardBond and Convertible Bond underlyings, the format is dependent on the price quotation method of the referenced bond:
    - Percentage of Par: the InitialPrice should be given as e.g. 1.02 for 102% relative dirty price
    - Currency per Unit: the InitialPrice should be given as e.g. 0.51 for a dirty amount of 51 USD per unit of the bond worth (say) 50.0 USD.
  * the weighted price of one unit of the bond underlying basket, notice that this is always a "percentage of par" price regardless of the quotation style of the single bonds in the basket
  * the (weighted) price of (one unit of) the equity underlying (basket)
  * the (weighted) price of (one unit of) the equity option underlying (basket)
  * an *absolute amount in the initial price ccy ("dollar amount")* if more than one underlying is specified and if a derivative is specified
  * absolute NPV if underlying is a CBO

  Notice that for an equity basket underlying with several currencies involved, the initial price is assumed to be given in the return currency in case no InitialPriceCurrency is given.

  Allowable values: A real number. If omitted or left blank it defaults to the equity (or bond) price of the valuation date associated with the start date. When this valuation date is in the future there is no fixed price, and in these cases the InitialPrice defaults to the forward price.

– InitialPriceCurrency [Optional]: Only relevant if InitialPrice is given. This specifies whether the initial price is given in the asset currency, the return currency or the funding currency.

Allowable values: One of the currencies in ReturnData / Currency (return currency), FundingData/ LegData / currency (funding currency) or the currency of the underlying asset. Defaults to the return currency if omitted.

- FXTerms [Mandatory when underlying asset / return / additional cashflow / funding currencies differ]: If the underlying asset currency is different from the return currency, an FXIndex for the conversion underlying / return currency must be given. The same holds for the funding and additional cashflow currencies: Whenever one of these currencies are different from the underlying currency, an FXIndex for the conversion to the underlying currency must be given. If multiple currencies differ, multiple FXIndex elements must be given.
  - FXIndex: The fx index to use for the conversion, this must contain the funding / return / additional cashflow currency and the underlying asset currency (in the order defined in table 28, i.e. it does not matter which one is the funding / return / additional cashflow currency and which is the underlying currency)

    Allowable values: see 28

  Notice that for an underlying of type EquityPosition or EquityOptionPosition additional `FXIndex` entries are required if there is more than one equity position in a different currency: Eventually, for each equity currency there must be a `FXIndex` specifying the conversion from the equity currency to the funding currency (or for the return/cashflow vs funding currency conversion). In this case multiple `FXIndex` entries are used within a single `FXTerms` node, see 137.

Listing 137: FXTerms with multiple FXIndex

```
<FXTerms>
      <FXIndex>FX-TR20H-GBP-SEK</FXIndex>
      <FXIndex>FX-TR20H-GBP-EUR</FXIndex>
      <FXIndex>FX-TR20H-GBP-USD</FXIndex>
</FXTerms>
```

- PayUnderlyingCashFlowsImmediately [Optional]: If true, underlying cashflows like coupon or amortisation payments from bonds or dividend payments from equities, are paid when they occur. If false, these cashflows are paid together with the next return payment. If omitted, the default value is false for trade type TotalReturnSwap and true for trade type ContractForDifference.

  Allowable values: true (immediate payment of underlying cashflwos) or false (underlying cashflows are paid on the next return payment date)

- The `FundingData` block specifies the details of the funding leg(s). The block is optional and can be omitted if no funding legs are present in the swap (e.g. for CFDs). It contains one or more LegData nodes, see 19.3.3. Allowed leg types are
  - Fixed
  - Floating
  - CMS

  - CMB

  The number of coupons defined by the legs often match the number of periods of the return schedule, but this is not a strict requirement. All funding legs must share the same payment currency.

  There are several ways to determine the notional of each funding leg, which is determined by additional, optional NotionalType tags. If given, there must be exactly one NotionalType tag for each LegData nodes. The types have the following meanings:

  - "PeriodReset": the notional of a funding period is determined by the underlying price on the last valuation date before or on the accrual start date of the relevant funding coupon, this price is converted to the funding currency using the FX rate on this same valuation date for compo / cross currency swaps.
  - "DailyReset": the notional of a funding period is determined by the underlying price on each day of the accrual period, again converted to the funding currency using the FX rate of the same date for compo / cross currency swaps. This notional type is only supported for fixed rate funding legs.
  - "Fixed": The notional is explicitly given in the leg data.

  If the NotionalType tags are not given, they default to "PeriodReset" in case no explicit notional is given on the leg and "Fixed" in case an explicit notional is given on the leg. See listing 138 for and example with two funding legs, one with a notional of type DailyReset and one with a notional of type PeriodReset.

  If a FundingResetGracePeriod is given, a lag of the given number of calendar days is applied when determining the relevant return valuation date that determines the funding notional. For example if FundingResetGracePeriod is set to 2, a valuation date that lies at most 2 calendar days after the funding accrual start date will be still considered eligible for this period.

- The `AdditionalCashflowData` block is optional and specifies unpaid amounts to be included in the NPV. The type of this leg must be Cashflow. The currency of the leg must be either the asset currency or the funding currency or the return currency.

Listing 138: Generic Total Return Swap with Convertible Bond underlying

```
<TotalReturnSwapData>
  <UnderlyingData>
    <Trade>
      <TradeType>Bond</TradeType>
      <BondData>
        <SecurityId>ISIN:XY1000000000</SecurityId>
        <BondNotional>1000000.00</BondNotional>
      </BondData>
    </Trade>
  </UnderlyingData>
  <ReturnData>
    <Payer>false</Payer>
    <Currency>EUR</Currency>
    <ScheduleData>...</ScheduleData>
    <ObservationLag>0D</ObservationLag>
    <ObservationConvention>P</ObservationConvention>
    <ObservationCalendar>USD</ObservationCalendar>
    <PaymentLag>2D</PaymentLag>
    <PaymentConvention>F</PaymentConvention>
    <PaymentCalendar>TARGET</PaymentCalendar>
    <!-- <PaymentDates> -->
    <!--   <PaymentDate> ... </PaymentDate> -->
    <!--   <PaymentDate> ... </PaymentDate> -->
    <!-- </PaymentDates> -->
    <InitialPrice>1.05</InitialPrice>
    <InitialPriceCurrency>EUR</InitialPriceCurrency>
    <FXTerms>
      <FXIndex>FX-ECB-EUR-USD</FXIndex>
      <FXIndex>FX-ECB-GBP-USD</FXIndex>
    </FXTerms>
    <PayUnderlyingCashFlowsImmediately>false</PayUnderlyingCashFlowsImmediately>
  </ReturnData>
  <FundingData>
    <FundingResetGracePeriod>2</FundingResetGracePeriod>
    <NotionalType>DailyReset</NotionalType>
    <LegData>
      <Payer>true</Payer>
      <LegType>Fixed</LegType>
      ...
    </LegData>
    <NotionalType>PeriodReset</NotionalType>
    <LegData>
      <Payer>true</Payer>
      <LegType>Floating</LegType>
      ...
    </LegData>
  </FundingData>
  <AdditionalCashflowData>
    <LegData>
      <Payer>false</Payer>
      <LegType>Cashflow</LegType>
      ...
    </LegData>
  </AdditionalCashflowData>
</TotalReturnSwapData>
```

Listing 139: Generic Total Return Swap with equity basket underlying

```
    <TotalReturnSwapData>
      <UnderlyingData>
        <Trade>
          <TradeType>EquityPosition</TradeType>
          <EquityPositionData>
            <!-- basket price = quantity x sum_i ( weight_i x equityPrice_i x fx_i ) -->
            <Quantity>1000</Quantity>
            <Underlying>
              <Type>Equity</Type>
              <Name>BE0003565737</Name>
              <Weight>0.5</Weight>
              <IdentifierType>ISIN</IdentifierType>
              <Currency>EUR</Currency>
              <Exchange>XFRA</Exchange>
            </Underlying>
            <Underlying>
              <Type>Equity</Type>
              <Name>GB00BH4HKS39</Name>
              <Weight>0.5</Weight>
              <IdentifierType>ISIN</IdentifierType>
              <Currency>GBP</Currency>
              <Exchange>XLON</Exchange>
            </Underlying>
          </EquityPositionData>
        </Trade>
      </UnderlyingData>
      <ReturnData>
        ...
        <InitialPrice>112.0</InitialPrice>
        <InitialPriceCurrency>USD</InitialPriceCurrency>
        <FXTerms>
          <FXIndex>FX-ECB-EUR-USD</FXIndex>
          <FXIndex>FX-TR20H-GBP-USD</FXIndex>
        </FXTerms>
      </ReturnData>
      <FundingData>
        <LegData>
          <Payer>true</Payer>
          <LegType>Floating</LegType>
          <Currency>USD</Currency>
          ...
        </LegData>
      </FundingData>
      <AdditionalCashflowData>
        <LegData>
          <Payer>false</Payer>
          <LegType>Cashflow</LegType>
          ...
        </LegData>
      </AdditionalCashflowData>
    </TotalReturnSwapData>
  </Trade>
```

Listing 140: Generic Total Return Swap with bond basket underlying

```
    <TotalReturnSwapData>
      <UnderlyingData>
        <Trade>
          <TradeType>BondPosition</TradeType>
          <BondBasketData>
            <Quantity>100000000</Quantity>
            <Identifier>ISIN:GB00B4KT9Q30</Identifier>
          </BondBasketData>
        </Trade>
      </UnderlyingData>
      <!-- omitting ReturnData, FundingData, AdditionalCashflowData -->
    </TotalReturnSwapData>
  </Trade>
```

Listing 141: Generic Total Return Swap on a Derivative underlying

```
    <TotalReturnSwapData>
      <UnderlyingData>
        <Derivative>
          <Id>DERIV:TEST1</Id>
          <Trade>
            <TradeType>Swaption</TradeType>
              <SwaptionData> ... </SwaptionData>
          </Trade>
        </Derivative>
        <Derivative>
          <Id>DERIV:TEST2</Id>
          <Trade>
            <TradeType>Swap</TradeType>
              <SwapData> ... </SwapData>
          </Trade>
        </Derivative>
         <Derivative>
          <Id>DERIV:TEST3</Id>
          <Trade>
            <TradeType>EquityPosition</TradeType>
              <EquityPositionData> ... </EquityPositionData>
          </Trade>
        </Derivative>
      </UnderlyingData>
      <!-- omitting ReturnData, FundingData, AdditionalCashflowData -->
    </TotalReturnSwapData>
  </Trade>
```

Listing 142: Generic Total Return Swap on a PortfolioIndexTradeData underlying

```
    <TotalReturnSwapData>
      <UnderlyingData>
        <PortfolioIndexTradeData>
          <BasketName>MSFDSJP</BasketName>
        </PortfolioIndexTradeData>
      </UnderlyingData>
      <!-- omitting ReturnData, FundingData, AdditionalCashflowData -->
    </TotalReturnSwapData>
  </Trade>
```

Listing 143: Generic Total Return Swap on a commodity index underlying

```
    <TotalReturnSwapData>
      <UnderlyingData>
        <Trade>
          <TradeType>CommodityPosition</TradeType>
          <CommodityPositionData>
            <!-- basket price = quantity x sum_i ( weight_i x price_i x fx_i ) -->
            <Quantity>1000</Quantity>
            <Underlying>
              <Type>Commodity</Type>
              <Name>RIC:.BCOM</Name>
              <Weight>1.0</Weight>
              <PriceType>Spot</PriceType>
            </Underlying>
          </CommodityPositionData>
        </Trade>
      </UnderlyingData>
      <!-- omitting ReturnData, FundingData, AdditionalCashflowData -->
    </TotalReturnSwapData>
  </Trade>
```

Listing 144: CFD on STOXX50E with initial price 3399.20 EUR

```
<ContractForDifferenceData>
  <UnderlyingData>
    <Trade>
      <TradeType>EquityPosition</TradeType>
      <EquityPositionData>
        <Quantity>1000</Quantity>
        <Underlying>
          <Type>Equity</Type>
          <Name>.STOXX50E</Name>
          <Weight>1.0</Weight>
          <IdentifierType>RIC</IdentifierType>
        </Underlying>
      </EquityPositionData>
    </Trade>
  </UnderlyingData>
  <ReturnData>
    <Payer>false</Payer>
    <Currency>EUR</Currency>
    <ScheduleData>
      <Dates>
        <Dates>
          <!-- the start date of the CFD on which the initial price was set -->
          <Date>2018-09-28</Date>
          <!-- fictitious closing date, e.g. set to "tomorrow" -->
          <Date>2019-01-04</Date>
        </Dates>
      </Dates>
    </ScheduleData>
    <InitialPrice>3399.20</InitialPrice>
    <InitialPriceCurrency>EUR</InitialPriceCurrency>
  </ReturnData>
  </ContractForDifferenceData>
```

19.2.69 Equity Outperformance Option

An Equity Outperformance option has a payoff that depends on the 'outperformance' of two equity indices (i.e. the difference between their returns) against a strike return. The buyer has the right but not the obligation to receive the outperformance in exchange for the strike rate at a predetermined time in the future.

The trade may optionally have a knockIn or knockOut price (or both). Only if the price of Underlying2 is above the knockIn value or below the knockOut value is the payoff paid.

The `EquityOutperformanceOptionData` node is the trade data container for the *EquityOutperformanceOption* trade type. The `EquityOutperformanceOptionData` node includes one `OptionData` trade component sub-node plus elements specific to the Equity Outperformance Option.

The structure of an example `EquityOutperformanceOptionData` node for an Equity Outperformace Option is shown in Listing 145.

Listing 145: Equity Outperformance Option Data

```
<EquityOutperformanceOptionData>
    <OptionData>
      <LongShort>Long</LongShort>
      <OptionType>Call</OptionType>
      <Style>European</Style>
      <Settlement>Cash</Settlement>
      <ExerciseDates>
        <ExerciseDate>2022-09-21</ExerciseDate>
      </ExerciseDates>
      ...
    </OptionData>
  <Currency>USD</Currency>
  <Notional>500000</Notional>
  <Underlying1>
    <Type>Equity</Type>
    <Name>RIC:.SPX</Name>
  </Underlying1>
  <Underlying2>
    <Type>Equity</Type>
    <Name>RIC:.NDX</Name>
  </Underlying2>
  <InitialPrice1>2140</InitialPrice1>
  <InitialPrice2>13000</InitialPrice2>
  <StrikeReturn>0.01</StrikeReturn>
  <KnockInPrice>12500</KnockInPrice>
  <KnockOutPrice>14000</KnockOutPrice>
</EquityOutperformanceOptionData>
```

The Payoff is:

$$N \cdot \max(0, R_1 - R_2 - K)$$

where:

- N is the notional amount
- R_1 is the return of `Underlying1`
- R_2 is the return of `Underlying2`
- K is the `StrikeReturn`.

The meanings and allowable values of the elements in the `EquityOutperformaceOptionData` node follow below.

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the `OptionData` node for an EquityOutperformanceOption are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*. *Call* means that the holder has the right but not obligation to receive the Outperformance and pay the StrikeReturn. *Put* means that the holder has the right but not obligation to pay the Outperformance and receive the StrikeReturn.
  - `Style` The allowable value is *European*. Note that the Equity Outperformance Option type allows for *European* option style only.

  - `Settlement` The allowable values are *Cash* or *Physical*.
  - An `ExerciseDates` node where exactly one ExerciseDate date element must be given.
  - Premiums [Optional]: Option premium amounts paid by the option buyer to the option seller.

    Allowable values: See section 19.3.2

- Currency: The currency of the equity outperformance option.

  Allowable values: See Table 22 `Currency`.

- Underlying1: Specifies the first underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29. Note that the node name is `Underlying1`.

- Underlying2: Specifies the second underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29. Note that the node name is `Underlying2`.

  Also note that the equities in Underlying1 and Underlying2 must be quoted in the same currency.

- InitialPrice1: Specifies the initial price for first underlying equity.

  Allowable values: Any positive real number.

- InitialPrice2: Specifies the initial price for second underlying equity.

  Allowable values: Any positive real number.

- StrikeReturn: The option strike return.

  Allowable values: Any positive real number.

- Notional: The notional amount for the trade.

  Allowable values: Any positive real number.

- KnockInPrice[Optional]: The payoff is only paid if on the settlement date the price of underlying2 is above this value.

  Allowable values: Any positive real number.

- KnockOutPrice[Optional]: The payoff is only paid if on the settlement date the price of underlying2 is below this value.

  Allowable values: Any positive real number.

- InitialPriceCurrency1 [Optional]: Only relevant if InitialPrice1 is given in a currency other than Underlying1's currency.

  Allowable values: See Table 22 `Currency`.

- InitialPriceCurrency2 [Optional]: Only relevant if InitialPrice1 is given in a currency other than Underlying2's currency.

  Allowable values: See Table 22 `Currency`.

- InitialPriceFXTerms1 [Mandatory when InitialPriceCurrency1 is provided]: The node must be given if and only if the underlying currency is different from the initialPrice currency. The node contains the following sub nodes:
  - FXIndex: The fx index to use for the conversion, this must contain the underlying asset currency and the funding leg currency (in the order defined in table 28, i.e. it does not matter which one is the asset currency and which is the funding currency)

    Allowable values: see 28
  - InitialPriceFXTerms2 [Mandatory when InitialPriceCurrency2 is provided]: The node must be given if and only if the underlying currency is different from the initialPrice currency. Contains the same subnodes as InitialPriceFXTerms1.

    Allowable values: Any valid calendar, see Table 24 Calendar. Defaults to the *NullCalendar* (no holidays) if left blank or omitted.

19.2.70 Double Digital Option

The `DoubleDigitalOptionData` node is the trade data container for the DoubleDigitalOption trade type, listing 146 shows the structure of an example with two underlying FX rates. Equity, Commodity and IR underlyings are also supported in arbitrary combinations. A double digital option is a binary option that pays out a fixed amount if the two underlyings (FX spots, Equity or Commodity prices, interest rates) are simultaneously in the money w.r.t. given strikes and option types at the option expiry.

Listing 146: Double Digital Option data

```
<DoubleDigitalOptionData>
  <Expiry>2021-09-01</Expiry>
  <Settlement>2021-09-03</Settlement>
  <BinaryPayout>12000000</BinaryPayout>
  <BinaryLevel1>1.1</BinaryLevel1>
  <BinaryLevel2>0.006</BinaryLevel2>
  <BinaryLevelUpper2>0.008</BinaryLevelUpper2>
  <Type1>Call</Type1>
  <Type2>Collar</Type2>
  <Position>Long</Position>
  <Underlying1>
    <Type>FX</Type>
    <Name>ECB-EUR-USD</Name>
  </Underlying1>
  <Underlying2>
    <Type>FX</Type>
    <Name>ECB-JPY-USD</Name>
  </Underlying2>
  <PayCcy>USD</PayCcy>
</DoubleDigitalOptionData>
```

The meanings and allowable values of the elements in the `DoubleDigitalOptionData` node follow below.

- Expiry: The expiry date of the option. Allowable values are valid dates.
- Settlement: The payout settlement date. Allowable values are valid dates.

- BinaryPayout: The amount that is paid if the option is in the money. Allowable values are all non-negative numbers.
- BinaryLevel1: The strike for underlying 1 for *Call* or *Put* option and the lower bound for a *Collar* option. For an FX underlying (SOURCE-CCY1-CCY2) this is the number of units of CCY2 per units of CCY1. For an Equity underlying this is the equity price expressed in the equity ccy. For a Commodity underlying this is the commodity price expressed in the commodity ccy. For an IR underlying this is the rate expressed in decimal form. Allowable values are non-negative numbers.
- BinaryLevel2: The strike for underlying 2 for *Call* or *Put* option and the lower bound for a *Collar*. For an FX underlying (SOURCE-CCY1-CCY2) this is the number of units of CCY2 per units of CCY1. For an Equity underlying this is the equity price expressed in the equity ccy. For a Commodity underlying this is the commodity price expressed in the commodity ccy. For an IR underlying this is the rate expressed in decimal form. Allowable values are non-negative numbers.
- Type1: The option type that applies to underlying 1. Allowable values: *Call*, *Put* or *Collar*. Underlying 1 is considered to be in the money if the spot is above (Call) / below (Put) the BinaryLevel1 resp. between (Collar) the BinaryLevel1 and BinaryLevelUpper1 at the expiry.
- Type2: The option type that applies to underlying 2. Allowable values: *Call*, *Put* or *Collar*. Underlying 2 is considered to be in the money if the spot is above (Call) / below (Put) the BinaryLevel1 resp. between (Collar) the BinaryLevel2 andthe BinaryLevelUpper2 at the expiry.
- Position: The option position type. Allowable values: *Long* or *Short*.
- Underlying1: The first underlying, see 19.3.29.
- Underlying2: The second underlying, see 19.3.29. Note that Type for both underlyings has allowable values *Equity*, *Commodity*, *FX*, and *IR*.
- Underlying3 [Optional]: If defined, the first underlying in this transaction is treated as a spread between Underlying1 and Underlying3 (i.e. Underlying1 fixing minus Underlying3 fixing), see 19.3.29. Underlying3 Type must be the same as Underlying1 Type.
- Underlying4 [Optional]: If defined, the second underlying in this transaction is treated as a spread between Underlying2 and Underlying4 (i.e. Underlying2 fixing minus Underlying4 fixing), see 19.3.29. Underlying4 Type must be the same as Underlying2 Type.
- PayCcy: The currency in which the `BinaryPayout` is paid. See Table 22 for allowable currency codes.
- BinaryLevelUpper1 [Optional]: This is field is used only for Collar option. The upper bound for underlying 1. For an FX underlying (SOURCE-CCY1-CCY2) this is the number of units of CCY2 per units of CCY1. For an Equity underlying this is the equity price expressed in the equity ccy. For a Commodity underlying this is the commodity price expressed in the commodity ccy. For an IR underlying this is the rate expressed in decimal form. Allowable values are non-negative numbers.

- BinaryLevelUpper2 [Optional]: This field is used only for Collar option. The upper bound for underlying 2. For an FX underlying (SOURCE-CCY1-CCY2) this is the number of units of CCY2 per units of CCY1. For an Equity underlying this is the equity price expressed in the equity ccy. For a Commodity underlying this is the commodity price expressed in the commodity ccy. For an IR underlying this is the rate expressed in decimal form. Allowable values are non-negative numbers.

19.2.71 European Option Contingent on a Barrier

European exercise, American or European barrier, multi-asset

This is a plain vanilla European option on a given underlying, whose final payoff is contingent on the price level of another underlying (forming the barrier). Both underlyings can be of multiple asset classes.

The trade container for this product is the `EuropeanOptionBarrierData` node, and the corresponding trade type is EuropeanOptionBarrier. The barrier can be continuously monitored (American) or only be monitored on the option expiry date (European). Currently, we support Equity, FX, Commodity and InterestRate for the option underlying. For the barrier underlying, we support Equity, FX, Commodity and InterestRate for European-style barriers, but not InterestRate for an American-style barrier.

Listing 147 shows the structure of an Equity European option with an American-style FX barrier. For a European-style barrier, the `BarrierType` must be set to *European* and the `BarrierSchedule` can be omitted.

Listing 147: European Option Barrier data (continuous barrier)

```
<Trade id="Equity_EuropeanOptionWithAmericanFxBarrier">
  <TradeType>EuropeanOptionBarrier</TradeType>
  <Envelope>
    .....
  </Envelope>
  <EuropeanOptionBarrierData>
    <Quantity>8523</Quantity>
    <PutCall>Call</PutCall>
    <LongShort>Short</LongShort>
    <Strike>3520</Strike>
    <PremiumAmount>114.40</PremiumAmount>
    <PremiumCurrency>EUR</PremiumCurrency>
    <PremiumDate>2019-12-13</PremiumDate>
    <OptionExpiry>2020-06-19</OptionExpiry>
    <OptionUnderlying>
      <Type>Equity</Type>
      <Name>RIC:.STOXX50E</Name>
    </OptionUnderlying>
    <BarrierUnderlying>
      <Type>FX</Type>
      <Name>ECB-EUR-USD</Name>
    </BarrierUnderlying>
    <BarrierLevel>1.09335</BarrierLevel>
    <BarrierType>DownAndIn</BarrierType>
    <BarrierStyle>American</BarrierStyle>
    <BarrierSchedule>
      <Rules>
        <StartDate>2019-12-11</StartDate>
        <EndDate>2020-06-19</EndDate>
        <Tenor>1D</Tenor>
        <Calendar>USA</Calendar>
        <Convention>Following</Convention>
        <TermConvention>Following</TermConvention>
        <Rule>Forward</Rule>
      </Rules>
    </BarrierSchedule>
    <SettlementDate>2020-06-24</SettlementDate>
    <PayCcy>USD</PayCcy>
  </EuropeanOptionBarrierData>
</Trade>
```

The meanings and allowable values of the elements in the `EuropeanOptionBarrierData` node follow below.

- Quantity: Number of option contracts.
  Allowable values: Any non-negative number.
- PutCall: Option type.
  Allowable values: *Call, Put*
- LongShort: Own party position.
  Allowable values: *Long, Short*
- Strike: Option strike price.
  Allowable values: Any positive number.

- PremiumAmount: Premium amount per option.
  Allowable values: Any number.
- PremiumCurrency: Currency of the option premium.
  Allowable values: See `Currency` in Table 20.
- PremiumDate: The option premium payment date.
  Allowable values: See `Date` in Table 20.
- OptionExpiry: Option expiry date.
  Allowable values: See `Date` in Table 20.
- OptionUnderlying: The option underlying.
  Allowable values: A node of the same form as `Underlying`, (see 19.3.29). The supported types are *Equity*, *FX*, *Commodity* and *InterestRate*.
- BarrierUnderlying: The underlying monitored against the barrier level.
  Allowable values: A node of the same form as `Underlying`, (see 19.3.29). The supported types are *Equity*, *FX*, *Commodity* and *InterestRate* if `BarrierStyle` is *European*. For `BarrierStyle` *American*, we only support *Equity*, *FX* and *Commodity*.
- BarrierLevel: Knock-in/knock-out barrier level.
  Allowable values: Any number.
- BarrierType: The type of knock-in or knock-out barrier.
  Allowable values: *DownIn, UpIn, DownOut, UpOut*
- BarrierStyle: Whether the barrier is continuously monitored or only at the option expiry date.
  Allowable values: *American, European*
- BarrierSchedule [Optional]: The schedule specifying the schedule of trading days over which the continuous barrier will be monitored. This is required only when `BarrierStyle` is *American*.
  Allowable values: A node of the same form as `ScheduleData`, (see 19.3.4).
- SettlementDate: Settlement date of the option exercise payoff.
  Allowable values: See `Date` in Table 20. The Settlement date must be on or after the Option expiry date.
- PayCcy: Settlement currency.
  Allowable values: See Table 22 `Currency`.

19.2.72 Autocallable Type 01

The `Autocallable_01` node is the trade data container for the Autocallable_01 trade type, listing 148 shows the structure of an example.

Listing 148: Autocallable Type 01 data

```
<Autocallable01Data>
  <NotionalAmount>12000000</NotionalAmount>
  <DeterminationLevel>11.0</DeterminationLevel>
  <TriggerLevel>9.8</TriggerLevel>
  <Underlying>
    <Type>FX</Type>
    <Name>ECB-EUR-NOK</Name>
  </Underlying>
  <Position>Long</Position>
  <PayCcy>EUR</PayCcy>
  <FixingDates>
    <ScheduleData>
      <Dates>
        <Dates>
          <Date>2018-09-27</Date>
          <Date>2019-09-27</Date>
          <Date>2020-09-27</Date>
          <Date>2021-09-29</Date>
          <Date>2022-09-28</Date>
        </Dates>
      </Dates>
    </ScheduleData>
  </FixingDates>
  <SettlementDates>
    <ScheduleData>
      <Dates>
        <Dates>
          <Date>2018-10-07</Date>
          <Date>2019-10-09</Date>
          <Date>2020-10-07</Date>
          <Date>2021-10-07</Date>
          <Date>2022-10-07</Date>
        </Dates>
      </Dates>
    </ScheduleData>
  </SettlementDates>
  <AccumulationFactors>
    <Factor>0.344</Factor>
    <Factor>0.733</Factor>
    <Factor>0.911</Factor>
    <Factor>1.123</Factor>
    <Factor>1.544</Factor>
  </AccumulationFactors>
  <Cap>1.0</Cap>
</Autocallable01Data>
```

If a trigger event occurs on the i-th fixing date, the option holder receives the following:

$$Payout = \texttt{NotionalAmount} * AccumulationFactor_i.$$

If a trigger event never occurs and the underlying spot at the fixing date f_n is above the `DeterminationLevel`, the option holder pays the following:

$$Payout = \min(\texttt{Cap}, \texttt{Underlying}(f_n) - \texttt{DeterminationLevel}).$$

The meanings and allowable values of the elements in the `Autocallable01Data` node follow below.

- NotionalAmount: The notional amount of the option. Allowable values are non-negative numbers.
- DeterminationLevel: The determination level. For an FX underlying FX-SOURCE-CCY1-CCY2 this is the number of units of CCY2 per units of CCY1. For an EQ underlying this is the equity price expressed in the equity ccy. Allowable values are non-negative numbers.
- TriggerLevel: The trigger level. For an FX underlying FX-SOURCE-CCY1-CCY2 this is the number of units of CCY2 per units of CCY1. For an EQ underlying this is the equity price expressed in the equity ccy. Allowable values are non-negative numbers.
- Underlying: The option underlying, see 19.3.29.
- Position: The option position type. Allowable values: *Long* or *Short*.
- PayCcy: The pay currency of the option. See the appendix for allowable currency codes.
- FixingDates: The fixing date schedule given as a `ScheduleData` subnode, see 19.3.4
- SettlementDates: The settlement date schedule given as a `ScheduleData` subnode, see 19.3.4
- AccumulationFactors: The accumulation factors given as a list of values corresponding to the fixing dates. Allowable values are non-negative numbers.
- Cap: The maximum amount, per unit of notional, payable by the option holder if a trigger event never occurred (i.e. underlying value was never below the `TriggerLevel` on any of the fixing dates) and if the underlying value is greater than the `DeterminationLevel` at the last fixing date. Allowable values are non-negative numbers.

19.2.73 Performance Option Type 01

Payoff

The performace option of type "01" is characterized by the following data

- a notional amount N
- a participation rate q
- a valuation date V and a settlement date S
- a number of underlyings U_i for $i = 1, \ldots, n$
- weights for the underlyings w_i for $i = 1, \ldots, n$
- initial strike prices for the underlyings s_i for $i = 1, \ldots, n$
- an option strike K

On the valuation date the average performance of the underlying basket is computed as

$$P = \max\left(\sum_{i=1}^{n} w_i \left(\frac{U_i(V)}{s_i} - K\right), 0\right)$$

The option holder receives an amount $N \cdot q \cdot P$ on the settlement date S. The underlyings can be Equity, FX or Commodity underlyings.

The above payoff includes the strike in the performance calculation. There is another variant with excluded strike and payoff:

$$P = \max\left(\left[\sum_{i=1}^{n} w_i \frac{U_i(V)}{s_i}\right] - K, 0\right)$$

Input

The `PerformanceOption_01` node is the trade data container for the PerformanceOption_01 trade type, listing 149 shows the structure of an example.

Listing 149: Performance Option Type 01 data

```
<PerformanceOption01Data>
  <NotionalAmount>12500000</NotionalAmount>
  <ParticipationRate>0.9</ParticipationRate>
  <ValuationDate>2022-05-03</ValuationDate>
  <SettlementDate>2022-05-05</SettlementDate>
  <Underlyings>
    <Underlying>
      <Type>FX</Type>
      <Name>ECB-CHF-EUR</Name>
      <Weight>0.34</Weight>
    </Underlying>
    <Underlying>
      <Type>FX</Type>
      <Name>ECB-NOK-EUR</Name>
      <Weight>0.32</Weight>
    </Underlying>
    <Underlying>
      <Type>FX</Type>
      <Name>ECB-SEK-EUR</Name>
      <Weight>0.24</Weight>
    </Underlying>
    <Underlying>
      <Type>FX</Type>
      <Name>ECB-SEK-EUR</Name>
      <Weight>0.10</Weight>
    </Underlying>
  </Underlyings>
  <StrikePrices>
    <StrikePrice>0.910002</StrikePrice>
    <StrikePrice>0.097192</StrikePrice>
    <StrikePrice>0.096085</StrikePrice>
    <StrikePrice>0.035032</StrikePrice>
  </StrikePrices>
  <Strike>1.15</Strike>
  <StrikeIncluded>true</StrikeIncluded>
  <Position>Long</Position>
  <PayCcy>EUR</PayCcy>
</PerformanceOption01Data>
```

The meanings and allowable values of the elements in the `PerformanceOption01Data` node follow below.

- NotionalAmount: The notional amount of the option. Allowable valus are non-negative numbers.
- ParticipationRate: The participation rate. Allowable values are non-negative numbers. Usually the value will be between 0 and 1.
- ValuationDate: The valuation date. Allowable values are valid dates.
- SettlementDate: The settlement date. Allowable values are valid dates.
- Underlyings: The underlyings of the option. See 19.3.29 for each underlying.
- StrikePrices: The initial strike prices of the underlyings. For an FX underlying

FX-SOURCE-CCY1-CCY2 this is the number of units of CCY2 per units of CCY1. For an EQ underlying this is the equity price expressed in the equity ccy. For a Commodity underlying this is the commodity price quoted as per the underlying commodity. Allowable values are non-negative numbers.

- Strike: The option strike. This is expressed in terms of the performance of the underlying basket (see the product description for more details). Allowable values are numbers.
- StrikeIncluded [optional]: If true the strike is included in the performance calculation, this is also the default if the flag not given. If false the strike is excluded.
- Position: The option position. Allowable values are *Long* or *Short*.
- PayCcy: The payment currency of the option. See the appendix for allowable currency codes.

19.2.74 Window Barrier Option

The `FxWindowBarrierOptionData`, `EquityWindowBarrierOptionData`, `CommodityWindowBarrierOptionData` is the trade data container for the FxWindowBarrierOption, EquityWindowBarrierOption, CommodityBarrierOption trade type. The following listing shows the structure of an example trade for an FX underlying.

```
<Trade id="FX_Window_BarrierOption">
  <TradeType>FxWindowBarrierOption</TradeType>
  <Envelope>
   ...
  </Envelope>
  <FxWindowBarrierOptionData>
    <Currency>USD</Currency>
    <FixingAmount>1000000</FixingAmount>
    <Underlying>
      <Type>FX</Type>
      <Name>ECB-EUR_USD</Name>
    </Underlying>
    <Strike>1.1</Strike>
    <StartDate>2023-03-01</StartDate>
    <EndDate>024-03-01</EndDate>
    <OptionData>
      <LongShort>Long</LongShort>
      <OptionType>Call</OptionType>
      <ExerciseDates>
        <ExerciseDate>2026-03-01</ExerciseDate>
      </ExerciseDates>
      <Premiums>
        <Premium>
          <Amount>10900</Amount>
          <Currency>EUR</Currency>
          <PayDate>2018-03-01</PayDate>
        </Premium>
      </Premiums>
      <PaymentData>
        <Dates>
          <Date>2026-03-01</Date>
        </Dates>
```

```
      </PaymentData>
    </OptionData>
    <BarrierData>
      <Type>UpAndOut</Type>
      <Levels>
        <Level>1.3</Level>
      </Levels>
    </BarrierData>
  </FxWindowBarrierOptionData>
</Trade>
```

The meanings and allowable values of the elements in the data node follow below.

- Currency: The payout currency. The result of the payout formula above is treated to be in this currency. Note that for (non-quanto) FxWindowBarrierOption this should be the domestic (`CCY2`) currency. For non-quanto Equity- and CommodityWindowBarrierOptions this should be the currency the equity or commodity is quoted in. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 `Currency`.
- FixingAmount: The notional amount. For FxWindowBarrierOptions: The FixingAmount is expressed in the foreign currency (`CCY1`). For EquityWindowBarrierOptions: The FixingAmount is expressed as number of shares/units of the underlying equity or equity index. For CommodityWindowBarrierOptions: The FixingAmount is expressed as number of units of the underlying commodity.
  Allowable values: Any positive real number.
- Underlying: A node with the underlying of the Window Barrier instrument. For FxWindowBarrierOption: `Type` is set to *FX* and `Name` is a string of the form *SOURCE-CCY1-CCY2* where `CCY1` is the foreign currency, `CCY2` is the domestic currency, and *SOURCE* is the fixing source, see Table 28 and 19.3.29. For EquityWindowBarrierOption: `Type` is set to *Equity* and `Name` and other fields are as outlined in 19.3.29. For CommodityWindowBarrierOption: `Type` is set to *Commodity* and `Name` is an identifier of the commodity as outlined in 19.3.29 and in Table 32.
  Allowable values: Any FX, Equity or Commodity underlying as specified in 19.3.29
- StrikeData [Optional]: A node containing the strike in `Value` and the currency in which both the underlying and the strike are quoted in `Currency`. Only supported for EquityWindowBarrierOption.
  Allowable values: See `Currency` in Table 20. The strike may be any positive real number. The currency provided in this node may be quoted as corresponding minor currency to the underlying major currency.
- Strike [Optional]: For FxWindowBarrierOptions: The the fx strike rate is defined as amount in domestic currency (`CCY2`) for one unit of foreign currency (`CCY1`). For Equity- and CommodityWindowBarrierOptions: The strike value for one unit/share/contract of the underlying equity or commodity, expressed in the currency the equity or commodity is quoted in. For EquityWindowBarrierOptions, the `StrikeData` node (see above) should be used.
  Allowable values: Any positive real number.

- StartDate: The start date of the continuous observation window.
  Allowable values: Any valid date.
- EndDate: The end date of the continuous observation window.
  Allowable values: Any valid date.
- OptionData: See 19.3.1. The relevant fields in the `OptionData` node for a Window Barrier Option are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `OptionType` The allowable values are *Call* or *Put*.
  - `ExerciseDates` Must contain the single exercise date of the option.
  - `Premiums` [Optional] Option premium amounts paid by the option buyer to the option seller. See section 19.3.2
  - `PaymentData` [Optional] Optional pay date associated to the exercise date.
- BarrierData: Specification of the barrier type and level. The barrier is continuosly monitored between the StartDate and EndDate.
  Allowable values: See 19.3.31, allowable types are *UpAndOut*, *DownAndOut*,*UpAndIn*, *DownAndIn*. Exactly one level must be given as a positive real number.

Window Barrier Options can be alternatively represented as *scripted trades*, refer to vre/docs/scripted_trade for an introduction.

Trade input and the payoff script are described below.

```
<Trade id="WindowBarrierOption#1">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope/>
  <WindowBarrierOptionData>
    <PremiumPayDate type="event">2018-03-01</PremiumPayDate>
    <Settlement type="event">2026-03-01</Settlement>
    <Expiry type="event">2026-03-01</Expiry>
    <StartDate type="event">2023-03-01</StartDate>
    <EndDate type="event">2024-03-01</EndDate>
    <PremiumAmount type="number">10900</PremiumAmount>
    <BarrierLevel type="number">1.3</BarrierLevel>
    <PutCall type="optionType">Call</PutCall>
    <BarrierType type="barrierType">UpOut</BarrierType>
    <LongShort type="longShort">Long</LongShort>
    <Quantity type="number">1000000</Quantity>
    <Strike type="number">1.1</Strike>
    <PayCcy type="currency">USD</PayCcy>
    <PremiumCcy type="currency">EUR</PremiumCcy>
    <Underlying type="index">FX-ECB-EUR-USD</Underlying>
  </WindowBarrierOptionData>
</Trade>
```

The WindowBarrierOption script referenced in the trade above is shown in Listing 150.

The meanings and allowable values of the elements in the `WindowBarrierOptionData` node below.

- [event] `Expiry`: Option expiry date.

Listing 150: Payoff script for a WindowBarrierOption.

```
REQUIRE BarrierType == 1 OR BarrierType == 2 OR BarrierType == 3 OR BarrierType == 4;

NUMBER i, Payoff, TriggerProbability, ExerciseProbability, isUp, currentNotional;

IF BarrierType == 1 OR BarrierType == 3 THEN
  TriggerProbability = BELOWPROB(Underlying, StartDate, EndDate, BarrierLevel);
ELSE
  TriggerProbability = ABOVEPROB(Underlying, StartDate, EndDate, BarrierLevel);
END;

Payoff = Quantity * PutCall * (Underlying(Expiry) - Strike);
IF Payoff > 0.0 THEN
  IF BarrierType == 1 OR BarrierType == 2 THEN
    Option = PAY(Payoff * TriggerProbability, Expiry, Settlement, PayCcy);
    ExerciseProbability = TriggerProbability;
  ELSE
    Option = PAY(Payoff * (1 - TriggerProbability), Expiry, Settlement, PayCcy);
    ExerciseProbability = (1 - TriggerProbability);
  END;
END;

Option = LongShort * (Option - PAY(PremiumAmount, PremiumPayDate, PremiumPayDate, PremiumCcy));
currentNotional = Quantity * Strike;
```

Allowable values: See `Date` in Table 20.

- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.
- [event] `StartDate`: Barrier monitoring start date.
  Allowable values: See `Date` in Table 20.
- [event] `EndDate`: Barrier monitoring start date.
  Allowable values: See `Date` in Table 20.
- [number] `Strike`: The option strike price. For FX, this should be the amount of `CCY1` divided by amount of `CCY2` as defined in *Index*.
  Allowable values: Any positive real number.
- [number] `BarrierLevel`: Knock-in or knock-out barrier level. For FX, this should be quoted as amount of `CCY2` per unit `CCY1` as defined in *Index*.
  Allowable values: Any positive real number.
- [barrierType] `BarrierType`: Barrier type.
  Allowable values: *DownIn, UpIn, DownOut, UpOut*
- [optionType] `PutCall`: Option type. For FX, this should be Call if we buy `CCY1` and sell `CCY2`, Put if we buy `CCY2` and sell `CCY1` (with `CCY1`, `CCY2` defined in *Index*).
  Allowable values: *Call, Put*
- [longShort] `LongShort`: *Long* if we buy the option. *Short* if we sell the option.
  Allowable values: *Long, Short*
- [number] `Quantity`: Number of option contracts. For FX, this should be the bought

amount.
Allowable values: Any positive real number.

- [index] `Underlying`: Underlying index.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes. Allowable values: See `Currency` in Table 20.
- [event] `PremiumPayDate`: Option premium payment date.
  Allowable values: See `Date` in Table 20.
- [number] `PremiumAmount`: Option premium amount in *PremiumPayCcy*.
  Allowable values: Any positive real number.
- [currency] `PremiumCcy`: Option premium currency.
  Allowable values: See Table 22 `Currency`.

19.2.75 Generic Barrier Option

Generic Barrier Options are defined using one of the trade types *FxGenericBarrierOption*, *EquityGenericBarrierOption*, *CommodityGenericBarrierOption* depending on the underlying asset class and an associated node FxGenericBarrierOptionData, EquityGenericBarrierOptionData, CommodityGenericBarrierOptionData. Listing 151 shows an example for an FX Underlying. Generic Barrier Option can have one or multiple underlyings. In the case of multiple underlyings, there must be one level per underlying provided in each barrier, see 152. The nodes have the following meaning:

- Underlying: The underlying definition. Note that for FX underlyings the order of the currencies defines the observed underlying value, i.e. for EUR-USD the domestic currency is USD (the observed value is e.g. 1.2 USD per EUR) while for USD-EUR the domestic currency is EUR (the observed value is e.g. 0.8 EUR per USD).

  Allowable Values: See 19.3.29
- Underlyings: An alternative to *Underlying* - only one can be present. Can contain multiple *Underlying* nodes. Allowable Values: A list of *Underlying* nodes, with each node given by 19.3.29
- PayCurrency: The payment currency. This is required for all Payoff Types and is usually
  - the domestic currency if underlying = FX
  - the eq / comm currency if underlying = Equity, Commodity

  But we allow for quanto payoffs as well, i.e.
  - the foreign currency if underlying = FX or also
  - a third currency if underlying = FX and

  - a currency not equal to the equity, commodity currency for these underlying types.

  See payoff description which amount is paid in which currency dependent on the type.

  Allowable Values: See `Currency` in Table 20.

- OptionData: The option descripting, see 19.3.1 Relevant sub nodes are:
  - LongShort (allowable values: *Long* or *Short*)
  - PayoffType, with S = Underlying Value and K = Strike this is:
    * *Vanilla*: $\max(0, S - K)$ for a call or $\max(0, K - S)$ for a put, this is paid in PayCurrency
    * *AssetOrNothing*: S paid in PayCurrency
    * *CashOrNothing*: Amount paid in PayCurrency
  - OptionType: Required for PayoffType = *Vanilla*, *Call* or *Put*.
  - ExerciseDate: The exercise date
  - Premiums [Optional]: Option premiums to be paid unconditionally. See section 19.3.2

- SettlementDate [Optional]: The date on which the option payoff is settled. The SettlementDate is used unadjusted as given. Instead of the SettlementDate, a settlement lag, convention and calendar relative to the ExerciseDate can be specified, see below. If the SettlementDate is given on the other hand, SettlementLag, SettlementCalendar and SettlementConvention must *not* be given.

  Allowable Values: any valid date greater or equal to the exercise date

- SettlementLag [Optional]: Alternative specification of the option settlement date via a lag, see the explanation under SettlementDate. Defaults to 0D if not given.

  Allowable Values: any valid period (1D, 2W, 3M, ...)

- SettlementCalendar [Optional]: Alternative specification of the option settlement date via a lag, see the explanation under SettlementDate. Defaults to the underlying calendar, if not given.

  Allowable values: See Table 24 Calendar.

- SettlementConvention [Optional]: Alternative specification of the option settlement date via a lag, see the explanation under SettlementDate. Defaults to Following if not given.

  Allowable values: See Table 21 Roll Convention.

- Quantity: The option quantity. Required for PayoffType = *AssetOrNothing*, *Vanilla*. For FX this is the amount in foreign currency. For Equity, Commodity this is the number of equities, commodities.

  Allowable Values: any real number

- Strike: Required for PayoffType = *Vanilla*.

  Allowable Values: any real number

- Amount: Required for PayoffType = *CashOrNothing*.

  Allowable Values: any real number

- Barriers. The barrier definition. Subnodes are:
  - ScheduleData [Optional]: the observation schedule for the barrier (see 19.3.4. Instead of the ScheduleData, a daily schedule w.r.t. the underlying calendar can be specified by populating the StartDate and EndDate nodes.
  - StartDate [Optional]: Start date of the observation schedule, see the explanation under ScheduleData.
  - EndDate [Optional]: End date of the observation schedule, see the explanation under ScheduleData.
  - BarrierData [Optional]: a sequence of barrier definitions. See 19.3.31. Relevant fields are:
    - Type: The barrier type (allowed values are *UpAndIn*, *UpAndOut*, *DownAndIn*, *DownAndOut*)
    - Levels: Exactly one barrier level per BarrierData block must be given.
    - Rebate [Optional]: The rebate amount. Defaults to zero. Rebate amounts and currencies can be different across barriers if only "out" barriers are defined, but must be identical as soon as at least one "in" barrier is defined.
    - RebateCurrency [Optional]: The currency in which the rebate is paid. Defaults to PayCurrency.
    - RebatePayTime [Optional]: *atExpiry* or *atHit*. For "in" barriers only *atExpiry* is allowed.
    - StrictComparison [Optional]: *0*, *1*, or *2*. Defaults to *0*. Determines how the barrier is checked as per:

      0: the barrier checks use <=, >= to check In-barriers and <, > to check Out-barriers.

      1: the barrier checks use strict comparison < and > for both In- and Out-barriers.

      2: the barrier checks use strict or equal comparison <= and >= for both In- and Out-barriers.
  - KikoType: Required if both a KI and KO barriers are defined. Allowable values are *KoAlways*, *KoBeforeKi*, *KoAfterKi*.

- TransatlanticBarrier [Optional]: An additional barrier to be checked on option expiry. May contain exactly one BarrierData block with number of levels equal to the number of underlyings or contain same amount of BarrierData blocks with the number of underlyings, exactly one level is allowed in each block for each underlying. The Type (*UpAndIn, UpAndOut, DownAndIn, DownAndOut*), the (unique) Level, StrictComparison and the Rebate are relevant fields.
  The rebate is paid if there is no knock-out from the American barriers and no payoff from the Transatlantic barrier.

Listing 151: Generic Barrier Option data (FX Underlying)

```
<FxGenericBarrierOptionData>
  <Underlying>
    <Type>FX</Type>
    <Name>ECB-EUR-USD</Name>
  </Underlying>
  <PayCurrency>USD</PayCurrency>
  <OptionData>
    <LongShort>Long</LongShort>
    <PayoffType>Vanilla</PayoffType>
    <OptionType>Call</OptionType>
    <ExerciseDates>
      <ExerciseDate>2023-06-06</ExerciseDate>
    </ExerciseDates>
  </OptionData>
  <SettlementDate>2023-06-08</SettlementDate>
  <Quantity>100000000</Quantity>
  <Strike>1.2</Strike>
  <Barriers>
    <ScheduleData>
      <Rules>
        <StartDate>2021-07-10</StartDate>
        <EndDate>2023-06-06</EndDate>
        <Tenor>1D</Tenor>
        <Calendar>TGT,US</Calendar>
        <Convention>F</Convention>
        <TermConvention>F</TermConvention>
        <Rule>Forward</Rule>
      </Rules>
    </ScheduleData>
    <BarrierData>
      <Type>DownAndOut</Type>
      <Levels>
        <Level>1.1</Level>
      </Levels>
      <Rebate>1000000</Rebate>
      <RebateCurrency>USD</RebateCurrency>
      <RebatePayTime>atExpiry</RebatePayTime>
      <StrictComparison>1</StrictComparison>
    </BarrierData>
    <BarrierData>
      <Type>UpAndIn</Type>
      <Levels>
        <Level>1.3</Level>
      </Levels>
      <Rebate>1000000</Rebate>
      <RebateCurrency>USD</RebateCurrency>
      <RebatePayTime>atExpiry</RebatePayTime>
    </BarrierData>
    <KikoType>KoAfterKi</KikoType>
  </Barriers>
  <TransatlanticBarrier>
    <BarrierData>
      <Type>UpAndOut</Type>
      <Levels>
        <Level>1.3</Level>
      </Levels>
      <Rebate>2000000</Rebate>
      <RebateCurrency>USD</RebateCurrency>
      <StrictComparison>1</StrictComparison>
    </BarrierData>
  </TransatlanticBarrier>
</FxGenericBarrierOptionData>
```

Listing 152: Multi Asset Generic Barrier Option data (FX Underlyings)

```
<FxGenericBarrierOptionData>
  <Underlyings>
    <Underlying>
      <Type>FX</Type>
      <Name>ECB-EUR-USD</Name>
    </Underlying>
    <Underlying>
      <Type>FX</Type>
      <Name>ECB-USD-JPY</Name>
    </Underlying>
  </Underlyings>
  <PayCurrency>USD</PayCurrency>
  <OptionData>
    <LongShort>Long</LongShort>
    <PayoffType>Vanilla</PayoffType>
    <OptionType>Call</OptionType>
    <ExerciseDates>
      <ExerciseDate>2023-06-06</ExerciseDate>
    </ExerciseDates>
  </OptionData>
  <Amount>1500000</Amount>
  <SettlementDate>2023-06-08</SettlementDate>
  <Barriers>
    <ScheduleData>
      <Rules>
        <StartDate>2021-07-10</StartDate>
        <EndDate>2023-06-06</EndDate>
        <Tenor>1D</Tenor>
        <Calendar>TGT,US</Calendar>
        <Convention>F</Convention>
        <TermConvention>F</TermConvention>
        <Rule>Forward</Rule>
      </Rules>
    </ScheduleData>
    <BarrierData>
      <Type>DownAndOut</Type>
      <Levels>
        <Level>1.1</Level>
        <Level>125</Level>
      </Levels>
      <Rebate>1000000</Rebate>
      <RebateCurrency>USD</RebateCurrency>
      <RebatePayTime>atExpiry</RebatePayTime>
    </BarrierData>
    <BarrierData>
      <Type>UpAndIn</Type>
      <Levels>
        <Level>1.3</Level>
        <Level>135</Level>
      </Levels>
      <Rebate>1000000</Rebate>
      <RebateCurrency>USD</RebateCurrency>
      <RebatePayTime>atExpiry</RebatePayTime>
    </BarrierData>
    <KikoType>KoAfterKi</KikoType>
  </Barriers>
  <TransatlanticBarrier>
    <BarrierData>
      <Type>UpAndOut</Type>
      <Levels>
        <Level>1.3</Level>
        <Level>135</Level>
```

19.2.76 Best Entry Option

Best Entry Options are defined using one of the trade types *FxBestEntryOption*, *EquityBestEntryOption*, *CommodityBestEntryOption* depending on the underlying asset class and an associated node FxBestEntryOptionData, EquityBestEntryOptionData, CommodityBestEntryOptionData. Listing 153 shows an example for an Equity Underlying. For a more detailed description of the computation of the payoff of this option, please see the product description. The nodes have the following meaning:

- Underlying: The underlying definition. Note that for FX underlyings the order of the currencies defines the observed underlying value, i.e. for EUR-USD the domestic currency is USD (the observed value is e.g. 1.2 USD per EUR) while for USD-EUR the domestic currency is EUR (the observed value is e.g. 0.8 EUR per USD).

  Allowable Values: See 19.3.29

- Currency: The payment currency.

  Allowable Values: See `Currency` in Table 20.

- SettlementDate: The date on which the option payoff is settled. The SettlementDate is used unadjusted as given.

  Allowable Values: any valid date greater or equal to the exercise date.

- Notional: The notional amount.

  Allowable Values: any real number

- Strike: The strike value used to compute the payoff of the option. This value should be provided as a decimal, representing a percentage of the value of $Index_{Initial}$, e.g. a value of $K = 0.6 \implies 0.6 \times Index_{Initial}$ in the computation of the payoff. Allowable Values: any real number

- Multiplier: The payoff multiplier used in the case that the underlying index is greater than the strike on the settlement date. If omitted defaults to 1.

  Allowable Values: any real number

- TriggerLevel: The value that is compared to the underlying index on each strike observation date to determine if a Trigger Event has occurred. This should be provided as a decimal, representing a percentage of the value of Strike Index Level, ie the value of the underlying on the `StrikeDate`.

  Allowable Values: any real number

- LongShort: Denotes whether the payoff is computed relative to the holder or seller of the option.

  Allowable Values: *Long*, *Short*.

- Cap: The maximum value of the payoff (before the notional and multiplier are applied). This value should be interpreted as a percentage and should be in decimal format in the trade XML, e.g. 0.06 = 6%.

  Allowable Values: any real number

- ResetMinimum: The minimum value of $Index_{Initial}$ in the case that a Trigger Event has occurred at least once during the option's lifetime. This should be provided as a decimal, representing a percentage of the value of Strike Index Level used in the computation of $Index_{Initial}$.

  Allowable Value: any real number

- StrikeDate: The date on which the level on the underlying index is used to compute the payoff, in the case that there has not been a Trigger Event during the option's lifetime.

  Allowable Value: any valid date before the option expiry date.

- ExpiryDate: The date on which the option expires and the payoff is computed.

  Allowable Values: any valid date before the SettlementDate

- Premium: The option premium. Defaults to 0.

  Allowable Values: any real number.

- PremiumDate: The date on which the option premium is paid.

  Allowable Values: any valid date.

- StrikeObservationDates: The set of dates on which the underlying index level is observed - the lowest of which is used to compute the option payoff if the underlying index is greater than the strike on the expiry date.

  Allowable Values: any valid dates schedule (see 19.3.4).

Listing 153: Best Entry Option data (Equity Underlying)

```
<EquityBestEntryOptionData>
  <LongShort>Long</LongShort>
  <Strike>0.85</Strike>
  <Cap>0.06</Cap>
  <ResetMinimum>0.85</ResetMinimum>
  <Notional>1000000</Notional>
  <Multiplier>1</Multiplier>
  <TriggerLevel>0.95</TriggerLevel>
  <SettlementDate>2021-11-20</SettlementDate>
  <PremiumDate>2021-11-22</PremiumDate>
  <StrikeDate>2020-12-15</StrikeDate>
  <Underlying>
    <Type>Equity</Type>
    <Name>RIC:.SPX</Name>
  </Underlying>
  <StrikeObservationDates>
    <Dates>
      <Dates>
        <Date>2021-03-01</Date>
        <Date>2021-06-01</Date>
        <Date>2021-09-01</Date>
      </Dates>
    </Dates>
  </StrikeObservationDates>
  <Currency>USD</Currency>
  <Premium>100</Premium>
  <ExpiryDate>2021-11-20</ExpiryDate>
</EquityBestEntryOptionData>
```

19.2.77 Basket Options

Basket Options are represented as traditional trades or *scripted trades*, refer to vre/docs/scripted_trade for an introduction of the latter. Each of the supported variations is represented by a separate payoff script as shown in Table 89.

Basket Option Type	Payoff Script Name
Vanilla	VanillaBasketOption
Asian	AsianBasketOption
Average strike	AverageStrikeBasketOption
Lookback call	LookbackCallBasketOption
Lookback put	LookbackPutBasketOption

Table 89: Basket option types and associated script names.

The supported underlying types are Equity, Fx or Commodity resulting in corresponding trade types and trade data container names

- EquityBasketOption / EquityBasketOptionData
- FxBasketOption / FxBasketOptionData
- CommodityBasketOption / CommodityBasketOptionData

Trade input and the associated payoff script are described in the following for all supported Basket Option variations.

Vanilla Basket Option

The traditional trade representation is as follows, using an equity underlying in this example:

```
<Trade id="VanillaBasketOption#1">
  <TradeType>EquityBasketOption</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <EquityBasketOptionData>
    <Currency>USD</Currency>
    <Notional>1</Notional>
    <Strike>5000</Strike>
    <Underlyings>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.SPX</Name>
        <Weight>1.0</Weight>
      </Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.STOXX50E</Name>
        <Currency>EUR</Currency>
        <Weight>1.0</Weight>
      </Underlying>
    </Underlyings>
    <OptionData>
      <LongShort>Long</LongShort>
      <OptionType>Call</OptionType>
      <PayoffType>Vanilla</PayoffType>
      <ExerciseDates>
        <ExerciseDate>2020-02-15</ExerciseDate>
      </ExerciseDates>
      <Premiums> ... </Premiums>
    </OptionData>
    <Settlement>2020-02-20</Settlement>
  </EquityBasketOptionData>
</Trade>
```

with the following elements:

- Currency: The pay currency. Notice section “Payment Currency” in vre/docs/scripted_trade.
  Allowable values: all supported currency codes, see Table 22 `Currency`.
- Notional: The quantity (for equity, commodity underlyings) / foreign amount (fx underlying)
  Allowable values: all positive real numbers
- Strike: The strike of the option
  Allowable values: all positive real numbers
- Underlyings: The basket of underlyings.
  Allowable values: for each underlying see 19.3.29
- OptionData: relevant are the long/short flag, the call/put flag, the payoff type (must be set to Vanilla to identify the payoff), and the exercise date (exactly one date must be given). A *Premiums* node can be added to represent deterministic option premia to be

paid by the option holder.
Allowable values: see 19.3.1 for the general structure of the option data node

- Settlement: the settlement date (optional, if not given defaults to the exercise date)
  Allowable values: each valid date.

The representation as a scripted trade is as follows:

```
<Trade id="VanillaBasketOption#1">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope/>
  <VanillaBasketOptionData>
    <Expiry type="event">2020-02-15</Expiry>
    <Settlement type="event">2020-02-20</Settlement>
    <PutCall type="optionType">Call</PutCall>
    <LongShort type="longShort">Long</LongShort>
    <Notional type="number">1</Notional>
    <Strike type="number">5000</Strike>
    <Underlyings type="index">
      <Value>EQ-RIC:.SPX</Value>
      <Value>EQ-RIC:.STOXX50E</Value>
    </Underlyings>
    <Weights type="number">
      <Value>1.0</Value>
      <Value>1.0</Value>
    </Weights>
    <PayCcy type = "currency">USD</PayCcy>
  </VanillaBasketOptionData>
</Trade>
```

The VanillaBasketOption script referenced in the trade above is shown in Listing 154.

Listing 154: Payoff script for a VanillaBasketOption.

```
REQUIRE SIZE(Underlyings) == SIZE(Weights);

NUMBER u, basketPrice, ExerciseProbability, Payoff, currentNotional;

FOR u IN (1, SIZE(Underlyings)) DO
    basketPrice = basketPrice + Underlyings[u](Expiry) * Weights[u];
END;

Payoff = max(PutCall * (basketPrice - Strike), 0);

Option = LongShort * Notional * PAY(Payoff, Expiry, Settlement, PayCcy);

IF Payoff > 0 THEN
    ExerciseProbability = 1;
END;
currentNotional = Notional * Strike;
```

The meanings and allowable values of the elements in the `VanillaBasketOptionData` node are given below, with data type indicated in square brackets.

- [event] `Expiry`: Option expiry date.
  Allowable values: See `Date` in Table 20.
- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.
- [optionType] `PutCall`: Option type with
  Allowable values *Call, Put*.
- [longShort] `LongShort`: Position type, *Long* if we buy, *Short* if we sell.

Allowable values: *Long, Short*.

- [number] `Notional`: Quantity multiplier applied to the basket price
  Allowable values: Any positive real number.
- [number] `Strike`: Strike basket price in PayCcy (see below)
  Allowable values: Any positive real number.
- [index] `Underlyings`: List of underlying indices enclosed by <Value> and </Value> tags.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `Weights`: List of weights applied to the underlying prices in the basket, given in the same order as the Underlyings above, each weight enclosed by <Value> and </Value> tags.
  Allowable values: Any positive real number.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 `Currency` for allowable currency codes.

Asian Basket Option

The traditional trade representation is as follows, using an equity underlying in this example:

```
<Trade id="AsianBasketOption#1">
  <TradeType>EquityBasketOption</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <EquityBasketOptionData>
    <Currency>USD</Currency>
    <Notional>1</Notional>
    <Strike>5000</Strike>
    <Underlyings>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.SPX</Name>
        <Currency>USD</Currency>
        <Weight>1.0</Weight>
      </Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.STOXX50E</Name>
        <Weight>1.0</Weight>
      </Underlying>
    </Underlyings>
    <OptionData>
      <LongShort>Long</LongShort>
      <OptionType>Call</OptionType>
      <PayoffType>Asian</PayoffType>
      <ExerciseDates>
        <ExerciseDate>2020-02-15</ExerciseDate>
      </ExerciseDates>
      <Premiums> ... </Premiums>
    </OptionData>
    <Settlement>2020-02-20</Settlement>
    <ObservationDates>
        <Rules>
          <StartDate>2019-02-06</StartDate>
```

```
          <EndDate>2020-02-06</EndDate>
          <Tenor>1D</Tenor>
          <Calendar>US</Calendar>
          <Convention>F</Convention>
          <TermConvention>F</TermConvention>
          <Rule>Forward</Rule>
        </Rules>
    </ObservationDates>
  </EquityBasketOptionData>
</Trade>
```

with the following elements:

- Currency: The pay currency. Notice section “Payment Currency” in vre/docs/scripted_trade.
  Allowable values: all supported currency codes, see Table 22 `Currency`.
- Notional: The quantity (for equity, commodity underlyings) / foreign amount (fx underlying)
  Allowable values: all positive real numbers
- Strike: The strike of the option
  Allowable values: all positive real numbers
- Underlyings: The basket of underlyings.
  Allowable values: for each underlying see 19.3.29
- OptionData: relevant are the long/short flag, the call/put flag, the payoff type (must be set to Asian to identify the payoff), and the exercise date (exactly one date must be given). `PayoffType2` can be optionally set to *Arithmetic* or *Geometric* and defaults to *Arithmetic* if not given. Furthermore, a *Premiums* node can be added to represent deterministic option premia to be paid by the option holder.
  Allowable values: see 19.3.1 for the general structure of the option data node
- Settlement: the settlement date (optional, if not given defaults to the exercise date)
  Allowable values: each valid date.
- ObservationDates: the observation dates for the asian
  Allowable values: See the definition in 19.3.4

The representation as a scripted trade is as follows:

```
<Trade id="AsianBasketOption#1">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope/>
   <AsianBasketOptionData>
    <Expiry type="event">2020-02-15</Expiry>
    <Settlement type="event">2020-02-20</Settlement>
    <ObservationDates type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2019-02-06</StartDate>
          <EndDate>2020-02-06</EndDate>
          <Tenor>1D</Tenor>
          <Calendar>US</Calendar>
          <Convention>F</Convention>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
    </ObservationDates>
    <PutCall type="optionType">Call</PutCall>
    <LongShort type="longShort">Long</LongShort>
    <Notional type="number">1</Notional>
```

```
    <Strike type="number">5000</Strike>
    <Underlyings type="index">
      <Value>EQ-RIC:.SPX</Value>
      <Value>EQ-RIC:.STOXX50E</Value>
    </Underlyings>
    <Weights type="number">
      <Value>1.0</Value>
      <Value>1.0</Value>
    </Weights>
    <PayCcy type="currency">USD</PayCcy>
  </AsianBasketOptionData>
</Trade>
```

The AsianBasketOption script referenced in the trade above is shown in Listing 155.

Listing 155: Payoff script for an AsianBasketOption.

```
REQUIRE SIZE(Underlyings) == SIZE(Weights);

NUMBER d, u, basketPrice, ExerciseProbability, Payoff;
NUMBER currentNotional;

FOR d IN (1, SIZE(ObservationDates)) DO
    FOR u IN (1, SIZE(Underlyings)) DO
        basketPrice = basketPrice + Underlyings[u](ObservationDates[d]) * Weights[u];
    END;
END;

basketPrice = basketPrice / SIZE(ObservationDates);

Payoff = max(PutCall * (basketPrice - Strike), 0);

Option = LongShort * Notional * PAY(Payoff, Expiry, Settlement, PayCcy);

IF Payoff > 0 THEN
    ExerciseProbability = 1;
END;

currentNotional = Notional * Strike;
```

The meanings and allowable values of the elements in the `AsianBasketOptionData` node are given below, with data type indicated in square brackets.

- [event] `Expiry`: Option expiry date.
  Allowable values: See `Date` in Table 20.
- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.
- [event] `ObservationDates`: Price monitoring schedule.
  Allowable values: See section 19.3.4 Schedule Data and Dates, or DerivedSchedule (see vre/docs/scripted_trade).
- [optionType] `PutCall`: Option type with
  Allowable values *Call, Put*.
- [longShort] `LongShort`: Position type, *Long* if we buy, *Short* if we sell.
  Allowable values: *Long, Short*.
- [number] `Notional`: Quantity multiplier applied to the basket price
  Allowable values: Any positive real number.
- [number] `Strike`: Strike basket price in PayCcy (see below)
  Allowable values: Any positive real number.

- [index] `Underlyings`: List of underlying indices enclosed by <Value> and </Value> tags.
  See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `Weights`: List of weights applied to the underlying prices in the basket, given in the same order as the Underlyings above, each weight enclosed by <Value> and </Value> tags.
  Allowable values: Any positive real number.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 `Currency` for allowable currency codes.

Average Strike Basket Option

The traditional trade representation is as follows, using an equity underlying in this example:

```
<Trade id="AverageStrikeBasketOption#1">
  <TradeType>EquityBasketOption</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <EquityBasketOptionData>
    <Currency>USD</Currency>
    <Notional>1</Notional>
    <Underlyings>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.SPX</Name>
        <Currency>USD</Currency>
        <Weight>1.0</Weight>
      </Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.STOXX50E</Name>
        <Weight>1.0</Weight>
      </Underlying>
    </Underlyings>
    <OptionData>
      <LongShort>Long</LongShort>
      <OptionType>Call</OptionType>
      <PayoffType>AverageStrike</PayoffType>
      <ExerciseDates>
        <ExerciseDate>2020-02-15</ExerciseDate>
      </ExerciseDates>
      <Premiums> ... </Premiums>
    </OptionData>
    <Settlement>2020-02-20</Settlement>
    <ObservationDates>
      <Rules>
        <StartDate>2019-02-06</StartDate>
        <EndDate>2020-02-06</EndDate>
        <Tenor>1D</Tenor>
        <Calendar>US</Calendar>
        <Convention>F</Convention>
        <TermConvention>F</TermConvention>
        <Rule>Forward</Rule>
      </Rules>
    </ObservationDates>
  </EquityBasketOptionData>
</Trade>
```

with the following elements:

- Currency: The pay currency. Notice section “Payment Currency” in vre/docs/scripted_trade.
  Allowable values: all supported currency codes, see Table 22 `Currency`.
- Notional: The quantity (for equity, commodity underlyings) / foreign amount (fx underlying)
  Allowable values: all positive real numbers
- Underlyings: The basket of underlyings.
  Allowable values: for each underlying see 19.3.29
- OptionData: relevant are the long/short flag, the call/put flag, the payoff type (must be set to AverageStrike to identify the payoff), and the exercise date (exactly one date must be given). `PayoffType2` can be optionally set to *Arithmetic* or *Geometric* and defaults to *Arithmetic* if not given. Furthermore, a *Premiums* node can be added to represent deterministic option premia to be paid by the option holder.
  Allowable values: see 19.3.1 for the general structure of the option data node
- Settlement: the settlement date (optional, if not given defaults to the exercise date)
  Allowable values: each valid date.
- ObservationDates: the observation dates for the average strike
  Allowable values: See the definition in 19.3.4

The representation as a scripted trade is as follows:

```
<Trade id="AverageStrikeBasketOption#1">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope/>
  <AverageStrikeBasketOptionData>
    <Expiry type="event">2020-02-15</Expiry>
    <Settlement type="event">2020-02-20</Settlement>
    <ObservationDates type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2019-02-06</StartDate>
          <EndDate>2020-02-06</EndDate>
          <Tenor>1D</Tenor>
          <Calendar>US</Calendar>
          <Convention>F</Convention>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
    </ObservationDates>
    <PutCall type="optionType">Call</PutCall>
    <LongShort type="longShort">Long</LongShort>
    <Notional type="number">1</Notional>
    <Underlyings type="index">
      <Value>EQ-RIC:.SPX</Value>
      <Value>EQ-RIC:.STOXX50E</Value>
    </Underlyings>
    <Weights type="number">
      <Value>1.0</Value>
      <Value>1.0</Value>
    </Weights>
    <PayCcy type="currency">USD</PayCcy>
  </AverageStrikeBasketOptionData>
</Trade>
```

The AverageStrikeBasketOption script referenced in the trade above is shown in Listing 156.

Listing 156: Payoff script for an AverageStrikeBasketOption.

```
REQUIRE SIZE(Underlyings) == SIZE(Weights);

NUMBER d, u, timeAverageBasketPrice, currentNotional;
FOR d IN (1, SIZE(ObservationDates)) DO
    FOR u IN (1, SIZE(Underlyings)) DO
        timeAverageBasketPrice = timeAverageBasketPrice
          + Underlyings[u](ObservationDates[d]) * Weights[u];
    END;
END;
timeAverageBasketPrice = timeAverageBasketPrice / SIZE(ObservationDates);

NUMBER expiryBasketPrice;
FOR u IN (1, SIZE(Underlyings)) DO
   expiryBasketPrice = expiryBasketPrice + Underlyings[u](Expiry) * Weights[u];
END;

NUMBER Payoff;
Payoff = max(PutCall * (expiryBasketPrice - timeAverageBasketPrice), 0);

Option = LongShort * Notional * PAY(Payoff, Expiry, Settlement, PayCcy);

NUMBER ExerciseProbability;
IF Payoff > 0 THEN
    ExerciseProbability = 1;
END;
FOR u IN (1, SIZE(Underlyings)) DO
  currentNotional = currentNotional + Notional * Underlyings[u](ObservationDates[1])
                                               * Weights[u];
END;
```

The meanings and allowable values of the elements in the `AverageStrikeBasketOptionData` node are given below, with data type indicated in square brackets.

- [event] `Expiry`: Option expiry date.
  Allowable values: See `Date` in Table 20.
- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.
- [event] `ObservationDates`: Price monitoring schedule.
  Allowable values: See section 19.3.4 Schedule Data and Dates, or DerivedSchedule (see vre/docs/scripted_trade).
- [optionType] `PutCall`: Option type with
  Allowable values *Call, Put*.
- [longShort] `LongShort`: Position type, *Long* if we buy, *Short* if we sell.
  Allowable values: *Long, Short*.
- [number] `Notional`: Quantity multiplier applied to the basket price
  Allowable values: Any positive real number.
- [index] `Underlyings`: List of underlying indices enclosed by `<Value>` and `</Value>` tags.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `Weights`: List of weights applied to the underlying prices in the basket, given in the same order as the Underlyings above, each weight enclosed by `<Value>` and `</Value>` tags.

Allowable values: Any positive real number.

- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 `Currency` for allowable currency codes.

Lookback Call Basket Option

The traditional trade representation is as follows, using an equity underlying in this example:

```
<Trade id="LookbackCallBasketOption#1">
  <TradeType>EquityBasketOption</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <EquityBasketOptionData>
    <Currency>USD</Currency>
    <Notional>1</Notional>
    <Underlyings>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.SPX</Name>
        <Currency>USD</Currency>
        <Weight>1.0</Weight>
      </Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.STOXX50E</Name>
        <Weight>1.0</Weight>
      </Underlying>
    </Underlyings>
    <OptionData>
      <LongShort>Long</LongShort>
      <PayoffType>LookbackCall</PayoffType>
      <ExerciseDates>
        <ExerciseDate>2020-02-15</ExerciseDate>
      </ExerciseDates>
      <Premiums> ... </Premiums>
    </OptionData>
    <Settlement>2020-02-20</Settlement>
    <ObservationDates>
      <Rules>
        <StartDate>2019-02-06</StartDate>
        <EndDate>2020-02-06</EndDate>
        <Tenor>1D</Tenor>
        <Calendar>US</Calendar>
        <Convention>F</Convention>
        <TermConvention>F</TermConvention>
        <Rule>Forward</Rule>
      </Rules>
    </ObservationDates>
  </EquityBasketOptionData>
</Trade>
```

with the following elements:

- Currency: The pay currency. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: all supported currency codes, see Table 22 `Currency`.
- Notional: The quantity (for equity, commodity underlyings) / foreign amount (fx underlying)

Allowable values: all positive real numbers

- Underlyings: The basket of underlyings.
  Allowable values: for each underlying see 19.3.29
- OptionData: relevant are the long/short flag, the payoff type (must be set to LookbackCall to identify the payoff), and the exercise date (exactly one date must be given). Furthermore, a *Premiums* node can be added to represent deterministic option premia to be paid by the option holder.
  Allowable values: see 19.3.1 for the general structure of the option data node
- Settlement: the settlement date (optional, if not given defaults to the exercise date)
  Allowable values: each valid date.
- ObservationDates: the observation dates for the lookback call
  Allowable values: See the definition in 19.3.4

The representation as a scripted trade is as follows:

```
<Trade id="LookbackCallBasketOption#1">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope/>
  <LookbackCallBasketOptionData>
    <Expiry type="event">2020-02-15</Expiry>
    <Settlement type="event">2020-02-20</Settlement>
    <ObservationDates type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2019-02-06</StartDate>
          <EndDate>2020-02-06</EndDate>
          <Tenor>1D</Tenor>
          <Calendar>US</Calendar>
          <Convention>F</Convention>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
    </ObservationDates>
    <LongShort type="longShort">Long</LongShort>
    <Notional type="number">1</Notional>
    <Underlyings type="index">
      <Value>EQ-RIC:.SPX</Value>
      <Value>EQ-RIC:.STOXX50E</Value>
    </Underlyings>
    <Weights type="number">
      <Value>1.0</Value>
      <Value>1.0</Value>
    </Weights>
    <PayCcy type="currency">USD</PayCcy>
  </LookbackCallBasketOptionData>
</Trade>
```

The LookbackCallBasketOption script referenced in the trade above is shown in Listing 157.

The meanings and allowable values of the elements in the `LookbackCallBasketOptionData` node are given below, with data type indicated in square brackets.

- [event] `Expiry`: Option expiry date.
  Allowable values: See `Date` in Table 20.
- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.

Listing 157: Payoff script for a LookbackCallBasketOption.

```
REQUIRE SIZE(Underlyings) == SIZE(Weights);

NUMBER d, u, basketPrice, minBasketPrice, currentNotional;
FOR d IN (1, SIZE(ObservationDates)) DO
    basketPrice = 0;
    FOR u IN (1, SIZE(Underlyings)) DO
        basketPrice = basketPrice + Underlyings[u](ObservationDates[d]) * Weights[u];
    END;
    IF d == 1 THEN
        minBasketPrice = basketPrice;
    END;
    IF basketPrice < minBasketPrice THEN
        minBasketPrice = basketPrice;
    END;
END;

NUMBER expiryBasketPrice;
FOR u IN (1, SIZE(Underlyings)) DO
   expiryBasketPrice = expiryBasketPrice + Underlyings[u](Expiry) * Weights[u];
END;

NUMBER Payoff;
Payoff = max(expiryBasketPrice - minBasketPrice, 0);

Option = LongShort * Notional * PAY(Payoff, Expiry, Settlement, PayCcy);

NUMBER ExerciseProbability;
IF Payoff > 0 THEN
    ExerciseProbability = 1;
END;
FOR u IN (1, SIZE(Underlyings)) DO
  currentNotional = currentNotional + Notional * Underlyings[u](ObservationDates[1])
                                                * Weights[u];
END;
```

- [event] `ObservationDates`: Price monitoring schedule.
  Allowable values: See section 19.3.4 Schedule Data and Dates, or DerivedSchedule (see vre/docs/scripted_trade).
- [longShort] `LongShort`: Position type, *Long* if we buy, *Short* if we sell.
  Allowable values: *Long, Short*.
- [number] `Notional`: Quantity multiplier applied to the basket price
  Allowable values: Any positive real number.
- [index] `Underlyings`: List of underlying indices enclosed by <Value> and </Value> tags.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `Weights`: List of weights applied to the underlying prices in the basket, given in the same order as the Underlyings above, each weight enclosed by <Value> and </Value> tags.
  Allowable values: Any positive real number.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 `Currency` for allowable currency codes.

Lookback Put Basket Option

The traditional trade representation is as follows, using an equity underlying in this example:

```
<Trade id="LookbackPutBasketOption#1">
  <TradeType>EquityBasketOption</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <EquityBasketOptionData>
    <Currency>USD</Currency>
    <Notional>1</Notional>
    <Underlyings>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.SPX</Name>
        <Currency>USD</Currency>
        <Weight>1.0</Weight>
      </Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.STOXX50E</Name>
        <Weight>1.0</Weight>
      </Underlying>
    </Underlyings>
    <OptionData>
      <LongShort>Long</LongShort>
      <PayoffType>LookbackPut</PayoffType>
      <ExerciseDates>
        <ExerciseDate>2020-02-15</ExerciseDate>
      </ExerciseDates>
      <Premiums> ... </Premiums>
    </OptionData>
    <Settlement>2020-02-20</Settlement>
    <ObservationDates>
      <Rules>
        <StartDate>2019-02-06</StartDate>
        <EndDate>2020-02-06</EndDate>
        <Tenor>1D</Tenor>
        <Calendar>US</Calendar>
        <Convention>F</Convention>
        <TermConvention>F</TermConvention>
        <Rule>Forward</Rule>
      </Rules>
    </ObservationDates>
  </EquityBasketOptionData>
</Trade>
```

with the following elements:

- Currency: The pay currency. Notice section “Payment Currency” in vre/docs/scripted_trade.
  Allowable values: all supported currency codes, see Table 22 `Currency`.
- Notional: The quantity (for equity, commodity underlyings) / foreign amount (fx underlying)
  Allowable values: all positive real numbers
- Underlyings: The basket of underlyings.
  Allowable values: for each underlying see 19.3.29
- OptionData: relevant are the long/short flag, the payoff type (must be set to LookbackPut to identify the payoff), and the exercise date (exactly one date must be given). Furthermore, a *Premiums* node can be added to represent deterministic option

premia to be paid by the option holder.
Allowable values: see 19.3.1 for the general structure of the option data node

- Settlement: the settlement date (optional, if not given defaults to the exercise date)
  Allowable values: each valid date.
- ObservationDates: the observation dates for the lookback call
  Allowable values: See the definition in 19.3.4

The representation as a scripted trade is as follows:

```
<Trade id="LookbackPutBasketOption#1">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope/>
   <LookbackPutBasketOptionData>
    <Expiry type="event">2020-02-15</Expiry>
    <Settlement type="event">2020-02-20</Settlement>
    <ObservationDates type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2019-02-06</StartDate>
          <EndDate>2020-02-06</EndDate>
          <Tenor>1D</Tenor>
          <Calendar>US</Calendar>
          <Convention>F</Convention>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
    </ObservationDates>
    <LongShort type="longShort">Long</LongShort>
    <Notional type="number">1</Notional>
    <Underlyings type="index">
      <Value>EQ-RIC:.SPX</Value>
      <Value>EQ-RIC:.STOXX50E</Value>
    </Underlyings>
    <Weights type="number">
      <Value>1.0</Value>
      <Value>1.0</Value>
    </Weights>
    <PayCcy type="currency">USD</PayCcy>
  </LookbackPutBasketOptionData>
</Trade>
```

The LookbackCallBasketOption script referenced in the trade above is shown in Listing 158.

The meanings and allowable values of the elements in the `LookbackPutBasketOptionData` node are given below, with data type indicated in square brackets.

- [event] `Expiry`: Option expiry date.
  Allowable values: See `Date` in Table 20.
- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.
- [event] `ObservationDates`: Price monitoring schedule.
  Allowable values: See section 19.3.4 Schedule Data and Dates, or DerivedSchedule (see vre/docs/scripted_trade).
- [longShort] `LongShort`: Position type, *Long* if we buy, *Short* if we sell.
  Allowable values: *Long, Short*.
- [number] `Notional`: Quantity multiplier applied to the basket price
  Allowable values: Any positive real number.

Listing 158: Payoff script for a LookbackPutBasketOption.

```
REQUIRE SIZE(Underlyings) == SIZE(Weights);

NUMBER d, u, basketPrice, maxBasketPrice;
FOR d IN (1, SIZE(ObservationDates)) DO
    basketPrice = 0;
    FOR u IN (1, SIZE(Underlyings)) DO
        basketPrice = basketPrice + Underlyings[u](ObservationDates[d]) * Weights[u];
    END;
    IF d == 1 THEN
        maxBasketPrice = basketPrice;
    END;
    IF basketPrice > maxBasketPrice THEN
        maxBasketPrice = basketPrice;
    END;
END;

NUMBER expiryBasketPrice, Payoff;
FOR u IN (1, SIZE(Underlyings)) DO
   expiryBasketPrice = expiryBasketPrice + Underlyings[u](Expiry) * Weights[u];
END;

Payoff = max(maxBasketPrice - expiryBasketPrice, 0);
Option = LongShort * Notional * LOGPAY(Payoff, Expiry, Settlement, PayCcy);
```

- [index] `Underlyings`: List of underlying indices enclosed by <Value> and </Value> tags.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `Weights`: List of weights applied to the underlying prices in the basket, given in the same order as the Underlyings above, each weight enclosed by <Value> and </Value> tags.
  Allowable values: Any positive real number.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 `Currency` for allowable currency codes.

19.2.78 Worst Of Basket Swaps

The FxWorstOfBasketSwap, EquityWorstOfBasketSwap, and CommodityWorstOfBasketSwap trade types have trade data containers (respectively):

- `FxWorstOfBasketSwapData`
- `EquityWorstOfBasketSwapData`
- `CommodityWorstOfBasketSwapData`

Listing 159 shows the structure of example trades for an Equity underlying.

Listing 159: EquityWorstOfBasketSwap data

```
<Trade id="EQ_WorstOfBasketSwap">
  <TradeType>EquityWorstOfBasketSwap</TradeType>
  <Envelope>
    ......
  </Envelope>
  <EquityWorstOfBasketSwapData>
    <LongShort>Long</LongShort>
    <Currency>EUR</Currency>
    <Quantity>1955000</Quantity>
    <Underlyings>
      <Underlying>
        .......
      </Underlying>
    </Underlyings>
    <InitialPrices>
      <InitialPrice>...</InitialPrice>
    </InitialPrices>
    <FloatingPeriodSchedule>
      <Dates>
        ...
      </Dates>
    </FloatingPeriodSchedule>
    <FixedDeterminationSchedule>
          ............
    </FixedDeterminationSchedule>
    <KnockOutDeterminationSchedule>
           ............
    </KnockOutDeterminationSchedule>
    <FloatingPayDates>
      <Dates>
        ....
      </Dates>
    </FloatingPayDates>
    <KnockInDeterminationSchedule>
          ............
    </KnockInDeterminationSchedule>
    <FixedPayDates>
      ............
    </FixedPayDates>
    <KnockOutLevels>
      <KnockOutLevel>...</KnockOutLevel>
    </KnockOutLevels>
    <FixedTriggerLevels>
      <FixedTriggerLevel>...</FixedTriggerLevel>
    </FixedTriggerLevels>
    <KnockInLevel>0.75</KnockInLevel>
    <FixedRate>0.04</FixedRate>
    <FixedAccrualSchedule>
      <Rules>
        ...
      </Rules>
    </FixedAccrualSchedule>
    <FloatingIndex>EUR-EURIBOR-3M</FloatingIndex>
    <FloatingDayCountFraction>Actual/360</FloatingDayCountFraction>
    <FloatingFixingSchedule>
          ............
    </FloatingFixingSchedule>
  </EquityWorstOfBasketSwapData>
</Trade>
```

The net cashflows under a worst of basket swap from the long perspective are:

1. Receive an initial fixed amount: $\texttt{InitialFixedRate} * \texttt{Quantity}$,
2. At each determination date (if a knock-out has not occurred for any previous determination date):
    - Pay a floating amount: $\texttt{Quantity} * floatingRate * accrualFraction$,

- If a coupon trigger event occurs for this date, receive a fixed coupon amount (as well as any other fixed coupon amounts not previously received because a coupon trigger event had not occurred): `Quantity` $*$ `CouponRate`.

  If `AccumulatingFixedCoupons` is *True*, the coupon amounts are guaranteed to be paid out, with the amounts scaled relative to the fraction of days during the accrual period where a coupon trigger event occurs: `Quantity` $*$ `CouponRate` $*triggerAccrualFraction$

3. On the final determination date, if $worstPerformance < \min(Strike, KnockInLevel)$, pay `Quantity` $*$(`Strike`$-worstPerformance$), where $worstPerformance$ is the performance, i.e. S_T/S_0, of the worst-performing asset as of the final determination date T.

The meanings and allowable values of elements in the `EquityWorstOfBasketSwapData` node follow below.

- `Currency`: The payout currency for all cashflows. For FX, this should be `CCY2` as defined (see Table 28) in the `Name` field in the `Underlying` node. Note that for (non-quanto) FxWorstOfBasketSwaps, this should be the domestic (`CCY2`) currency, as defined in the Name field in the `Underlying` node. For non-quanto Equity- and CommodityWorstOfBasketSwaps, this should be currency the equity or commodity is quoted in. Notice Section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 `Currency`.

- `LongShort`: Own party position.
  Allowable values: *Long*, *Short*

- `Quantity`: The equity notional amount, or the quantity multiplier.
  Allowable values: Any number.

- `InitialFixedRate` [Optional]: The coupon rate for the initial fixed payment, to be paid to the *Long* party.
  Allowable values: Any number, as a percentage expressed in decimal form. If left blank or omitted, defaults to zero.

- `InitialFixedPayDate` [Optional]: Settlement date for the initial fixed payment.
  Allowable values: See `Date` in Table 20. If left blank or omitted, this defaults to the first date in the `FloatingPayDates` schedule.

- `Underlyings`: The basket of underlyings.
  Allowable values: For each underlying, see 19.3.29.

- `InitialPrices`: The observed price of each underlying in `Underlyings`.
  Allowable values: For each underlying, an `InitialPrice` sub-node with any positive number.

- `BermudanKnockIn` [Optional]: Whether the knock-in event is observed on a regular basis (*True*), as defined by the `KnockInDeterminationSchedule`, or only at the final date in the `FloatingPeriodSchedule` (*False*).
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36. If left blank or omitted, defaults to *False*.

- `KnockInLevel`: The barrier level used to determine whether a knock-in event has

occurred for the given knock-in observation date/s, upon which the equity amount is payable at the `KnockInPayDate`, subject to no knock-out event having occurred over the life of the contract.
Allowable values: Any number, as a percentage of the initial prices expressed in decimal form.

- `FloatingPeriodSchedule`: Period dates used for calculating floating coupon accrual fractions, and for observing underlying prices for calculating performances.
Allowable values: See section 19.3.4 Schedule Data and Dates/Rules, or DerivedSchedule for scripted trades (see the Event data structure in the scripted trade documentation at vre/docs/scripted_trade).

- `FloatingFixingSchedule` [Optional]: Fixing dates for floating coupons linked to Ibor indices. The d^{th} date in this schedule is the fixing date for the $(d+1)^{\text{th}}$ date in the `FloatingPayDates`. For floating coupons linked to overnight indices this scheulde is not relevant, the fixing dates are derived from the FloatingPeriodScheudle and FloatingLookback.
Allowable values: See section 19.3.4 Schedule Data and Dates/Rules, or DerivedSchedule for scripted trades (see the Event data structure in vre/docs/scripted_trade). If left blank or omitted, defaults to a `DerivedSchedule` with the `FloatingPeriodSchedule` set as the base schedule.

- `FixedDeterminationSchedule` [Optional]: If `AccruingFixedCoupons` is *False*, the observation dates on which a fixed trigger event is evaluated, upon which a fixed coupon payout is made. Otherwise, there is no payout. If `AccruingFixedCoupons` is *True*, these will be the dates on which fixed coupon amounts are calculated. The coupon amount is scaled by the number of fixed trigger days relative to the total number of days in the period.
Allowable values: See section 19.3.4 Schedule Data and Dates/Rules, or DerivedSchedule for scripted trades (see the Event data structure in vre/docs/scripted_trade). If left blank or omitted, defaults to a `DerivedSchedule` with the `FloatingPeriodSchedule` set as the base schedule.

- `KnockOutDeterminationSchedule` [Optional]: Dates on which a knock-out event is evaluated, after which the instrument is considered to be terminated and no payout is made.
Allowable values: See section 19.3.4 Schedule Data and Dates/Rules, or DerivedSchedule for scripted trades (see the Event data structure in vre/docs/scripted_trade). If left blank or omitted, defaults to a `DerivedSchedule` with the `FloatingPeriodSchedule` set as the base schedule.

- `KnockInDeterminationSchedule` [Optional]: Observation dates for determining if a knock-in event has occurred. If `BermudanKnockIn` is *True*, this node is required.
Allowable values: See section 19.3.4 Schedule Data and Dates/Rules, or DerivedSchedule for scripted trades (see the Event data structure in vre/docs/scripted_trade).

- `FixedAccrualSchedule` [Optional]: The fixed coupon paid out for each determination date in the `FixedDeterminationSchedule` is scaled according to the number of days in the `FixedAccrualSchedule` for which a fixed trigger event occurred, as a fraction of the total number of days in the period (defined by the

FixedDeterminationSchedule). If AccruingFixedCoupons is *True*, this node is required.
Allowable values: See section 19.3.4 Schedule Data and Dates/Rules, or DerivedSchedule for scripted trades (see the Event data structure in vre/docs/scripted_trade).

- FloatingPayDates: Floating coupon payment dates.
  Allowable values: See section 19.3.4 Schedule Data and Dates, Rules, or DerivedSchedule (see vre/docs/scripted_trade).
- FixedPayDates [Optional]: Fixed coupon payment dates.
  Allowable values: See section 19.3.4 Schedule Data and Dates/Rules, or DerivedSchedule for scripted trades (see the Event data structure in vre/docs/scripted_trade). If left blank or omitted, defaults to the FloatingPayDates.
- KnockInPayDate [Optional]: Settlement date for the knock-in payment, subject to a knock-in event having been triggered.
  Allowable values: See Date in Table 20. If left blank or omitted, this defaults to the last date in the FloatingPayDates schedule.
- KnockOutLevels: For each determination date in KnockOutDeterminationSchedule except for the first date, the barrier level used to determine whether a knock-out trigger event has occurred, resulting in a knock-out (i.e. contract termination).
  Allowable values: For each knock-out determination date, a KnockOutLevel sub-node with any positive number, as a percentage of the initial prices, expressed in decimal form.
- FixedTriggerLevels: For each fixed trigger determination date, the barrier level used to determine whether a coupon trigger event has occurred, upon which a fixed coupon is paid out at the corresponding payment date.
  Allowable values: For each fixed trigger evaluation date, a FixedTriggerLevel sub-node with any positive number, as a percentage of the initial prices expressed in decimal form.
- FixedRate: Rate of the fixed coupon leg.
  Allowable values: Any number, as a percentage expressed in decimal form.
- AccumulatingFixedCoupons [Optional]: Whether the fixed coupons are accumulating or not.
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36. If left blank or omitted, defaults to *False*.
- AccruingFixedCoupons [Optional]: Whether the fixed coupons are paid out on an accrual basis (*True*), or subject to a single fixed trigger event (given by the FixedDeterminationSchedule).
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36. If left blank or omitted, defaults to *False*.
- FloatingIndex: Underlying index of the floating leg.
  Allowable values: See vre/docs/scripted_trade (section Data Node / Index). This must be an InterestRate underlying. Overnight indices are not supported.

- `FloatingSpread` [Optional]: Spread on the floating rate.
  Allowable values: Any number. If left blank or omitted, defaults to zero.
- `FloatingDayCountFraction`: The day count fraction for the floating leg accrual calculations.
  Allowable values: See `DayCount Convention` in Table 25.
- `Strike` [Optional]: Strike price for for each underlying in the basket. When calculating the equity amount, this is the price used to calculate the number of shares (for Equity underlyings) owing of the worst-performing asset.
  Allowable values: Any number, as a percentage of the initial prices, expressed in decimal form. If left blank or omitted, defaults to one, i.e. the initial price of the underlying.
- `FloatingLookback` [Optional]: Only applicable when the floating leg reference rate is an overnight index. The shift applied to each value date in `FloatingFixingSchedule` to obtain the fixing reference date.
  Allowable values: Any period definition in unit "Days" (e.g. *2D*, *30D*, but not *1M*). If omitted or left blank, defaults to *0D*.
- `FloatingRateCutoff` [Optional]: Only applicable when the floating leg reference rate is an overnight index. The number of fixing dates at the end of the fixing period for which the fixing value is held constant and set to the previous value.
  Allowable values: Any non-negative whole number. If left blank or omitted, defaults to zero.
- `IsAveraged` [Optional]: Only applicable when the floating leg reference rate is an overnight index (where there are multiple fixings over a period). If *True*, the average of the fixings is used to calculate the coupon. If *False*, the coupon is calculated by compounding the fixings.
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36. Defaults to *false* If left blank or omiited.
- `IncludeSpread` [Optional]: Only applicable when the floating leg reference rate is an overnight index and if IsAveraged is false, i.e. the coupon rate is compounded. If *true* the spread is included in the compounding, otherwise it is excluded.
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.

Worst Of Basket Swaps can alternatively be represented as *scripted trades*, refer to vre/docs/scripted_trade for an introduction.

```
<TradeType>ScriptedTrade</TradeType>
<Envelope>
   ....
</Envelope>
<WorstOfBasketSwapData>
  <LongShort type="longShort">Long</LongShort>
  <Quantity type="number">1955000</Quantity>
  <InitialFixedRate type="number">0.015</InitialFixedRate>
  <Underlyings type="index">
    <Value>EQ-RIC:.STOXX50E</Value>
    <Value>EQ-RIC:.SPX</Value>
  </Underlyings>
  <InitialPrices type="number">
    <Value>3481.44</Value>
    <Value>3714.24</Value>
```

```
    </InitialPrices>
    <DeterminationDates type="event">
      <ScheduleData>
          ....
      </ScheduleData>
    </DeterminationDates>
    <SettlementDates type="event">
      <ScheduleData>
        <Dates>
          ....
        </Dates>
      </ScheduleData>
    </SettlementDates>
    <KnockOutLevels type="number">
      <Value>...</Value>
    </KnockOutLevels>
    <CouponTriggerLevels type="number">
      <Value>...</Value>
    </CouponTriggerLevels>
    <KnockInLevel type="number">0.75</KnockInLevel>
    <CouponRate type="number">0.04</CouponRate>
    <AccumulatingCoupons type="bool">true</AccumulatingCoupons>
    <FloatingIndex type="index">EUR-EURIBOR-3M</FloatingIndex>
    <FloatingSpread type="number">0.002</FloatingSpread>
    <FloatingDayCountFraction type="dayCounter">Actual/360</FloatingDayCountFraction>
    <FixingSchedule type="event">
      <DerivedSchedule>
          ....
      </DerivedSchedule>
    </FixingSchedule>
    <Strike type="number">1.0</Strike>
    <PayCcy type="currency">EUR</PayCcy>
  </WorstOfBasketSwapData>
```

The WorstOfBasketSwap script referenced in the trade above is shown in Listing 160.

The net cashflows under a (scripted) worst of basket swap from the long perspective are:

1. Receive an initial fixed amount: `InitialFixedRate` $*$ `Quantity`,
2. At each determination date (if a knock-out has not occurred for any previous determination date):
   - Pay a floating amount: `Quantity` $* floatingRate * accrualFraction$,
   - If a coupon trigger event occurs for this date, receive a fixed coupon amount (as well as any other fixed coupon amounts not previously received because a coupon trigger event had not occurred): `Quantity` $*$ `CouponRate`
3. On the final determination date, if $worstPerformance < \min(Strike, KnockInLevel)$, pay `Quantity` $*$(`Strike`$-worstPerformance$), where $worstPerformance$ is the performance, i.e. S_T/S_0, of the worst-performing asset as of the final determination date T.

The meanings and allowable values of the elements in the `WorstOfBasketSwap` node below.

- [longShort] `LongShort`: Own party position.
  Allowable values: *Long, Short*
- [number] `Quantity`: Quantity multiplier, or the equity notional amount.
  Allowable values: Any number.
- [number] `InitialFixedRate`: Rate of the initial fixed payment.
  Allowable values: Any number, as a percentage expressed in decimal form.

- [index] `Underlyings`: The basket of underlyings.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `InitialPrices`: The agreed initial price for each underlying in the basket.
  Allowable values: Any positive number. The number of `Value` sub-nodes should match the number of underlyings. This must have the same number of entries as `Underlyings`.
- [event] `DeterminationDates`: Floating leg period dates, knock-out determination dates, and fixed trigger determination dates. The first date is only used for calculating the floating leg accrual fraction, and corresponds to the contract effective date.
  Allowable values: See Section 19.3.4 ScheduleData, or DerivedSchedule (see vre/docs/scripted_trade) if derived from `SettlementDates` or `FixingSchedule`. The number of determination dates should match the number of settlement and fixing dates.
- [event] `SettlementDates`: The set of settlement dates corresponding to each determination date. While the first settlement date is unused in the calculations, this format simplifies the input, e.g. the settlement schedule can be a derived schedule, with the *DeterminationDates* as the base schedule.
  Allowable values: See section 19.3.4 ScheduleData, or DerivedSchedule (see vre/docs/scripted_trade) if derived from `DeterminationDates` or `FixingSchedule`. The number of settlement dates should match the number of determination and fixing dates.
- [number] `KnockOutLevels`: For each determination date, the barrier level used to determine whether a knock-out trigger event has occurred, resulting in a knock-out (i.e. contract termination) at the corresponding settlement date.
  Allowable values: Any number, as a percentage of the initial prices expressed in decimal form. This should have the same number of entries as the number of dates in `DeterminationDates` minus one.
- [number] `CouponTriggerLevels`: For each determination date, the barrier level used to determine whether a coupon trigger event has occurred, upon which a fixed coupon is paid out at the corresponding settlement date.
  Allowable values: Any number, as a percentage of the initial prices expressed in decimal form. This should have the same number of entries as the number of dates in `DeterminationDates` minus one.
- [number] `KnockInLevel`: For the final determination date, the barrier level used to determine whether a knock-in event has occurred, upon which the equity amount is payable at the final settlement date, subject to no knock-out event having occurred in the life of the contract.
  Allowable values: Any number, as a percentage of the initial price expressed in decimal form.
- [number] `CouponRate`: Rate of the fixed coupon leg.
  Allowable values: Any number, as a percentage expressed in decimal form.
- [bool] `AccumulatingCoupons`: Whether the fixed coupons are accumulating or not.
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.
- [index] `FloatingIndex`: Underlying index of the floating leg.

Allowable values: See vre/docs/scripted_trade's Index Section for allowable values. Overnight indices are not supported.

- [number] `FloatingSpread`: Spread on the floating rate.
  Allowable values: Any number.
- [dayCounter] `FloatingDayCountFraction`: The day count fraction for the floating leg accrual calculations.
  Allowable values: See `DayCount Convention` in Table 25.
- [event] `FixingSchedule`: The fixing schedule of the floating leg payments.
  Allowable values: See Section 19.3.4 ScheduleData, or DerivedSchedule (see vre/docs/scripted_trade) if derived from `DeterminationDates` or `SettlementDates`. The number of fixing dates should match the number of determination and settlement dates.
- [number] `Strike`: Strike price for each underlying in the basket. When calculating the equity amount, this is the price used to calculate the number of shares owing of the worst-performing asset.
  Allowable values: Any number, as a percentage of the initial prices, expressed in decimal form.
- [currency] `PayCcy`: Settlement currency. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 `Currency`.

19.2.79 Accumulators and Decumulators

The `FxAccumulatorData`, `EquityAccumulatorData`, `CommodityAccumulatorData` is the trade data container for the FxAccumulator, EquityAccumulator, CommodityAccumulator trade type. The following listings and show the structure of example trades for an FX and Equity underlying. Here the FX accumulator is of "type 01" meaning that a settlement takes place on each observation date while the equity accumulator is of "type 02" meaning that a settlement takes place on specific period end dates for all observation dates in that period.

```
<Trade id="FX_ACCUMULATOR">
  <TradeType>FxAccumulator</TradeType>
  <Envelope>
   ...
  </Envelope>
  <FxAccumulatorData>
    <Currency>USD</Currency>
    <FixingAmount>1000000</FixingAmount>
    <Strike>1.1</Strike>
    <Underlying>
      <Type>FX</Type>
      <Name>ECB-EUR-USD</Name>
    </Underlying>
    <OptionData>
      <LongShort>Long</LongShort>
      <PayoffType>Accumulator</PayoffType>
    </OptionData>
    <StartDate>2016-03-01</StartDate>
    <ObservationDates>
```

```
      <Dates>
        <Dates>
          <Date>2017-03-01</Date>
          <Date>2020-03-01</Date>
          <Date>2025-03-01</Date>
          <Date>2029-03-01</Date>
        </Dates>
      </Dates>
    </ObservationDates>
    <SettlementDates>
      <Dates>
        <Dates>
          <Date>2017-03-03</Date>
          <Date>2020-03-03</Date>
          <Date>2025-03-03</Date>
          <Date>2029-03-03</Date>
        </Dates>
     </Dates>
    </SettlementDates>
    <RangeBounds>
      <RangeBound>
        <RangeTo>1.14</RangeTo>
        <Leverage>1</Leverage>
      </RangeBound>
      <RangeBound>
        <RangeFrom>1.14</RangeFrom>
        <Leverage>1</Leverage>
      </RangeBound>
    </RangeBounds>
    <Barriers>
      <BarrierData>
        <Type>UpAndOut</Type>
        <Style>American</Style>
        <Levels>
          <Level>1.5</Level>
        </Levels>
      </BarrierData>
      <BarrierData>
        <Type>FixingFloor</Type>
        <Levels>
          <Level>2</Level>
        </Levels>
      </BarrierData>
    </Barriers>
  </FxAccumulatorData>
</Trade>
```

```
<Trade id="Equity_Decumulator">
  <TradeType>EquityAccumulator</TradeType>
  <Envelope>
   ...
  </Envelope>
  <EquityAccumulatorData>
    <FixingAmount>30</FixingAmount>
    <StrikeData>
      <Value>4000</Value>
      <Currency>EUR</Currency>
```

```
</StrikeData>
<Underlying>
    <Type>Equity</Type>
    <Name>.STOXX50</Name>
    <IdentifierType>RIC</IdentifierType>
</Underlying>
<OptionData>
  <LongShort>Long</LongShort>
  <PayoffType>Decumulator</PayoffType>
</OptionData>
<StartDate>20190925</StartDate>
<ObservationDates>
  <Rules>
    <StartDate>20190925</StartDate>
    <EndDate>20200925</EndDate>
    <Tenor>1D</Tenor>
    <Calendar>TARGET</Calendar>
    <Convention>F</Convention>
    <TermConvention>F</TermConvention>
    <Rule>Forward</Rule>
  </Rules>
</ObservationDates>
<PricingDates>
  <Dates>
    <Dates>
      <Date>20211025</Date>
      <Date>20211125</Date>
      ...
    </Dates>
  </Dates>
</PricingDates>
<SettlementLag>2D</SettlementLag>
<SettlementCalendar>TARGET</SettlementCalendar>
<SettlementConvention>F</SettlementConvention>
<RangeBounds>
  <RangeBound>
    <RangeTo>4000</RangeTo>
    <Leverage>1</Leverage>
  </RangeBound>
  <RangeBound>
    <RangeFrom>4000</RangeFrom>
    <Leverage>2</Leverage>
  </RangeBound>
</RangeBounds>
<Barriers>
  <BarrierData>
    <Type>DownAndOut</Type>
    <LevelData>
      <Level>
        <Value>3500</Value>
        <Currency>EUR</Currency>
      </Level>
    </LevelData>
  </BarrierData>
  <BarrierData>
    <Type>FixingFloor</Type>
    <Levels>
      <Level>1</Level>
```

```
        </Levels>
      </BarrierData>
    </Barriers>
    <KnockOutSettlementAtPeriodEnd>false<KnockOutSettlementAtPeriodEnd>
  </EquityAccumulatorData>
</Trade>
```

`Accumulator Payout Formula`

The payout formula, from the perspective of the party that is long, for each observation date of an Accumulator is:

$$Payout = RangeBound(Leverage) \cdot FixingAmount \cdot (fix - Strike),$$

i.e. for an Accumulator the holder pays [`Strike` * `FixingAmount` * `RangeBound`(Leverage)] expressed in `Currency` and receives/buys [fix * `FixingAmount` * `RangeBound`(Leverage)]'s worth of equity/CCY1/commodity in `Currency` on each observation date.

If `NakedOption` is *true*, the payout formula for an Accumulator (long perspective) is:

$$Payout = |RangeBound(Leverage)| \cdot FixingAmount \cdot \max(0, \phi \cdot (fix - Strike)),$$

where $\phi = \pm 1$ is the option type (+1 for Call, -1 for Put) inferred from the sign of the values of `RangeBound`(Leverage), which are all required to be the same.

`Decumulator Payout Formula`

The payout formula, from the perspective of the party that is long, for each observation date of a Decumulator is:

$$Payout = RangeBound(Leverage) \cdot FixingAmount \cdot (Strike - fix),$$

For a Decumulator the holder pays/sells [fix * `FixingAmount` * `RangeBound`(Leverage)]'s worth of equity/CCY1/commodity in `Currency`, and receives [`Strike` * `FixingAmount` * `RangeBound`(Leverage)] expressed in `Currency` on each observation date.

If `NakedOption` is *true*, the payout formula for a Decumulator (long perspective) is:

$$Payout = |RangeBound(Leverage)| \cdot FixingAmount \cdot \max(0, \phi \cdot (Strike - fix),$$

where $\phi = \pm 1$ is the option type (+1 for Call, -1 for Put) inferred from the sign of the values of `RangeBound`(Leverage), which are all required to be the same.

The meanings and allowable values of the elements in the data node follow below.

- StrikeData [Optional]: A node containing the global strike in `Value` and the currency in which both the underlying and the strike are quoted in `Currency`. Only supported for EquityAccumulators.

Allowable values: See `Currency` in Table 20. The strike may be any positive real number. The currency provided in this node may be quoted as corresponding minor currency to the underlying major currency. If omitted, local strikes should be used in each `RangeBound` node.

- Currency: The payout currency. The result of the payout formula above is treated to be in this currency. Note that for (non-quanto) FxAccumulators this should be the domestic (`CCY2`) currency. For non-quanto Equity- and CommodityAccumulators this should be the currency the equity or commodity is quoted in. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 `Currency`.

- FixingAmount: The unleveraged notional amount accumulated at each fixing date.

  For FxAccumulators: The FixingAmount is expressed in the foreign currency (`CCY1`).

  For EquityAccumulators: The FixingAmount is expressed as number of shares/units of the underlying equity or equity index.

  For CommodityAccumulators: The FixingAmount is expressed as number of units of the underlying commodity.

  Allowable values: Any real number. Note that a negative amount causes an Accumulator to become a Decumulator, and vice-versa.

- DailyFixingAmount [Optional]: For accumulator type "01" only: If *true*, the fixing amount for a period is calculated as the given FixingAmount times the number of calendar days in the period. The periods are given by [StartDate, ObservationDate(1)], [ObservationDate(1), ObservationDate(2)], ... etc. i.e. if *true*, a StartDate is required to determine the calendar days in the first period. If *false*, the fixing amount is used directly without any weighting.

  Allowable values: *true* or *false*. Defaults to *false* if left blank or omitted.

- Strike [Optional]: Global strike associated to the ranges. Is overwritten by local strikes defined in `RangeBounds` or modified by range strike adjustments. Note that each RangeBound needs a strike, either the global strike defined here, or a local one in the `RangeBounds` nodes which then overwrites the global one.

  For FxAccumulators: The the fx strike rate (global and local) is defined as amount in domestic currency (`CCY2`) for one unit of foreign currency (`CCY1`).

  For Equity- and CommodityAccumulators: The strike value (global and local) for one unit/share/contract of the underlying equity or commodity, expressed in the currency the equity or commodity is quoted or in any other pay currency. In case of a composite option style a FxIndex is required (see below). For EquityAccumulators, the `StrikeData` node (see above) should be used.

  The logic for global and local strikes is as follows:

  - if a local Strike for an interval is given, this local Strike will be used for the interval (a global Strike will be ignored if given)
  - otherwise if a global Strike and a local StrikeAdjustment are given, the strike used for the interval will be global Strike + local StrikeAdjustment

- otherwise if a global Strike, but no local StrikeAdjustment is given, the strike used for the interval will be the global Strike
- if no global strike and no local strike is given for an interval, an error is thrown, since the strike information is not sufficient

Allowable values: Any positive real number.

- Underlying: A node with the underlying of the Accumulator instrument.

  For FxAccumulators: `Type` is set to *FX* and `Name` is a string of the form *SOURCE-CCY1-CCY2* where `CCY1` is the foreign currency, `CCY2` is the domestic currency, and *SOURCE* is the fixing source, see Table 28 and 19.3.29.

  For EquityAccumulators: `Type` is set to *Equity* and `Name` and other fields are as outlined in 19.3.29

  For CommodityAccumulators: `Type` is set to *Commodity* and `Name` is an identifier of the commodity as outlined in 19.3.29 and in Table 32.

  Allowable values: Any FX, Equity or Commodity underlying as specified in 19.3.29

- OptionData: See 19.3.1. A node that describes whether the instrument is *Long* or *Short*, whether the payoff type is *Accumulator* or *Decumulator*,

  The relevant fields in the `OptionData` node for an Accumulator are:

  – `LongShort` The allowable values are *Long* or *Short*. Note that a *Long Accumulator* is equivalent to a *Short Decumulator* except when there are guaranteed fixings, i.e. when a Barrier node of Type *FixingFloor* is present. The *FixingFloor* causes asymmetry in the payoff.

    `PayoffType` The allowable values are *Accumulator* or *Decumulator*. For a long accumulator the strike is paid and the underlying received, for a long decumulator it is the other way around.

- StartDate [Mandatory for American Style Barrier or if a daily fixing amount is used, Optional for European Style or no Barrier and no daily fixing amount is used]: Used to set the start of the knock out monitoring - American knock out events are monitored from this date on. This field is only relevant for accumulators of type American Knock-Out, and not used otherwise. Can be omitted for European knock outs, or if there is no barrier.
  Allowable values: See `Date` in Table 20.

- ObservationDates: The observation dates given as schedule data.
  Allowable values: See the ScheduleData definition 19.3.4.

- PricingDates[Optional]: The dates defining observation period end dates on which the settlement for all dates in the period takes place. Note that the inclusion or not of PricingDates determines the Accumulator type. If included the Accumulator is of "type 02", and otherwise, if PricingDates are not included it is of "type 01".
  Allowable values: See the ScheduleData definition 19.3.4. Note that the final pricing period end date (defining the final observation period end date) must be on or after the final observation date.

- SettlementLag [Optional]: The settlement delay. Optional, if not given it is defaulted to 0D.
  Allowable values: Any period definition (e.g. *2D*, *1W*, *1M*, *1Y*)
- SettlementCalendar [Optional]: The calendar used to compute the settlement date from the corresponding observation date.
  Allowable values: Any valid calendar, see Table 24 Calendar.. Optional, defaults to the null calendar (no holidays).
- SettlementConvention [Optional]: The convention used to compute the settlement date from the corresponding observation date. Optional, defaults to F (following).
  Allowable values: Any valid roll convention (*U,F,P,MF,MP*), see Table 21 Roll Convention.
- SettlementDates [Optional]: The settlement dates can also be given as an explicit list of dates, see the FX accumulator listing above for an example. For a "type 01" accumulator the number of dates must be equal to the number of observation dates. For a "type 02" accumulator the number of dates must be equal to the number of pricing dates. If an explicit list of settlement dates is given, no settlement lag, calendar, conventions should be given.
- NakedOption [Optional]: If *true*, the payoff represents that of an option, and only positive values are accumulated in the instrument. The option type (Call or Put) is inferred from the sign of the `Leverage` values in `RangeBound`, which are all required to be the same.
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36. Defaults to *false* if left blank or omitted.
- RangeBounds: Nodes for the specification of ranges and associated leverages. A leverage can be specified to apply for all values the underlying can take, or for specific ranges using `RangeFrom` and `RangeTo`. Multiple `RangeBound` sub-nodes can be included within the `RangeBounds` node. If no leverage is specified for a range, it defaults to 1 for this range. In addition a range bound specific strike can be specified for Accumulators of "type 01" (and only this type!) which overwrites the global Strike field. If all range bounds have a specific strike defined, the global Strike field might be omitted.
  Allowable values: For each range, see 19.3.32. Only the `Leverage` is relevant for a given range. All `Leverage` parameters in one instrument must have the same sign. For "type 01" Accumulators, the Strike is relevant too and overwrites the global strike if given. Finally, for "type 01" Accumulators a range bound specific strike can be specified by specifying a range bound specific strike adjustment to the global strike (see the range bound spec for details).
- Barriers [Optional]: Specification of barriers and fixing floors (guaranteed fixings). Multiple `BarrierData` sub-nodes can be included within the `Barriers` node. Relevant fields for each `BarrierData` sub-node are `Type`, `Style`, and `Level`. The barrier is monitored on the
  - the observation dates (type 02 accumulator, i.e. PricingDates are given)
  - the observation dates (type 01 accumulator, i.e. no PricingDates are given), if barrier style is European

  - from the start date to the first observation date and between the observation dates on a continuous basis (type 01 accumulator), if barrier style is American

  For *type 01 accumulators* (no pricing dates are given), the FixingFloor guarantees a specific number of fixings to be settled even in case of a knock out. On a guaranteed fixing date, Only positive payouts (from the buyer/long perspective) are realised. Where payout = fix - strike for accumulators and strike - fix for decumulators.

  For *type 02 accumulators* (pricing dates are given), the FixingFloor specifies the number of *periods* (rather than observation dates) that are guaranteed to settle. If a knock out event occurs within a settlement period the fixings after the knock event within this period are assumed to fall into the first defined range and the settlement takes place on the knock out day plus the defined settlement delay.

  The barrier `Level` for `Type` *UpAndOut* and *DownAndOut* is expressed in the same way as `Strike` outlined above. For EquityAccumulators, each `Level` node should be provided a `Value` and a `Currency` and contained in a `LevelData` node (see example trade above). This `Currency` supports minor currencies.
  The barrier `Level` for `Type` *FixingFloor* is a non-negative integer.

  Allowable values: For each `BarrierData` node, see 19.3.31. For Accumulators/Decumulators, the following values for `Type` are relevant: *UpAndOut*, *DownAndOut* and *FixingFloor*. The barrier `Style` can be *European* (monitored on observation dates) or *American* (continuously monitored).

- KnockOutSettlementAtPeriodEnd [Optional]: Only relevant for *type 02 accumulator*. Controlls the settlement behavior in case of an knock out events. If *true* the settlement of a knock out event takes place at the end of the observation period, otherwise on the knock out event date plus the defined settlement delay. Defaults to *false*.
- KnockOutFixingAtKOSettlement [Optional]: Only relevant for *type 02 accumulator*. Controlls the settlement behavior in case of an knock out events. If *true* the fixing at knockout settlement date is used to compute the knockout flow, otherwise the fixing at knockout date. Defaults to *false*.
- FxIndex [Optional]: Required for composite accumulators of type 2, where the strike, knockout barrier and pay currency are different from the underlying quote currency. Intended for Commodity and Equity Accumulators. A node with the underlying of the FX conversion from quote currency to pay currency. `Type` is set to *FX* and `Name` is a string of the form *SOURCE-CCY1-CCY2* where `CCY1` is the foreign (quote) currency, `CCY2` is the domestic (pay) currency, and *SOURCE* is the fixing source, see Table 28 and 19.3.29.

Accumulators can also be represented as scripted trades, refer to the separate documentation in vre/docs/scripted_trade for an introduction. Listing 161 shows the structure of an Accumulator (type 01) example, here on a FX underlying (EQ or COMM underlyings are possible as well).

Listing 161: Accumulator (type 01) Scripted Representation

```
<Trade id="SCRIPTED_FX_ACCUMULATOR">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <Accumulator01Data>
    <Strike type="number">1.1</Strike>
    <FixingAmount type="number">1000000</FixingAmount>
    <LongShort type="longShort">Long</LongShort>
    <Underlying type="index">FX-ECB-EUR-USD</Underlying>
    <PayCcy type="currency">USD</PayCcy>
    <StartDate type="event">2016-03-01</StartDate>
    <FixingDates type="event">
      <ScheduleData>
        <Dates>
          <Dates>
            <Date>2017-03-01</Date>
            <Date>2020-03-01</Date>
            <Date>2025-03-01</Date>
            <Date>2029-03-01</Date>
          </Dates>
        </Dates>
      </ScheduleData>
    </FixingDates>
    <SettlementDates type="event">
      <DerivedSchedule>
        <BaseSchedule>FixingDates</BaseSchedule>
        <Shift>2D</Shift>
        <Calendar>TARGET</Calendar>
        <Convention>F</Convention>
      </DerivedSchedule>
    </SettlementDates>
    <RangeUpperBounds type="number">
      <Value>1.14</Value>
      <Value>10000</Value>
    </RangeUpperBounds>
    <RangeLowerBounds type="number">
      <Value>0</Value>
      <Value>1.14</Value>
    </RangeLowerBounds>
    <RangeLeverages type="number">
      <Value>1</Value>
      <Value>2</Value>
    </RangeLeverages>
    <KnockOutLevel type="number">1.5</KnockOutLevel>
    <KnockOutType type="barrierType">UpOut</KnockOutType>
    <AmericanKO type="bool">true</AmericanKO>
    <GuaranteedFixings type="number">2</GuaranteedFixings>
  </Accumulator01Data>
</Trade>
```

The meanings and allowable values of the elements in the `Accumulator01Data` representation follow below.

- Strike: The strike value the bought currency is purchased at.
- FixedAmount: The unleveraged notional amount accumulated at each fixing date
- LongShort: 1 for a long, -1 for a short position
- Underlying: See vre/docs/scripted_trade's Index section for allowable values.
- PayCcy: The payment currency of the trade. Notice section Notice section "Payment Currency" in vre/docs/scripted_trade.
- StartDate: The start date. American knock out events are monitored from this date on. Notice that the start date must be given in the scripted trade representation for European knock outs, although it is not used for this variant.
- FixingDates: The fixing dates, given as a ScheduleData, or DerivedSchedule (see vre/docs/scripted_trade).
- SettlementDates: The fixing dates, given as a ScheduleData, or DerivedSchedule (see vre/docs/scripted_trade).
- RangeUpperBound: Values of upperbounds for the leverage ranges. If a given range has no upperbound add 100000
- RangeLowerBound: Values of lowerbounds for the leverage ranges. If a given range has no lowerbound add 0
- RangeLeverages: Values of leverages for the leverage ranges.
- KnockOutLevel: The KnockOut Barrier level
- KnockOutType: The KnockOut Barrier type, can be UpOut or DownOut
- AmericanKO: If true, knock out events are monitored on a continuous basis, otherwise they are monitored on the fixing dates only.
- GuaranteedFixings: Number of the first n Fixings that are guaranteed, regardless of whether or not the trade has been knocked out.

The script 'Accumulator01' referenced in the trade above is shown in Listing 162.

Listing 162: Accumulator type 01 script

```
REQUIRE KnockOutType == 3 OR KnockOutType == 4;
NUMBER Payoff, fix, d, r, Alive, currentNotional, Factor, ThisPayout, Fixing[SIZE(FixingDates)];
Alive = 1;
FOR d IN (1, SIZE(FixingDates), 1) DO
    fix = Underlying(FixingDates[d]);
    Fixing[d] = fix;

    IF AmericanKO == 1 THEN
      IF KnockOutType == 4 THEN
        IF FixingDates[d] >= StartDate THEN
           IF d == 1 OR FixingDates[d-1] <= StartDate THEN
              Alive = Alive * (1 - ABOVEPROB(Underlying, StartDate, FixingDates[d], KnockOutLevel));
           ELSE
              Alive = Alive * (1 - ABOVEPROB(Underlying, FixingDates[d-1], FixingDates[d], KnockOutLevel));
           END;
        END;
      ELSE
        IF FixingDates[d] >= StartDate THEN
           IF d == 1 OR FixingDates[d-1] <= StartDate THEN
              Alive = Alive * (1 - BELOWPROB(Underlying, StartDate, FixingDates[d], KnockOutLevel));
           ELSE
              Alive = Alive * (1 - BELOWPROB(Underlying, FixingDates[d-1], FixingDates[d], KnockOutLevel));
           END;
        END;
      END;
    ELSE
      IF {KnockOutType == 4 AND fix >= KnockOutLevel} OR
         {KnockOutType == 3 AND fix <= KnockOutLevel} THEN
        Alive = 0;
      END;
    END;

    IF d <= GuaranteedFixings THEN
      Factor = 1;
    ELSE
      Factor = Alive;
    END;

    FOR r IN (1, SIZE(RangeUpperBounds), 1) DO
      IF fix > RangeLowerBounds[r] AND fix <= RangeUpperBounds[r] THEN
        ThisPayout = RangeLeverages[r] * FixingAmount * (fix - Strike) * Factor;
        IF d > GuaranteedFixings OR ThisPayout >= 0 THEN
          Payoff = Payoff + LOGPAY(RangeLeverages[r] * FixingAmount * (fix - Strike) * Factor,
                                   FixingDates[d], SettlementDates[d], PayCcy);
        END;
      END;
    END;
END;
value = LongShort * Payoff;
currentNotional = FixingAmount * Strike;
```

Accumulators of Type 02 can also be represented as scripted trades, see vre/docs/scripted_trade for an introduction. Listing 163 shows the structure of an Accumulator (Type 02) example, here on an equity underlying. FX and COMM underlyings are possible as well.

Listing 163: EQ Accumulator (type 02) Scripted Representation

```
<Trade id="EqDecumulator">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <Accumulator02Data>
    <Strike type="number">59.9842</Strike>
    <FixingAmount type="number">11</FixingAmount>
    <LongShort type="longShort">Long</LongShort>
    <Underlying type="index">EQ-RIC:.SPX</Underlying>
    <PayCcy type="currency">EUR</PayCcy>
    <ObservationDates type="event">
      <ScheduleData>
        <Rules>
          <StartDate>20190925</StartDate>
          <EndDate>20200925</EndDate>
          <Tenor>1D</Tenor>
          ...
        </Rules>
      </ScheduleData>
    </ObservationDates>
    <KnockOutSettlementDates type="event">
      <DerivedSchedule>
        <BaseSchedule>ObservationDates</BaseSchedule>
        <Shift>2D</Shift>
        <Calendar>TARGET</Calendar>
        <Convention>F</Convention>
      </DerivedSchedule>
    </KnockOutSettlementDates>
    <ObservationPeriodEndDates type="event">
      <ScheduleData>
        <Dates>
          <Dates>
            <Date>20191025</Date>
            <Date>20191125</Date>
            ...
            <Date>20200925</Date>
          </Dates>
        </Dates>
      </ScheduleData>
    </ObservationPeriodEndDates>
    <SettlementDates type="event">
      <ScheduleData>
        <Dates>
          <Dates>
            <Date>20191027</Date>
            <Date>20191127</Date>
            ...
            <Date>20200927</Date>
          </Dates>
        </Dates>
      </ScheduleData>
    </SettlementDates>
    <RangeUpperBounds type="number">
      <Value>59.9842</Value>
      <Value>10000000.0</Value>
    </RangeUpperBounds>
    <RangeLowerBounds type="number">
      <Value>-10000000.0</Value>
      <Value>59.9842</Value>
```

The script ‘Accumulator02’ referenced in the trade above is shown in Listing 164.

The meanings and allowable values of the elements in the `DecumulatorData` node folllow below.

- Strike: The forward price. For an FX underlying `FX-SOURCE-CCY1-CCY2` this is the number of units of `CCY2` per units of `CCY1`. For an EQ underlying this is the equity price expressed in the equity ccy. Allowable values are non-negative numbers.
- FixingAmount: The number of shares per day (EQ Undlerinyg) resp. the foreign amount (FX Undderlying). Allowable values are non-negative values.
- LongShort: The position, allowable values are “Long” and “Short”
- Underlying: The underlying index
  See vre/docs/scripted_trade’s Index section for allowable values.
- PayCcy: The payment currency. See the appendix for allowable currency codes. Notice section Notice section “Payment Currency” in vre/docs/scripted_trade.
- ObservationDates: The observation date schedule. See vre/docs/scripted_trade on how this is set up.
- KnowOutSettlementDates: The settlement dates associated to the observation dates in case of a knock out event, the number of observation and knock out settlement dates must be equal. See vre/docs/scripted_trade on how this is set up.
- ObservationPeriodEndDates: The last date for each observation period. See vre/docs/scripted_trade on how this is set up.
- SettlementDates: The settlement dates for each observation period, the number of settlement dates and the number of observation period end dates must be equal. See vre/docs/scripted_trade on how this is set up.
- RangeUpperBounds: The multiplier for the “number of days below” in the payoff. Allowable values are non-negative numbers.
- RangeLowerBounds: The multiplier for the “number of days above” in the payoff. Allowable values are non-negative numbers.
- RangeLeverages: The multiplier for the defined ranges. Allowable values are non-negative numbers.
- DefaultRange: The default range used in case of a knock-out event to classify the remaining days until the GuaranteedPeriodEndDate. Allowable values are $1, 2, \ldots, r$ where r is the number of defined ranges.
- KnockOutLevel: The knock out price. See Strike for more details and allowable values.
- KnockOutType: The KnockOut Barrier type, can be UpOut or DownOut
- GuaranteedPeriodEndDate: The last date of the guaranteed period. Allowable values are any valid dates.

Listing 164: Accumulator type 02 script.

```
REQUIRE SIZE(ObservationDates) == SIZE(KnockOutSettlementDates);
REQUIRE SIZE(ObservationPeriodEndDates) == SIZE(SettlementDates);
REQUIRE SIZE(RangeUpperBounds) == SIZE(RangeLowerBounds);
REQUIRE SIZE(RangeUpperBounds) == SIZE(RangeLeverages);
NUMBER Payoff, fix, d, dd, KnockedOut, currentNotional, Days[SIZE(RangeUpperBounds)];
NUMBER currentPeriod, r;
currentPeriod = 1;
FOR d IN (1, SIZE(ObservationDates)) DO
  fix = Underlying(ObservationDates[d]);
  IF KnockedOut == 0 THEN
    IF {KnockOutType == 4 AND fix >= KnockOutLevel} OR
       {KnockOutType == 3 AND fix <= KnockOutLevel} THEN
       KnockedOut = 1;
       FOR dd IN (d + 1, SIZE(ObservationDates)) DO
         IF ObservationDates[d] <= GuaranteedPeriodEndDate THEN
            Days[DefaultRange] = Days[DefaultRange] + 1;
         END;
       END;
       FOR r IN (1, SIZE(RangeUpperBounds)) DO
         value = value + PAY( LongShort * FixingAmount * RangeLeverages[r] * Days[r]
                              * ( fix - Strike ),
                              ObservationDates[d], KnockOutSettlementDates[d], PayCcy );
       END;
    END;
  END;
  IF KnockedOut == 0 THEN
    FOR r IN (1, SIZE(RangeUpperBounds)) DO
      IF fix > RangeLowerBounds[r] AND fix <= RangeUpperBounds[r] THEN
        Days[r] = Days[r] + 1;
      END;
    END;
    IF ObservationDates[d] >= ObservationPeriodEndDates[currentPeriod] THEN
      FOR r IN (1, SIZE(RangeUpperBounds)) DO
        value = value + PAY( LongShort * FixingAmount * RangeLeverages[r] * Days[r]
                             * ( fix - Strike ),
                             ObservationDates[d], SettlementDates[currentPeriod], PayCcy );
      END;
    END;
  END;
  IF ObservationDates[d] >= ObservationPeriodEndDates[currentPeriod] THEN
    currentPeriod = currentPeriod + 1;
    IF currentNotional == 0 THEN
      FOR r IN (1, SIZE(RangeUpperBounds)) DO
        currentNotional = currentNotional + FixingAmount * RangeLeverages[r] * Days[r] * Strike;
      END;
    END;
    FOR r IN (1, SIZE(RangeUpperBounds)) DO
      Days[r] = 0;
    END;
  END;
END;
```

19.2.80 Target Redemption Forward (TaRF)

The `FxTaRFData`, `EquityTaRFData`, `CommodityTaRFData` is the trade data container for the FxTaRF, EquityTaRF, CommodityTaRF trade type, listings 165 and 166 show the structure of example trades for an FX and Equity underlying.

Listing 165: FxTaRF data

```
<Trade id="FX_TARF">
  <TradeType>FxTaRF</TradeType>
  <Envelope/>
  <FxTaRFData>
    <Currency>USD</Currency>
    <FixingAmount>1000000</FixingAmount>
    <TargetAmount>100000</TargetAmount>
    <Strike>1.1</Strike>
    <Underlying>
      <Type>FX</Type>
      <Name>ECB-EUR-USD</Name>
    </Underlying>
    <ScheduleData>
      <Dates>
        <Dates>
          <Date>2017-03-01</Date>
          <Date>2020-03-01</Date>
          <Date>2025-03-01</Date>
          <Date>2029-03-01</Date>
        </Dates>
      </Dates>
    </ScheduleData>
    <OptionData>
      <LongShort>Long</LongShort>
      <PayoffType>TargetExact</PayoffType>
    </OptionData>
    <RangeBounds>
      <RangeBound>
        <RangeTo>1.05</RangeTo>
        <Leverage>2</Leverage>
      </RangeBound>
      <RangeBound>
        <RangeFrom>1.1</RangeFrom>
        <Leverage>1</Leverage>
      </RangeBound>
    </RangeBounds>
    <Barriers>
      <BarrierData>
        <Type>CumulatedProfitCap</Type>
        <Levels>
          <Level>100000</Level>
        </Levels>
      </BarrierData>
    </Barriers>
  </FxTaRFData>
</Trade>
```

Listing 166: EquityTaRF data

```
<Trade id="EQ_TARF">
  <TradeType>EquityTaRF</TradeType>
  <Envelope/>
  <EquityTaRFData>
    <Currency>USD</Currency>
    <FixingAmount>775</FixingAmount>
    <Strike>2900</Strike>
    <Underlying>
      <Type>Equity</Type>
      <Name>RIC:.SPX</Name>
    </Underlying>
    <ScheduleData>
      <Dates>
        <Dates>
          <Date>2017-03-01</Date>
          <Date>2020-03-01</Date>
          <Date>2025-03-01</Date>
          <Date>2026-03-01</Date>
          <Date>2027-03-01</Date>
          <Date>2028-03-01</Date>
        </Dates>
      </Dates>
    </ScheduleData>
    <OptionData>
      <LongShort>Long</LongShort>
      <PayoffType>TargetFull</PayoffType>
    </OptionData>
    <RangeBounds>
        <RangeBound>
            <RangeTo>3300</RangeTo>
            <Leverage>1</Leverage>
        </RangeBound>
        <RangeBound>
            <RangeFrom>3300</RangeFrom>
            <Leverage>2</Leverage>
            <StrikeAdjustment>100</StrikeAdjustment>
        </RangeBound>
    </RangeBounds>
    <Barriers>
      <BarrierData>
        <Type>FixingCap</Type>
        <Levels>
          <Level>3</Level>
        </Levels>
      </BarrierData>
    </Barriers>
  </EquityTaRFData>
</Trade>
```

The payout formula for each fixing date of a TaRF - for a Long position - is:
$Payout = RangeBound(Leverage) \cdot FixingAmount \cdot (fix - Strike(global/local))$

The meanings and allowable values of the elements in the data node follow below.

- Currency: The payout currency of the TaRF. The TargetAmount and the result of the

payout formula above are expressed in this currency. For FX, this should be `CCY2` as defined in the Name field in the Underlying node. Note that for (non-quanto) FxTaRFs this should be the domestic (`CCY2`) currency, as defined in the Name field in the Underlying node. For non-quanto Equity- and CommodityTaRFs this should be the currency the equity or commodity is quoted in. Notice section "Payment Currency" in vre/docs/scripted_trade.
Allowable values: See Table 22 `Currency`.

- FixingAmount: The unlevered FixingAmount amount to be used at each fixing date, as per the payout formula above.

  For FxTaRFs: The FixingAmount is expressed in the foreign currency (`CCY1`).

  For EquityTaRFs: The FixingAmount is expressed as number of shares/units of the underlying equity or equity index.

  For CommodityTaRFs: The FixingAmount is expressed as number of units of the underlying commodity.

  Allowable values: Any positive real number. Note that the FixingAmount must be positive both for Long and Short positions.

- TargetAmount[Optional]: If PayoffType = *TargetExact*, after a knock-out event the last payout is adjusted so that the cumulated profit since the start of the TaRF matches the target amount. TargetAmount is optional, only required for PayoffType = *TargetExact*. The TargetAmount is a currency amount, for FxTaRFs it is in (`CCY2`).

  Notice that if the barrier type is *CumulatedProfitCapPoints* the last payout is adjusted in a way such that the cumulated profit in relative terms (as defined for this barrier type below) matches the target amount divided by the fixing amount (i.e. the target amount in points).
  Allowable values: Any positive real number.

- TargetPoints[Optional]: For PayoffType = *TargetExact*, this provides an alternative way of expressing the TargetAmount as a percentage of FixingAmount in decimal form. For example a FixingAmount = $1,000,000$ and TargetPoints = 0.0700 translates to a TargetAmount = $70,000$.
  Allowable values: Any positive real number.

- Strikes [Optional]: Global strikes associated to the ranges. Either this node or the Strike node must be given to specify (a) global strike(s) valid across all ranges. If no such global strikes are given, the strikes have to be specified as part of the range bound definition. The Strikes node represents time-varying strikes while the Strike node represents a strike that is constant over time. If the Strikes node is given, it must contain a number of Strike subnodes. Either

  – the Strike subnodes except the first one must contain an attribute startDate which indicates from which fixing date on the strike is valid. The first strike is valid until the minimum startDate given in the other nodes, or

  – the values in the Strike subnodes are associated to the given fixing dates in their order, the number of values must be equal to the number of fixing dates or less, in the latter case the last given value is repeated to match the number of fixing dates.

See the description under Strike for further information on the single strike values and the relation to strike information given in range bounds.

- Strike [Optional]: Global strike associated to the ranges. Is overwritten by local strikes defined in `RangeBounds` or modified by range strike adjustments. Note that each RangeBound needs a strike, either the global strike defined here, or a local one in the `RangeBounds` nodes which then overwrites the global one. For FX, the strike is expressed as amount in `CCY2` per one unit of `CCY1` as defined in the Name field in the Underlying node.

  The logic for global and local strikes is as follows:

  - if a local Strike for a range is given, this local Strike will be used for the range (a global Strike will be ignored if given)
  - otherwise if a global Strike and a local StrikeAdjustment are given, the strike used for the range will be global Strike + local StrikeAdjustment
  - otherwise if a global Strike, but no local StrikeAdjustment is given, the strike used for the range will be the global Strike
  - if no global strike and no local strike is given for an range, an error is thrown, since the strike information is not sufficient

  Allowable values: Any positive real number.

- Underlying: A node with the underlying of the TaRF instrument.

  For FxTaRFs: `Type` is set to *FX* and `Name` is a string on the form *SOURCE-CCY1-CCY2* where `CCY1` is the foreign currency, `CCY2` is the domestic currency, and *SOURCE* is the fixing source, see Table 28 and 19.3.29.

  For EquityTaRFs: `Type` is set to *Equity* and `Name` and other fields are as outlined in 19.3.29

  For CommodityTaRFs: `Type` is set to *Commodity* and `Name` is an identifier of the commodity as outlined in 19.3.29 and in Table 32.

  Allowable values: Any FX, Equity or Commodity underlying as specified in 19.3.29

- ScheduleData: The fixing dates schedule of the TaRF.
  Allowable values: See 19.3.4

- SettlementLag [Optional]: The settlement delay.
  Allowable values: Any period definition (e.g. *2D*, *1W*, *1M*, *1Y*). If omitted or left blank, defaults to *0D*.

- SettlementCalendar [Optional]: The calendar used to compute the settlement date from the corresponding fixing date.
  Allowable values: Any valid calendar, see Table 24 Calendar. If omitted or left blank, defaults to the *NullCalendar* (no holidays).

- SettlementConvention [Optional]: The convention used to compute the settlement date from the corresponding fixing date.
  Allowable values: Any valid roll convention (*U,F,P,MF,MP*), see Table 21 Roll Convention. If omitted or left blank, defaults to *F* (following).

- OptionData: This is a trade component sub-node outlined in section 19.3.1. The relevant fields in the `OptionData` node for an TaRF are:
  - `LongShort` The allowable values are *Long* or *Short*.
  - `PayoffType` can be *TargetFull*, *TargetExact*, or *TargetTruncated*, impacting the payment after a Knock-Out event as per:

    TargetExact: the payment is set to the pre-determined `TargetAmount`
    TargetFull: the unadjusted payment is made
    TargetTruncated: no payment is made

- RangeBoundSet: Time-varying range-bound information. Either this node or the RangeBounds node (for time-independent range bounds) must be given. If the RangeBoundSet node is given, it must contain a number of RangeBounds nodes. Either
  - the RangeBounds subnodes except the first one must contain an attribute startDate which indicates from which fixing date on the values are valid. The first value is valid until the minimum startDate given in the other nodes, or
  - the values in the RangeBounds subnodes are associated to the given fixing dates in their order, the number of RangeBounds subnodes must be equal to the number of fixing dates or less, in the latter case the last given RangeBounds subnode is repeated to match the number of fixing dates.

  See the description under RangeBounds for further information on the single RangeBounds nodes.

- RangeBounds: Nodes for the specification of ranges (intervals) and associated `Leverage`, local `Strike` and local `StrikeAdjustment`. A range/interval can be specified to apply for all values the underlying can take, or for specific ranges using `RangeFrom` and `RangeTo`. Multiple `RangeBound` sub-nodes can be included within the `RangeBounds` node. If the value for `RangeFrom` is omitted it is interpreted as $-\infty$ (unbounded towards minus infinity). If `RangeTo` is omitted it is interpreted as ∞ (unbounded towards plus infinity). Note that if the underlying takes a value not covered by any range/interval, no payout will occur.

  For FxTaRFs: The the `Strike`, `StrikeAdjustment`, `RangeFrom` and `RangeTo` Fx rates are all defined as amount in domestic currency (`CCY2`) for one unit of foreign currency (`CCY1`).

  For Equity- and CommodityTaRFs: The `Strike`, `StrikeAdjustment`, `RangeFrom` and `RangeTo` are all defined as the value of one unit/share/contract of the underlying equity or commodity, expressed in the currency the equity or commodity is quoted in. Allowable values: For each range, see 19.3.32. Note that an interval/range can't have both a local `Strike` and a local `StrikeAdjustment`. If no global `Strike` is given, each interval/range must have a local `Strike`.

- Barriers: Specification of barriers, see 19.3.31. Multiple `BarrierData` sub-nodes can be included within the `Barriers` node. Relevant fields for each TaRF `BarrierData` sub-node are `Type` and `Level`. For a TaRF, `Type` can be set to *CumulatedProfitCap*, *CumulatedProfitCapPoints* or *FixingCap*. Notice that *CumulatedProfitCapPoints* can not be combined with the other barrier types.

– *CumulatedProfitCap*: The TaRF terminates once the generated profit from the long side's view reaches the CumulatedProfitCap. (Continuous TaRF) For an FxTaRF the CumulatedProfitCap is quoted in `CCY2`.
Allowable values: The `Level` for a *CumulatedProfitCap* can be set to any non-negative real number.

– *CumulatedProfitCapPoints*: The TaRF terminates once the generated profit from the long side's view reaches the CumulatedProfitCapPoints. (Continuous TaRF) Here, the generated profit is measured as the absolute profit amount (as used in CumulatedProfitCap) divided by the FixingAmount and divided by the absolute value of the leverage.
Allowable values: The `Level` for a *CumulatedProfitCapPoints* can be set to any non-negative real number.

– *FixingCap*: The TaRF terminates once number of fixings where a profit is generated - from the long side's view -reaches the FixingCap. (Digital TaRF)
Allowable values: The `Level` for a *FixingCap* can be set to a non-negative integer.

Target Redemption Forwards can alternatively represented as *scripted trades*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction. This representation does not allow for time varying strikes or range bounds and also not for a target amount / cumulated profit cap specification in points.

```
<Trade id="TaRF#1">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope/>
  <TaRFData>
    <FixingAmount type="number">1000000</FixingAmount>
    <LongShort type="longShort">Long</LongShort>
    <Underlying type="index">FX-ECB-EUR-USD</Underlying>
    <PayCcy type="currency">USD</PayCcy>
    <FixingDates type="event">
      <ScheduleData>
        <Dates>
          <Dates>
              <Date>2017-03-01</Date>
              <Date>2020-03-01</Date>
              <Date>2025-03-01</Date>
              <Date>2029-03-01</Date>
          </Dates>
        </Dates>
      </ScheduleData>
    </FixingDates>
    <SettlementDates type="event">
      <DerivedSchedule>
        <BaseSchedule>FixingDates</BaseSchedule>
        <Shift>2D</Shift>
        <Calendar>TARGET</Calendar>
        <Convention>F</Convention>
      </DerivedSchedule>
    </SettlementDates>
    <RangeUpperBounds type="number">
        <Value>1.05</Value>
        <Value>10000</Value>
    </RangeUpperBounds>
```

```
    <RangeLowerBounds type="number">
        <Value>0</Value>
        <Value>1.1</Value>
    </RangeLowerBounds>
    <RangeLeverages type="number">
        <Value>2</Value>
        <Value>1</Value>
    </RangeLeverages>
    <RangeStrikes type="number">
      <Value>1.1</Value>
      <Value>1.1</Value>
    </RangeStrikes>
    <KnockOutProfitAmount type="number">100000</KnockOutProfitAmount>
    <KnockOutProfitEvents type="number">0</KnockOutProfitEvents>
    <TargetAmount type="number">100000</TargetAmount>
    <TargetType type="number">0</TargetType>
  </TaRFData>
</Trade>
```

The TaRF script referenced in the trade above is shown in Listing 167.

The meanings and allowable values of the elements in the `TaRFData` node below.

- [number] `FixingAmount`: Amount to buy/sell on each fixing day should be FixingAmount times Leverage of the range that the spot level fixes within.
  Allowable values: Any positive real number.
- [longShort] `LongShort`: *Long* if we buy the option. *Short* if we sell the option.
  Allowable values: *Long, Short*
- [index] `Underlying`: Underlying index.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.
- [event] `FixingDates`: The fixing dates, given as a ScheduleData node
  Allowable values: See section 19.3.4 Schedule Data and Dates, or DerivedSchedule (see the scripted trade documentation in vre/docs/scripted_trade).
- [event] `SettlementDates`: The settlement dates, given as a ScheduleData or DerivedSchedule node. Allowable values: See section 19.3.4 Schedule Data and Dates, or DerivedSchedule (see the scripted trade documentation in vre/docs/scripted_trade).
- [number] `RangeUpperBound`: Values of upperbounds for the leverage ranges. If a given range has no upperbound, add 100000.
  Allowable values: Any list of positive real numbers. *RangeUpperBound*, *RangeLowerBound* and *RangeLeverages* must have the same size.
  $RangeUpperBound[i] \geq RangeLowerBound[i]$ for all i
- [number] `RangeLowerBound`: Values of lowerbounds for the leverage ranges. If a given range has no lowerbound add 0
  Allowable values: Any list of positive real numbers.

- [number] `RangeLeverages`: Values of leverages for the leverage ranges. Positive for Bullish TaRF, Negative for Bearish TaRF.
  Allowable values: Any list of real numbers.
- [number] `RangeStrikes`: Strikes associated to the ranges. .
  Allowable values: Any positive real number.
- [number] `KnockOutProfitAmount`: If positive, cap of the profit the buyer can make.
  Allowable values: Any positive real number or zero if not applicable.
- [number] `KnockOutProfitEvents`: If positive, cap of the number of fixing days on which buyer makes profit.
  Allowable values: Any positive real number or zero if not applicable.
- [number] `TargetAmount`: The target amount to be paid as an accumulated profit for TargetType = exact.
  Allowable values: Any positive real number or zero if not applicable.
- [number] `TargetType`: *-1* for truncated, *0* for exact, *1* for full. If DigitalKnockOut is true the target type must be set to full.
  Allowable values: *-1,0,1*

19.2.81 Knock Out Swap

A Knock Out Swap refers to a vanilla fixed vs. float Interest Rate Swap that terminates when the float index fixing is above ("up and out") or below ("down and out") a given barrier level.

A Knock Out Swap is represented using the TradeType *KnockOutSwap* and a KnockOutSwapData block as shown in listing 168. It must have one `BarrierData` node, and two legs, one fixed and one floating, each represented by a `LegData` trade component.

A Knock Out Swap is a Swap with one Fixed and one Floating leg, where the Swap is terminated if the Floating leg Index hits a barrier. The barrier is monitored on all floating leg fixing dates after the BarrierStartDate.

The meanings and allowable values in this block are as follows:

- BarrierData: This node specifies the barrier Type and Level. The Type must be *UpAndOut* or *DownAndOut*. Exactly one Level must be specified. All other values of the barrier data block are not relevant.
  Allowable values: See 19.3.31.
- BarrierStartDate: The barrier is monitored on all floating leg fixing dates that are on or after the barrier start date.
  Allowable values: See `Date` in Table 20.
- LegData: This specifies the swap terms. Exactly two LegData nodes must be given, one of type Fixed and one of type Floating.
  Allowable values: See 19.3.3.

19.2.82 Rainbow Options

Rainbow options are European calls or puts on the maximum or mimimum of of a range of assets. We denote the prices of n assets at expiry $S_1, S_2, \ldots, S_n$ and initial prices

$S_1^0, S_2^2, \ldots, S_n^0$ with respective constant weights $w_1, \ldots, w_n$. The payoff at expiry of the rainbow options covered here is

Rainbow Options are represented as traditional trades or *scripted trades*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction of the latter. Each of the supported variations, all European style, is represented by a separate payoff script as shown in Table 90 (excluding Rainbow Call Spread Options and Worst Performance Rainbow Options).

Rainbow Option Payoff	Payoff Script Name
$\max(w_1 S_1, \ldots, w_n S_n, K)$	BestOfAssetOrCashRainbowOption
$\min(w_1 S_1, \ldots, w_n S_n, K)$	WorstOfAssetOrCashRainbowOption
$\max(\omega(\max(w_1 S_1, \ldots, w_n S_n) - K, 0)$	MaxRainbowOption
$\max(\omega(\min(w_1 S_1, \ldots, w_n S_n) - K, 0)$	MinRainbowOption

Table 90: Rainbow option types and associated script names. $\omega = \pm 1$ distinguishes call (+1) and put (-1).

The supported underlying types are Equity, Fx or Commodity resulting in corresponding trade types and trade data container names

- EquityRainbowOption / EquityRainbowOptionData
- FxRainbowOption / FxRainbowOptionData
- CommodityRainbowOption / CommodityRainbowOptionData

Trade input and the associated payoff script are described in the following for 12 supported Rainbow Option variations.

Best Of Asset Or Cash Rainbow Option

The traditional trade representation is as follows, using an equity underlying in this example:

```
<Trade id="BestOfAssetOrCashRainbowOption#1">
  <TradeType>EquityRainbowOption</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <EquityRainbowOptionData>
    <Currency>USD</Currency>
    <Notional>1</Notional>
    <Strike>2000</Strike>
    <Underlyings>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.SPX</Name>
        <Weight>1.0</Weight>
      </Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.STOXX50E</Name>
        <Currency>EUR</Currency>
        <Weight>1.0</Weight>
      </Underlying>
    </Underlyings>
    <OptionData>
      <LongShort>Long</LongShort>
      <PayoffType>BestOfAssetOrCash</PayoffType>
      <ExerciseDates>
```

```
        <ExerciseDate>2020-02-15</ExerciseDate>
      </ExerciseDates>
    </OptionData>
    <Settlement>2020-02-20</Settlement>
  </EquityRainbowOptionData>
</Trade>
```

with the following elements:

- Currency: The pay currency. Notice section “Payment Currency” in vre/docs/scripted_trade.
  Allowable values: all supported currency codes, see Table 22 `Currency`.
- Notional: The quantity (for equity, commodity underlyings) / foreign amount (fx underlying)
  Allowable values: all positive real numbers
- Strike: The strike of the option
  Allowable values: all positive real numbers
- Underlyings: The basket of underlyings.
  Allowable values: for each underlying see 19.3.29
- OptionData: relevant are the long/short flag, the payoff type (must be set to BestOfAssetOrCash to identify the payoff), and the exercise date (exactly one date must be given)
  Allowable values: see 19.3.1 for the general structure of the option data node
- Settlement: the settlement date (optional, if not given defaults to the exercise date)
  Allowable values: each valid date.

The representation as a scripted trade is as follows:

```
<Trade id="BestOfAssetOrCashRainbowOption#1">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope/>
  <BestOfAssetOrCashRainbowOptionData>
    <Expiry type="event">2020-02-15</Expiry>
    <Settlement type="event">2020-02-20</Settlement>
    <LongShort type="longShort">Long</LongShort>
    <Notional type="number">1</Notional>
    <Strike type="number">2000</Strike>
    <Underlyings type="index">
      <Value>EQ-RIC:.SPX</Value>
      <Value>EQ-RIC:.STOXX50E</Value>
    </Underlyings>
    <Weights type="number">
      <Value>1.0</Value>
      <Value>1.0</Value>
    </Weights>
    <PayCcy type="currency">USD</PayCcy>
  </BestOfAssetOrCashRainbowOptionData>
</Trade>
```

The BestOfAssetOrCashRainbowOption script referenced in the trade above is shown in Listing 169.

The meanings and allowable values of the elements in the `BestOfAssetOrCashRainbowOptionData` node are given below, with data type indicated in square brackets.

- [event] `Expiry`: Option expiry date.
  Allowable values: See `Date` in Table 20.
- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.
- [longShort] `LongShort`: Position type, *Long* if we buy, *Short* if we sell.
  Allowable values: *Long, Short*.
- [number] `Notional`: Quantity multiplier applied to the basket price
  Allowable values: Any positive real number.
- [number] `Strike`: Strike price in PayCcy (see below)
  Allowable values: Any positive real number.
- [index] `Underlyings`: List of underlying indices enclosed by `<Value>` and `</Value>` tags.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `Weights`: List of weights applied to each of the underlying prices, given in the same order as the Underlyings above, each weighted enclosed by `<Value>` and `</Value>` tags.
  Allowable values: Any positive real number.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Worst Of Asset Or Cash Rainbow Option

The traditional trade representation is as follows, using an equity underlying in this example:

```
<Trade id="WorstOfAssetOrCashRainbowOption#1">
  <TradeType>EquityRainbowOption</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <EquityRainbowOptionData>
    <Currency>USD</Currency>
    <Notional>1</Notional>
    <Strike>2000</Strike>
    <Underlyings>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.SPX</Name>
        <Currency>USD</Currency>
        <Weight>1.0</Weight>
      </Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.STOXX50E</Name>
        <Weight>1.0</Weight>
      </Underlying>
    </Underlyings>
    <OptionData>
      <LongShort>Long</LongShort>
      <PayoffType>WorstOfAssetOrCash</PayoffType>
      <ExerciseDates>
```

```
        <ExerciseDate>2020-02-15</ExerciseDate>
      </ExerciseDates>
    </OptionData>
    <Settlement>2020-02-20</Settlement>
  </EquityRainbowOptionData>
</Trade>
```

with the following elements:

- Currency: The pay currency. Notice section “Payment Currency” in vre/docs/scripted_trade.
  Allowable values: all supported currency codes, see Table 22 `Currency`.
- Notional: The quantity (for equity, commodity underlyings) / foreign amount (fx underlying)
  Allowable values: all positive real numbers
- Strike: The strike of the option
  Allowable values: all positive real numbers
- Underlyings: The basket of underlyings.
  Allowable values: for each underlying see 19.3.29
- OptionData: relevant are the long/short flag, the payoff type (must be set to WorstOfAssetOrCash to identify the payoff), and the exercise date (exactly one date must be given)
  Allowable values: see 19.3.1 for the general structure of the option data node
- Settlement: the settlement date (optional, if not given defaults to the exercise date)
  Allowable values: each valid date.

The representation as a scripted trade is as follows:

```
<Trade id="WorstOfAssetOrCashRainbowOption#1">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope/>
  <WorstOfAssetOrCashRainbowOptionData>
    <Expiry type="event">2020-02-15</Expiry>
    <Settlement type="event">2020-02-20</Settlement>
    <LongShort type="longShort">Long</LongShort>
    <Notional type="number">1</Notional>
    <Strike type="number">2000</Strike>
    <Underlyings type="index">
      <Value>EQ-RIC:.SPX</Value>
      <Value>EQ-RIC:.STOXX50E</Value>
    </Underlyings>
    <Weights type="number">
      <Value>1.0</Value>
      <Value>1.0</Value>
    </Weights>
    <PayCcy type="currency">USD</PayCcy>
  </WorstOfAssetOrCashRainbowOptionData>
</Trade>
```

The WorstOfAssetOrCashRainbowOption script referenced in the trade above is shown in Listing 170.

The meanings and allowable values of the elements in the `BestOfAssetOrCashRainbowOptionData` node are given below, with data type indicated in square brackets.

- [event] `Expiry`: Option expiry date.
  Allowable values: See `Date` in Table 20.
- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.
- [longShort] `LongShort`: Position type, *Long* if we buy, *Short* if we sell.
  Allowable values: *Long, Short*.
- [number] `Notional`: Quantity multiplier applied to the basket price
  Allowable values: Any positive real number.
- [number] `Strike`: Strike price in PayCcy (see below)
  Allowable values: Any positive real number.
- [index] `Underlyings`: List of underlying indices enclosed by `<Value>` and `</Value>` tags.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `Weights`: List of weights applied to each of the underlying prices, given in the same order as the Underlyings above, each weighted enclosed by `<Value>` and `</Value>` tags.
  Allowable values: Any positive real number.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Put/Call on Max Rainbow Option

The traditional trade representation is as follows, using an equity underlying in this example:

```
<Trade id="MaxRainbowOption#1">
  <TradeType>EquityRainbowOption</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <EquityRainbowOptionData>
    <Currency>USD</Currency>
    <Notional>1</Notional>
    <Strike>3000</Strike>
    <Underlyings>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.SPX</Name>
        <Weight>1.0</Weight>
      </Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.STOXX50E</Name>
        <Weight>1.0</Weight>
      </Underlying>
    </Underlyings>
    <OptionData>
      <LongShort>Long</LongShort>
      <OptionType>Call</OptionType>
      <PayoffType>MaxRainbow</PayoffType>
      <ExerciseDates>
```

```
        <ExerciseDate>2020-02-15</ExerciseDate>
      </ExerciseDates>
    </OptionData>
    <Settlement>2020-02-20</Settlement>
  </EquityRainbowOptionData>
</Trade>
```

with the following elements:

- Currency: The pay currency. Notice section “Payment Currency” in vre/docs/scripted_trade.
  Allowable values: all supported currency codes, see Table 22 `Currency`.
- Notional: The quantity (for equity, commodity underlyings) / foreign amount (fx underlying)
  Allowable values: all positive real numbers
- Strike: The strike of the option
  Allowable values: all positive real numbers
- Underlyings: The basket of underlyings.
  Allowable values: for each underlying see 19.3.29
- OptionData: relevant are the long/short flag, the option type, the payoff type (must be set to MaxRainbow to identify the payoff), and the exercise date (exactly one date must be given)
  Allowable values: see 19.3.1 for the general structure of the option data node
- Settlement: the settlement date (optional, if not given defaults to the exercise date)
  Allowable values: each valid date.

The representation as a scripted trade is as follows:

```
<Trade id="MaxRainbowOption#1">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope/>
  <MaxRainbowOptionData>
    <Expiry type="event">2020-02-15</Expiry>
    <Settlement type="event">2020-02-20</Settlement>
    <PutCall type="optionType">Call</PutCall>
    <LongShort type="longShort">Long</LongShort>
    <Notional type="number">1</Notional>
    <Strike type="number">3000</Strike>
    <Underlyings type="index">
      <Value>EQ-RIC:.SPX</Value>
      <Value>EQ-RIC:.STOXX50E</Value>
    </Underlyings>
    <Weights type="number">
      <Value>1.0</Value>
      <Value>1.0</Value>
    </Weights>
    <PayCcy type="currency">USD</PayCcy>
  </MaxRainbowOptionData>
</Trade>
```

The MainRainbowOption script referenced in the trade above is shown in Listing 171.

The meanings and allowable values of the elements in the `MaxRainbowOptionData` node are given below, with data type indicated in square brackets.

- [event] `Expiry`: Option expiry date.
  Allowable values: See `Date` in Table 20.

- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.
- [optionType] `PutCall`: Option type with
  Allowable values *Call, Put*.
- [longShort] `LongShort`: Position type, *Long* if we buy, *Short* if we sell.
  Allowable values: *Long, Short*.
- [number] `Notional`: Quantity multiplier applied to the basket price
  Allowable values: Any positive real number.
- [number] `Strike`: Strike price in PayCcy (see below)
  Allowable values: Any positive real number.
- [index] `Underlyings`: List of underlying indices enclosed by `<Value>` and `</Value>` tags.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `Weights`: List of weights applied to each of the underlying prices, given in the same order as the Underlyings above, each weight enclosed by `<Value>` and `</Value>` tags.
  Allowable values: Any positive real number.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Put/Call on Min Rainbow Option

The traditional trade representation is as follows, using an equity underlying in this example:

```
<Trade id="MinRainbowOption#1">
  <TradeType>EquityRainbowOption</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <EquityRainbowOptionData>
    <Currency>USD</Currency>
    <Notional>1</Notional>
    <Strike>2000</Strike>
    <Underlyings>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.SPX</Name>
        <Weight>1.0</Weight>
      </Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.STOXX50E</Name>
        <Weight>1.0</Weight>
      </Underlying>
    </Underlyings>
    <OptionData>
      <LongShort>Long</LongShort>
      <OptionType>Call</OptionType>
      <PayoffType>MinRainbow</PayoffType>
      <ExerciseDates>
```

```
        <ExerciseDate>2020-02-15</ExerciseDate>
      </ExerciseDates>
    </OptionData>
    <Settlement>2020-02-20</Settlement>
  </EquityRainbowOptionData>
</Trade>
```

with the following elements:

- Currency: The pay currency. Notice section “Payment Currency” in vre/docs/scripted_trade.
  Allowable values: all supported currency codes, see Table 22 `Currency`.
- Notional: The quantity (for equity, commodity underlyings) / foreign amount (fx underlying)
  Allowable values: all positive real numbers
- Strike: The strike of the option
  Allowable values: all positive real numbers
- Underlyings: The basket of underlyings.
  Allowable values: for each underlying see 19.3.29
- OptionData: relevant are the long/short flag, the option type, the payoff type (must be set to MaxRainbow to identify the payoff), and the exercise date (exactly one date must be given)
  Allowable values: see 19.3.1 for the general structure of the option data node
- Settlement: the settlement date (optional, if not given defaults to the exercise date)
  Allowable values: each valid date.

The representation as a scripted trade is as follows:

```
<Trade id="MinRainbowOption#1">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope/>
   <MinRainbowOptionData>
    <Expiry type="event">2020-02-15</Expiry>
    <Settlement type="event">2020-02-20</Settlement>
    <PutCall type="optionType">Call</PutCall>
    <LongShort type="longShort">Long</LongShort>
    <Notional type="number">1</Notional>
    <Strike type="number">2000</Strike>
    <Underlyings type="index">
      <Value>EQ-RIC:.SPX</Value>
      <Value>EQ-RIC:.STOXX50E</Value>
    </Underlyings>
    <Weights type="number">
      <Value>1.0</Value>
      <Value>1.0</Value>
    </Weights>
    <PayCcy type="currency">USD</PayCcy>
  </MinRainbowOptionData>
</Trade>
```

The MinRainbowOption script referenced in the trade above is shown in Listing 172.

The meanings and allowable values of the elements in the `MinRainbowOptionData` node are given below, with data type indicated in square brackets.

- [event] `Expiry`: Option expiry date.
  Allowable values: See `Date` in Table 20.

- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.
- [optionType] `PutCall`: Option type with
  Allowable values *Call, Put*.
- [longShort] `LongShort`: Position type, *Long* if we buy, *Short* if we sell.
  Allowable values: *Long, Short*.
- [number] `Notional`: Quantity multiplier applied to the basket price
  Allowable values: Any positive real number.
- [number] `Strike`: Strike price in PayCcy (see below)
  Allowable values: Any positive real number.
- [index] `Underlyings`: List of underlying indices enclosed by `<Value>` and `</Value>` tags.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `Weights`: List of weights applied to each of the underlying prices, given in the same order as the Underlyings above, each weight enclosed by `<Value>` and `</Value>` tags.
  Allowable values: Any positive real number.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

European Rainbow Call Spread Option

European rainbow call spread options are represented as *scripted trades*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction.

Trade input and the payoff script are described below.

```
<TradeType>ScriptedTrade</TradeType>
<Envelope/>
<EuropeanRainbowCallSpreadOptionData>
  <Expiry type="event">2020-02-15</Expiry>
  <Settlement type="event">2020-02-20</Settlement>
  <LongShort type="longShort">Long</LongShort>
  <Notional type="number">1000000</Notional>
  <Underlyings type="index">
    <Value>EQ-RIC:.SPX</Value>
    <Value>EQ-RIC:.STOXX50E</Value>
  </Underlyings>
  <InitialStrikes type="number">
    <Value>2100</Value>
    <Value>3000</Value>
  </InitialStrikes>
  <Weights type="number">
    <Value>0.8</Value>
    <Value>0.3</Value>
  </Weights>
  <Floor type="number">0.02</Floor>
  <Cap type="number">0.10</Cap>
  <PayCcy type="currency">USD</PayCcy>
</EuropeanRainbowCallSpreadOptionData>
```

The EuropeanRainbowCallSpreadOption script referenced in the trade above is shown in listing 173.

The meanings and allowable values of the elements in the `EuropeanRainbowCallSpreadOptionData` node below.

- [event] `Expiry`: Option expiry date.
  Allowable values: See `Date` in Table 20.
- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.
- [longShort] `LongShort`: long short flag.
  Allowable values: Long, Short
- [number] `Notional`: The notional.
  Allowable values: A real number.
- [index] `Underlyings`: The underlyings.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `InitialStrikes`: The initial strikes of the underlyings.
  Allowable values: A real number for each underlying.
- [number] `Weights`: The weights for the best, second best, ..., worst performing underlying.
  Allowable values: A real number for each rank.
- [number] `Floor`: The floor. If no floor applies, use e.g. -1E10. Allowable values: A real number.
- [number] `Cap`: The floor. If no floor applies, use e.g. 1E10. Allowable values: A real number.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Rainbow Call Spread Barrier Option

A rainbow call spread barrier option is an extension of the European Rainbow Call Spread Option, and is represented as *scripted trades*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction.

Trade input and the payoff script are described below.

```
<TradeType>ScriptedTrade</TradeType>
<Envelope>
  <CounterParty>CPTY_A</CounterParty>
  <NettingSetId>CPTY_A</NettingSetId>
  <AdditionalFields/>
</Envelope>
<RainbowCallSpreadBarrierOptionData>
  <Expiry type="event">2020-02-15</Expiry>
  <Settlement type="event">2020-02-20</Settlement>
  <LongShort type="longShort">Long</LongShort>
```

```
    <Notional type="number">1000000</Notional>
    <Underlyings type="index">
      <Value>EQ-RIC:.SPX</Value>
      <Value>EQ-RIC:.STOXX50E</Value>
    </Underlyings>
    <InitialPrices type="number">
      <Value>2100</Value>
      <Value>3000</Value>
    </InitialPrices>
    <Weights type="number">
      <Value>0.8</Value>
      <Value>0.3</Value>
    </Weights>
    <Strike type="number">1.0</Strike>
    <Floor type="number">0.02</Floor>
    <Cap type="number">0.10</Cap>
    <Gearing type="number">1.0</Gearing>
    <BermudanBarrier type="bool">false</BermudanBarrier>
    <BarrierLevel type="number">1000.0</BarrierLevel>
    <BarrierSchedule type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2018-12-31</StartDate>
          <EndDate>2020-02-15</EndDate>
          <Tenor>1M</Tenor>
          <Calendar>USD</Calendar>
          <Convention>ModifiedFollowing</Convention>
          <TermConvention>ModifiedFollowing</TermConvention>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
    </BarrierSchedule>
    <PayCcy type="currency">USD</PayCcy>
  </RainbowCallSpreadBarrierOptionData>
```

The EuropeanRainbowCallSpreadOption script referenced in the trade above is shown in listing 174.

The meanings and allowable values of the elements in the `RainbowCallSpreadBarrierOptionData` node below.

- [event] `Expiry`: Option expiry date.
  Allowable values: See `Date` in Table 20.
- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.
- [longShort] `LongShort`: long short flag.
  Allowable values: *Long*, *Short*
- [number] `Notional`: The notional.
  Allowable values: A real number.
- [index] `Underlyings`: The underlyings.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `InitialPrices`: The initial prices of the underlyings.
  Allowable values: A real number for each underlying.
- [number] `Weights`: The weights for the best, second best, ..., worst performing underlying.
  Allowable values: A real number for each rank.
- [number] `Strike`: The option strike price.

Allowable values: Any number, as a percentage expressed in decimal form.

- [number] `Floor`: The floor. If no floor applies, use e.g. -1E10.
  Allowable values: A real number.
- [number] `Cap`: The floor. If no floor applies, use e.g. 1E10.
  Allowable values: A real number.
- [number] `Gearing`: The gearing/payoff multiplier, applied after the cap and/or floor.
  Allowable values: A real number.
- [bool] `BermudanBarrier`: Whether the KI barrier observation is Bermudan (*True*) or European (*False*).
  Allowable values: *True* or *False*.
- [number] `BarrierLevel`: The agreed knock-in barrier price level.
  Allowable values: Any number.
- [event] `BarrierSchedule`: If `BermudanBarrier` is *True*, this is the barrier observation schedule. If *False*, this sub-node is still required but will not be used.
  Allowable values: See Section 19.3.4.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 `Currency` for allowable currency codes.

Asian Rainbow Call Spread Option

Asian rainbow call spread options are represented as *scripted trades*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction.

Trade input and the payoff script are described below.

```
<TradeType>ScriptedTrade</TradeType>
<Envelope/>
<AsianRainbowCallSpreadOptionData>
  <Expiry type="event">2020-02-15</Expiry>
  <AveragingDates type="event">
    <ScheduleData>
      <Dates>
        <Dates>
          <Date>2019-01-29</Date>
                .....
          <Date>2020-02-15</Date>
        </Dates>
        <Calendar>USD</Calendar>
        <Convention>MF</Convention>
      </Dates>
    </ScheduleData>
  </AveragingDates>
  <Settlement type="event">2020-02-20</Settlement>
  <LongShort type="longShort">Long</LongShort>
  <Notional type="number">1000000</Notional>
  <Underlyings type="index">
    <Value>EQ-RIC:.SPX</Value>
    <Value>EQ-RIC:.STOXX50E</Value>
  </Underlyings>
  <InitialStrikes type="number">
    <Value>2100</Value>
    <Value>3000</Value>
```

```
    </InitialStrikes>
    <Weights type="number">
      <Value>0.8</Value>
      <Value>0.3</Value>
    </Weights>
    <Floor type="number">0.02</Floor>
    <Cap type="number">0.10</Cap>
    <PayCcy type="currency">USD</PayCcy>
  </AsianRainbowCallSpreadOptionData>
```

The AsianRainbowCallSpreadOption script referenced in the trade above is shown in Listing 175.

The meanings and allowable values of the elements in the `AsianRainbowCallSpreadOptionData` node below.

- [event] `Expiry`: Option expiry date.
  Allowable values: See `Date` in Table 20.
- [event] `AveragingDates`: Observation dates for calculating the final (average) price of each underlying. Allowable values: See Section 19.3.4.
- [event] `Settlement`: Option settlement date.
  Allowable values: See `Date` in Table 20.
- [longShort] `LongShort`: long short flag.
  Allowable values: Long, Short
- [number] `Notional`: The notional.
  Allowable values: A real number.
- [index] `Underlyings`: The underlyings.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `InitialStrikes`: The initial strikes of the underlyings.
  Allowable values: A real number for each underlying.
- [number] `Weights`: The weights for the best, second best, ..., worst performing underlying.
  Allowable values: A real number for each rank.
- [number] `Floor`: The floor. If no floor applies, use e.g. -1E10. Allowable values: A real number.
- [number] `Cap`: The floor. If no floor applies, use e.g. 1E10. Allowable values: A real number.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Worst Performance Rainbow Option 01

A worst performance rainbow option 01 is represented as a *scripted trade*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction.

Trade input and the payoff script are described below.

```
<TradeType>ScriptedTrade</TradeType>
<Envelope>
  <CounterParty>CPTY_A</CounterParty>
  <NettingSetId>CPTY_A</NettingSetId>
  <AdditionalFields/>
</Envelope>
<WorstPerformanceRainbowOption01Data>
  <LongShort type="longShort">Long</LongShort>
  <Underlyings type="index">
    <Value>EQ-RIC:.STOXX50E</Value>
    <Value>EQ-RIC:.SPX</Value>
  </Underlyings>
  <InitialPrices type="number">
    <Value>5455.60</Value>
    <Value>500</Value>
  </InitialPrices>
  <Premium type="number">291264</Premium>
  <PremiumDate type="event">2020-03-11</PremiumDate>
  <Quantity type="number">72816000</Quantity>
  <PayoffMultiplier type="number">0.4625</PayoffMultiplier>
  <ObservationDate type="event">2020-09-04</ObservationDate>
  <SettlementDate type="event">2020-09-11</SettlementDate>
  <PayCcy type="currency">EUR</PayCcy>
</WorstPerformanceRainbowOption01Data>
```

The WorstPerformanceRainbowOption01 script referenced in the trade above is shown in listing 176.

The payout formula from the long perspective, determined on the `ObservationDate`, is

$$Payout = \texttt{Quantity} * (worstPerformance - 1)$$

where $worstPerformance$ is the performance, i.e. S_T/S_0, of the worst-performing asset as of the final determination date T.

The meanings and allowable values for the `WorstPerformanceRainbowOption01Data` node below.

- [longShort] `LongShort`: Own party position in the option.
  Allowable values: *Long, Short*.
- [index] `Underlyings`: The basket of underlyings.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `InitialPrices`: The agreed initial price for each basket underlying.
  Allowable values: Any positive number.
- [number] `Premium`: Total option premium amount in terms of the *PayCcy*
  Allowable values: See Table 22 `Currency`.
- [event] `PremiumDate`: The premium payment date.
  Allowable values: See `Date` in Table 20.
- [number] `Quantity`: A quantity multiplier applied to the option payoff.
  Allowable values: Any number.
- [number] `PayoffMultiplier`: A factor that is multiplied to the option payoff when the option buyer incurs a negative net cash flow, i.e. when the performance of the worst-performing asset is less than 1.

Allowable values: Any number, as a percentage expressed in decimal form.

- [event] `ObservationDate`: The date on which the final levels of the assets are determined.
  Allowable values: See `Date` in Table 20.
- [event] `SettlementDate`: The settlement date for the option payoff.
  Allowable values: See `Date` in Table 20.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Worst Performance Rainbow Option 02

A worst performance rainbow option 02 is represented as a *scripted trade*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction.

Trade input and the payoff script are described below.

```
<TradeType>ScriptedTrade</TradeType>
<Envelope>
  <CounterParty>CPTY_A</CounterParty>
  <NettingSetId>CPTY_A</NettingSetId>
  <AdditionalFields/>
</Envelope>
<WorstPerformanceRainbowOption02Data>
  <LongShort type="longShort">Long</LongShort>
  <Underlyings type="index">
    <Value>EQ-RIC:.STOXX50E</Value>
    <Value>EQ-RIC:.SPX</Value>
  </Underlyings>
  <InitialPrices type="number">
    <Value>4890.00</Value>
    <Value>108.84</Value>
  </InitialPrices>
  <Premium type="number">1731</Premium>
  <PremiumDate type="event">2020-02-27</PremiumDate>
  <Quantity type="number">90000000</Quantity>
  <PayoffMultiplier type="number">1.7</PayoffMultiplier>
  <Floor type="number">-0.05</Floor>
  <ObservationDate type="event">2020-11-13</ObservationDate>
  <SettlementDate type="event">2020-11-27</SettlementDate>
  <PayCcy type="currency">USD</PayCcy>
</WorstPerformanceRainbowOption02Data>
```

The WorstPerformanceRainbowOption02 script referenced in the trade above is shown in listing 177.

The payout formula from the long perspective, determined on the `ObservationDate`, is as follows, where $worstPerformance$ is the performance, i.e. S_T/S_0, of the worst-performing asset as of the final determination date T:

If $worstPerformance > 1$, receive

$$Payout = \texttt{Quantity} * \texttt{PayoffMultiplier} * (worstPerformance - 1).$$

If $worstPerformance \leq 1$, pay

$$Payout = \texttt{Quantity} * \max(\texttt{Floor}, worstPerformance - 1)$$

The meanings and allowable values for the `WorstPerformanceRainbowOption02Data` node below.

- [longShort] `LongShort`: Own party position in the option.
  Allowable values: *Long, Short*.
- [index] `Underlyings`: The basket of underlyings.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `InitialPrices`: The agreed initial price for each basket underlying.
  Allowable values: Any positive number.
- [number] `Premium`: Total option premium amount in terms of the *PayCcy*
  Allowable values: See Table 22 `Currency`.
- [event] `PremiumDate`: The premium payment date.
  Allowable values: See `Date` in Table 20.
- [number] `Quantity`: A quantity multiplier applied to the option payoff.
  Allowable values: Any number.
- [number] `PayoffMultiplier`: A factor that is multiplied to the option payoff when the option buyer incurs a positive net cash flow, i.e. when the performance of the worst-performing asset is greater than 1.
  Allowable values: Any number, as a percentage expressed in decimal form.
- [number] `Floor`: The maximum loss that the option buyer can incur.
  Allowable values: Any non-positive number, as a percentage expressed in decimal form.
- [event] `ObservationDate`: The date on which the final levels of the assets are determined.
  Allowable values: See `Date` in Table 20.
- [event] `SettlementDate`: The settlement date for the option payoff.
  Allowable values: See `Date` in Table 20.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 `Currency` for allowable currency codes.

Worst Performance Rainbow Option 03

A worst performance rainbow option 03 is an extension of the Worst Performance Rainbow Option 01, and is represented as a *scripted trade*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction.

Trade input and the payoff script are described below.

```
<TradeType>ScriptedTrade</TradeType>
<Envelope>
  <CounterParty>CPTY_A</CounterParty>
  <NettingSetId>CPTY_A</NettingSetId>
  <AdditionalFields/>
</Envelope>
```

```
<WorstPerformanceRainbowOption03Data>
  <LongShort type="longShort">Long</LongShort>
  <Underlyings type="index">
    <Value>EQ-RIC:.STOXX50E</Value>
    <Value>EQ-RIC:.SPX</Value>
  </Underlyings>
  <InitialPrices type="number">
    <Value>5455.60</Value>
    <Value>500</Value>
  </InitialPrices>
  <Premium type="number">291264</Premium>
  <PremiumDate type="event">2020-03-11</PremiumDate>
  <Strike type="number">1.0</Strike>
  <Quantity type="number">72816</Quantity>
  <PayoffMultiplier type="number">2.5</PayoffMultiplier>
  <Cap type="number">100.0</Cap>
  <Floor type="number">-100.0</Floor>
  <BermudanBarrier type="bool">false</BermudanBarrier>
  <BarrierLevel type="number">0.7</BarrierLevel>
  <BarrierSchedule type="event">
    <ScheduleData>
      <Rules>
        <StartDate>2020-03-11</StartDate>
        <EndDate>2020-09-04</EndDate>
        <Tenor>1D</Tenor>
        <Calendar>USD</Calendar>
        <Convention>ModifiedFollowing</Convention>
        <TermConvention>ModifiedFollowing</TermConvention>
        <Rule>Forward</Rule>
      </Rules>
    </ScheduleData>
  </BarrierSchedule>
  <ObservationDate type="event">2020-09-04</ObservationDate>
  <SettlementDate type="event">2020-09-11</SettlementDate>
  <PayCcy type="currency">EUR</PayCcy>
</WorstPerformanceRainbowOption03Data>
```

The WorstPerformanceRainbowOption03 script referenced in the trade above is shown in listing 178.

The payout formula from the long perspective, determined on the `ObservationDate`, is as follows, where $worstPerformance$ is the performance, i.e. S_T/S_0, of the worst-performing asset as of the final determination date T:

If $worstPerformance \geq$ `Strike`, receive

$$Payout = \texttt{Quantity} * \min\left(\texttt{Cap}, \max(\texttt{Floor}, worstPerformance - \texttt{Strike})\right).$$

If $worstPerformance <$ `Strike`, pay

$$Payout = \texttt{Quantity} * \texttt{PayoffMultiplier} * \min\left(\texttt{Cap}, \max(\texttt{Floor}, worstPerformance - \texttt{Strike})\right).$$

The above payouts are conteningent on a knock-in event. If no knock-in has occurred, the payout is zero.

The meanings and allowable values for the `WorstPerformanceRainbowOption03Data` node below.

- [longShort] `LongShort`: Own party position in the option.
  Allowable values: *Long, Short*.
- [index] `Underlyings`: The basket of underlyings.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.

- [number] `InitialPrices`: The agreed initial price for each basket underlying.
  Allowable values: Any positive number.
- [number] `Premium`: Total option premium amount in terms of the *PayCcy*
  Allowable values: See Table 22 `Currency`.
- [event] `PremiumDate`: The premium payment date.
  Allowable values: See `Date` in Table 20.
- [number] `Strike`: The option strike price.
  Allowable values: Any number, as a percentage expressed in decimal form.
- [number] `Quantity`: A quantity multiplier applied to the option payoff.
  Allowable values: Any number.
- [number] `PayoffMultiplier`: A factor that is multiplied to the option payoff when the option buyer incurs a positive net cash flow, i.e. when the performance of the worst-performing asset is greater than 1.
  Allowable values: Any number, as a percentage expressed in decimal form.
- [number] `Cap`: The maximum profit that the option buyer can receive.
  Allowable values: Any number, as a percentage expressed in decimal form.
- [number] `Floor`: The maximum loss that the option buyer can incur.
  Allowable values: Any number, as a percentage expressed in decimal form.
- [bool] `BermudanBarrier`: Whether the KI barrier observation is Bermudan (*True*) or European (*False*).
  Allowable values: *True* or *False*.
- [number] `BarrierLevel`: The agreed knock-in barrier price level. For example, in the case of a Bermudan barrier, if a knock-in is set to occur when one of the underlying prices falls below 70% of its initial price, then the appropriate value is *0.7*, as in the sample trade input above.
  Allowable values: Any number, as a percentage of the `InitialPrices` expressed in decimal form.
- [event] `BarrierSchedule`: If `BermudanBarrier` is *True*, this is the barrier observation schedule. If *False*, this sub-node is still required but will not be used.
  Allowable values: See Section 19.3.4.
- [event] `ObservationDate`: The date on which the final levels of the assets are determined.
  Allowable values: See `Date` in Table 20.
- [event] `SettlementDate`: The settlement date for the option payoff.
  Allowable values: See `Date` in Table 20.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Worst Performance Rainbow Option 04

A worst performance rainbow option 04 is represented as a *scripted trade*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction.

Trade input and the payoff script are described below.

```
<TradeType>ScriptedTrade</TradeType>
<Envelope>
  <CounterParty>CPTY_A</CounterParty>
  <NettingSetId>CPTY_A</NettingSetId>
  <AdditionalFields/>
</Envelope>
<WorstPerformanceRainbowOption04Data>
  <LongShort type="longShort">Long</LongShort>
  <Underlyings type="index">
    <Value>EQ-RIC:.STOXX50E</Value>
    <Value>EQ-RIC:.SPX</Value>
  </Underlyings>
  <InitialPrices type="number">
    <Value>5455.60</Value>
    <Value>500</Value>
  </InitialPrices>
  <Premium type="number">291264</Premium>
  <PremiumDate type="event">2020-03-11</PremiumDate>
  <Strike type="number">1.0</Strike>
  <Quantity type="number">72816</Quantity>
  <PayoffMultiplier type="number">2.5</PayoffMultiplier>
  <Cap type="number">100.0</Cap>
  <Floor type="number">-100.0</Floor>
  <BermudanBarrier type="bool">false</BermudanBarrier>
  <BarrierLevel type="number">0.7</BarrierLevel>
  <BarrierSchedule type="event">
    <ScheduleData>
      <Rules>
        <StartDate>2020-03-11</StartDate>
        <EndDate>2020-09-04</EndDate>
        <Tenor>1D</Tenor>
        <Calendar>USD</Calendar>
        <Convention>ModifiedFollowing</Convention>
        <TermConvention>ModifiedFollowing</TermConvention>
        <Rule>Forward</Rule>
      </Rules>
    </ScheduleData>
  </BarrierSchedule>
  <ObservationDate type="event">2020-09-04</ObservationDate>
  <SettlementDate type="event">2020-09-11</SettlementDate>
  <PayCcy type="currency">EUR</PayCcy>
</WorstPerformanceRainbowOption04Data>
```

The WorstPerformanceRainbowOption04 script referenced in the trade above is shown in listing 179.

The payout formula, determined on the `ObservationDate`, is as follows, where $worstPerformance$ is the performance, i.e. S_T/S_0, of the worst-performing asset as of the final determination date T:

If a knock-in event was triggered:

$$Payout = \texttt{Quantity} * (worstPerformance - \texttt{Strike}).$$

If no knock-in event was triggered:

$$Payout = \texttt{Quantity} * \min\Big(\texttt{Cap}, \max\big(\texttt{Floor}, \texttt{PayoffMultiplier} * (worstPerformance - \texttt{Strike})\big)\Big).$$

From the long perspective, the above amounts are received if they are positive, and paid out from the short perspective.

The meanings and allowable values for the `WorstPerformanceRainbowOption04Data` node below.

- [longShort] `LongShort`: Own party position in the option.
  Allowable values: *Long, Short*.
- [index] `Underlyings`: The basket of underlyings.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `InitialPrices`: The agreed initial price for each basket underlying.
  Allowable values: Any positive number.
- [number] `Premium`: Total option premium amount in terms of the *PayCcy*
  Allowable values: See Table 22 `Currency`.
- [event] `PremiumDate`: The premium payment date.
  Allowable values: See `Date` in Table 20.
- [number] `Strike`: The option strike price.
  Allowable values: Any number, as a percentage expressed in decimal form.
- [number] `Quantity`: A quantity multiplier applied to the option payoff.
  Allowable values: Any number.
- [number] `PayoffMultiplier`: A factor that is multiplied to the option payoff when no knock-in event has ocurred. This multiplier is applied before the cap and/or floor.
  Allowable values: Any number, as a percentage expressed in decimal form.
- [number] `Cap`: The maximum profit that the option buyer can receive when a knock-in event has occurred.
  Allowable values: Any number, as a percentage expressed in decimal form.
- [number] `Floor`: The maximum loss that the option buyer can incur when a knock-in event has occurred.
  Allowable values: Any number, as a percentage expressed in decimal form.
- [bool] `BermudanBarrier`: Whether the KI barrier observation is Bermudan (*True*) or European (*False*).
  Allowable values: *True* or *False*.
- [number] `BarrierLevel`: The agreed knock-in barrier price level. For example, in the case of a Bermudan barrier, if a knock-in is set to occur when one of the underlying prices falls below 70% of its initial price, then the appropriate value is *0.7*, as in the sample trade input above.
  Allowable values: Any number, as a percentage of the `InitialPrices` expressed in decimal form.
- [event] `BarrierSchedule`: If `BermudanBarrier` is *True*, this is the barrier observation schedule. If *False*, this sub-node is still required but will not be used.
  Allowable values: See Section 19.3.4.
- [event] `ObservationDate`: The date on which the final levels of the assets are determined.

Allowable values: See `Date` in Table 20.

- [event] `SettlementDate`: The settlement date for the option payoff.
  Allowable values: See `Date` in Table 20.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Worst Performance Rainbow Option 05

A worst performance rainbow option 05 is represented as a *scripted trade*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction.

Trade input and the payoff script are described below.

```
<TradeType>ScriptedTrade</TradeType>
<Envelope>
  <CounterParty>CPTY_A</CounterParty>
  <NettingSetId>CPTY_A</NettingSetId>
  <AdditionalFields/>
</Envelope>
<WorstPerformanceRainbowOption05Data>
  <LongShort type="longShort">Long</LongShort>
  <PutCall type="optionType">Put</PutCall>
  <Underlyings type="index">
    <Value>EQ-RIC:.STOXX50E</Value>
    <Value>EQ-RIC:.SPX</Value>
  </Underlyings>
  <InitialPrices type="number">
    <Value>5455.60</Value>
    <Value>500</Value>
  </InitialPrices>
  <Premium type="number">291264</Premium>
  <PremiumDate type="event">2020-03-11</PremiumDate>
  <Strike type="number">1.0</Strike>
  <Quantity type="number">72816</Quantity>
  <BarrierType type="barrierType">DownIn</BarrierType>
  <BarrierLevel type="number">0.6</BarrierLevel>
  <ObservationDate type="event">2020-09-04</ObservationDate>
  <SettlementDate type="event">2020-09-11</SettlementDate>
  <PayCcy type="currency">EUR</PayCcy>
</WorstPerformanceRainbowOption05Data>
```

The WorstPerformanceRainbowOption05 script referenced in the trade above is shown in listing 180.

The payout formula, determined on the `ObservationDate`, is as follows, where $worstPerformance$ is the performance, i.e. S_T/S_0, of the worst-performing asset as of the final determination date T. The payout for a long put option is as follows:

If a knock-in event was triggered,

$$Payout = \texttt{Quantity} * \max\Big(\big(\texttt{Strike} - worstPerformance\big), 0\Big).$$

Otherwise, the payoff is zero.

The meanings and allowable values for the `WorstPerformanceRainbowOption05Data` node below.

- [longShort] `LongShort`: Own party position in the option.
  Allowable values: *Long, Short*.
- [optionType] `PutCall`: Option type. For FX, this should be *Call* if we buy `CCY1` and sell `CCY2`, *Put* if we buy `CCY2` and sell `CCY1` (where the `Underlying` is in the form `FX-SOURCE-CCY1-CCY2`).
- [index] `Underlyings`: The basket of underlyings.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `InitialPrices`: The agreed initial price for each basket underlying.
  Allowable values: Any positive number.
- [number] `Premium`: Total option premium amount in terms of the *PayCcy*
  Allowable values: See Table 22 `Currency`.
- [event] `PremiumDate`: The premium payment date.
  Allowable values: See `Date` in Table 20.
- [number] `Strike`: The option strike price.
  Allowable values: Any number, as a percentage expressed in decimal form.
- [number] `Quantity`: A quantity multiplier applied to the option payoff.
  Allowable values: Any number.
- [barrierType] `BarrierType`: The knock-in barrier type. For trades with no barrier, set `BarrierType` to *UpIn* and `BarrierLevel` to zero (or any negative number).
  Allowable values: *DownIn, UpIn*.
- [number] `BarrierLevel`: The agreed European knock-in barrier level.
  Allowable values: Any number, as a percentage of the `InitialPrices` expressed in decimal form.
- [event] `ObservationDate`: The date on which the final levels of the assets are determined.
  Allowable values: See `Date` in Table 20.
- [event] `SettlementDate`: The settlement date for the option payoff.
  Allowable values: See `Date` in Table 20.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

19.2.83 Exotic Variance and Volatility Derivatives

These are vanilla variance/volatility swaps and options on an underlying with some additional payoff features, such as knock-in/knock-out barrier/s, conditions for the accrual of variance, basket underlyings, optionality on the payoff, etc.

Exotic variance/volatility swaps and options are represented as *scripted trades*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction. Each of the

supported variations and their payoff script name are shown in Table 91. All swaps have an optional cap/floor feature.

Variance/Volatility Product Variation	Payoff Script Name
Variance/Volatility Option	`VarianceOption`
Variance/Volatility Swap with KI/KO Barrier	`KIKOVarianceSwap`
Corridor Volatility/Variance Swap	`CorridorVarianceSwap`
Corridor Variance Swap with KI/KO Barrier	`KIKOCorridorVarianceSwap`
Conditional Variance/Volatility Swap	`ConditionalVarianceSwap01` `ConditionalVarianceSwap02`
Pairwise Variance Swap	`PairwiseVarianceSwap`
Variance Dispersion Swap	`VarianceDispersionSwap`
Corridor Variance Dispersion Swap	`CorridorVarianceDispersionSwap`
Corridor Variance Dispersion Swap with KO Barrier	`KOCorridorVarianceDispersionSwap`
Gamma Swap	`GammaSwap`
Basket Variance Swap	`BasketVarianceSwap`

Table 91: Exotic variance/volatility product types and associated script names.

Trade input and the associated payoff script are described in the following for 12 supported variations.

Variance Option

The traditional trade representation is as follows, using an EQ underlying in this example:

```
<Trade id="EQ_VarianceOption">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    .....
  </Envelope>
  <VarianceOptionData>
    <LongShort type="longShort">Long</LongShort>
    <PutCall type="optionType">Call</PutCall>
    <PremiumAmount type="number">0</PremiumAmount>
    <PremiumDate type="event">2020-11-26</PremiumDate>
    <Notional type="number">138000</Notional>
    <VarianceReference type="number">0.19</VarianceReference>
    <Strike type="number">0.19</Strike>
    <Underlying type="index">EQ-RIC:.SPX</Underlying>
    <ValuationSchedule type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2020-11-26</StartDate>
          <EndDate>2021-09-18</EndDate>
          <Tenor>1D</Tenor>
          <Convention>Following</Convention>
          <TermConvention>Following</TermConvention>
          <Calendar>USA</Calendar>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
    </ValuationSchedule>
    <SquaredPayoff type="bool">true</SquaredPayoff>
    <SettlementDate type="event">2021-09-22</SettlementDate>
    <PayCcy type="currency">USD</PayCcy>
  </VarianceOptionData>
</Trade>
```

The VarianceOption script referenced in the trade above is shown in listing 181

The payout formulas for a variance put and call option are:

$$PutPayoff = 100^2 * \frac{\texttt{Notional}}{2 * 100 * \texttt{VarianceReference}} * \max[\texttt{Strike} - \texttt{RealisedVariance}, 0],$$
$$CallPayoff = 100^2 * \frac{\texttt{Notional}}{2 * 100 * \texttt{VarianceReference}} * \max[\texttt{Strike} - \texttt{RealisedVariance}, 0].$$

The meanings and allowable values for the `VarianceOptionData` node below.

- [longShort] `LongShort`: Own party position in the option. *Long* corresponds to paying out on the fixed/strike variance (volatility) and receiving on the floating/realised variance (volatility). In other words, a long position has positive value if the realised variance (volatility) exceeds the variance (volatility) strike.
  Allowable values: *Long, Short*.
- [optionType] `PutCall`: Option type. For FX, this should be *Call* if we buy `CCY1` and sell `CCY2`, *Put* if we buy `CCY2` and sell `CCY1` (where the `Underlying` is in the form `FX-SOURCE-CCY1-CCY2`).
  Allowable values: *Put, Call*
- [number] `PremiumAmount`: The total option premium amount. Allowable values: Any non-negativenumber.
- [event] `PremiumDate`: The option premium payment date. Allowable values: See `Date` in Table 20.
- [number] `Notional`: The vega notional amount. If the option was struck in terms of a variance notional N_{var}, the corresponding vega notional is given by $N_{vol} = N_{var} \cdot 2 \cdot 100 \cdot K_{vol}$ (where K_{vol} is in absolute terms).
  Allowable values: Any non-negative number.
- [number] `Strike`: The volatility strike K_{vol} quoted in absolute terms. If the option was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.
- [number] `VarianceReference`: The parameter used to convert the vega notional amount into the corresponding variance amount. Similar to the `Strike`, this should be quoted in absolute terms, e.g. if the (volatility) strike price is 20% and the variance reference is 32.4%, then the `Strike` is *0.20* and the `VarianceReference` should be *0.324*.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.
- [index] `Underlying`: Underlying index.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [event] `ValuationSchedule`: The schedule defining the (daily) observation period for the variance accrual.
  Allowable values: See Section 19.3.4.
- [bool] `SquaredPayoff`: Flag indicating whether the trade is a variance option (*True*) or a volatility option (*False*).

Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.

- [event] `SettlementDate`: The date on which the option payoff is settled.
  Allowable values: See `Date` in Table 20.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Variance Swap with KI/KO Barrier

The traditional trade representation is as follows, using an FX underlying in this example:

```
<Trade id="FX_VarianceSwap_KIKO">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    .....
  </Envelope>
  <KIKOVarianceSwapData>
    <LongShort type="longShort">Long</LongShort>
    <Strike type="number">0.02</Strike>
    <Notional type="number">50000</Notional>
    <Underlying type="index">FX-ECB-EUR-USD</Underlying>
    <ValuationSchedule type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2018-12-31</StartDate>
          <EndDate>2019-05-05</EndDate>
          <Tenor>1D</Tenor>
          <Convention>F</Convention>
          <TermConvention>F</TermConvention>
          <Calendar>US</Calendar>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
    </ValuationSchedule>
    <SquaredPayoff type="bool">true</SquaredPayoff>
    <BarrierType type="barrierType">UpIn</BarrierType>
    <BarrierLevel type="number">0</BarrierLevel>
    <Cap type="number">2.5</Cap>
    <Floor type="number">0</Floor>
    <SettlementDate type="event">2019-05-06</SettlementDate>
    <PayCcy type="currency">USD</PayCcy>
  </KIKOVarianceSwapData>
</Trade>
```

The KIKOVarianceSwap script referenced in the trade above is shown in listing 182

The payout formula when no knock-out has occurred is:

$$Payout = 100^2 * varianceAmount * \left[\min\left(\max(realisedVariance^2, \texttt{Floor}), \texttt{Cap} \right) - \texttt{Strike}^2 \right].$$

If a knock-out has occurred, the above amount is scaled by the number of days between the first variance accrual date and the date of knock-out, as a fraction of the total number of variance accrual dates.

The meanings and allowable values for the `KIKOVarianceSwapData` node below.

- [longShort] `LongShort`: Own party position in the swap. *Long* corresponds to paying out on the fixed/strike variance (volatility) and receiving on the floating/realised variance (volatility). In other words, a long position has positive value if the realised variance (volatility) exceeds the variance (volatility) strike.
  Allowable values: *Long, Short*.
- [number] `Strike`: The volatility strike K_{vol} of the variance swap quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.
- [number] `Notional`: The vega notional amount. If the swap was struck in terms of a variance notional N_{var}, the corresponding vega notional is given by $N_{vol} = N_{var} \cdot 2 \cdot 100 \cdot K_{vol}$ (where K_{vol} is in absolute terms).
  Allowable values: Any non-negative number.
- [index] `Underlying`: Underlying index.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [event] `ValuationSchedule`: The schedule defining the (daily) observation period for the variance accrual.
  Allowable values: `See` Section 19.3.4.
- [bool] SquaredPayoff: Flag indicating whether the trade is a variance swap (*True*) or a volatility swap (*False*).
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.
- [barrierType] `BarrierType`: Whether the barrier is a knock-in (*DownIn* or *UpIn*) or a knock-out (*DownOut* or *UpOut*) barrier. For trades with no barrier, which is equivalent to a capped/floored variance swap, set `BarrierType` to *UpIn* and `BarrierLevel` to zero, as in the sample trade representation above.
  Allowable values: *DownIn, UpIn, DownOut, UpOut*.
- [number] `BarrierLevel`: The agreed knock-in/knock-out barrier price level.
  Allowable values: Any non-negative real number.
- [number] `Cap`: The cap on the realised variance (or volatility), as a factor of the `Strike`. For example, if `Cap` is 2.5, then the cap level will be $2.5^2 \times$ `Strike`^{2} for variance swaps, and $2.5 \times$ `Strike` for volatility swaps. For trades with no cap, set `Cap` to zero.
  Allowable values: Any non-negative number.
- [number] `Floor`: The floor on the realised variance (or volatility), as a factor of the `Strike`. For example, if `Floor` is 0.1, then the floor level will be $0.1^2 \times$ `Strike` for variance swaps, and $0.1 \times$ `Strike` for volatility swaps. For trades with no floor, set `Floor` to zero.
  Allowable values: Any non-negative number.
- [event] `SettlementDate`: The date on which the swap payoff is settled.
  Allowable values: See `Date` in Table 20.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided

in the form FX-SOURCE-CCY1-CCY2 (see Table 28) this should be CCY2. If CCY1 or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
Allowable values: See Table 22 for allowable currency codes.

Corridor Variance Swap

The traditional trade representation is as follows, using an FX underlying in this example:

```
<Trade id="FX_VarianceSwap_Corridor">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    .....
  </Envelope>
  <CorridorVarianceSwapData>
    <LongShort type="longShort">Long</LongShort>
    <Strike type="number">0.02</Strike>
    <Notional type="number">50000</Notional>
    <Underlying type="index">FX-ECB-EUR-USD</Underlying>
    <ValuationSchedule type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2018-12-31</StartDate>
          <EndDate>2019-05-05</EndDate>
          <Tenor>1D</Tenor>
          <Convention>F</Convention>
          <TermConvention>F</TermConvention>
          <Calendar>US</Calendar>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
    </ValuationSchedule>
    <SquaredPayoff type="bool">true</SquaredPayoff>
    <UpperBarrierLevel type="number">1000000000</UpperBarrierLevel>
    <LowerBarrierLevel type="number">0</LowerBarrierLevel>
    <CountBothObservations type="bool">true</CountBothObservations>
    <AccrualAdjustment type="bool">false</AccrualAdjustment>
    <Cap type="number">2.5</Cap>
    <Floor type="number">0</Floor>
    <SettlementDate type="event">2019-05-06</SettlementDate>
    <PayCcy type="currency">USD</PayCcy>
  </CorridorVarianceSwapData>
</Trade>
```

The CorridorVarianceSwap script referenced in the trade above is shown in listing 183

The payout formula is:

$$Payout = 100^2 * varianceAmount * \left[\min\left(\max(realisedVariance^2, \texttt{Floor}), \texttt{Cap} \right) - \texttt{Strike}^2 \right],$$

where the $realisedVariance$ only consists of variance accrual contributions for when the underlying price/level was trading within the window defined by the corridor.

The meanings and allowable values for the CorridorVarianceSwapData node below.

- [longShort] LongShort: Own party position in the swap. *Long* corresponds to paying out on the fixed/strike variance (volatility) and receiving on the floating/realised variance (volatility). In other words, a long position has positive value if the realised variance (volatility) exceeds the variance (volatility) strike.
  Allowable values: *Long, Short.*

- [number] `Strike`: The volatility strike K_{vol} of the variance swap quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.
- [number] `Notional`: The vega notional amount. If the swap was struck in terms of a variance notional N_{var}, the corresponding vega notional is given by $N_{vol} = N_{var} \cdot 2 \cdot 100 \cdot K_{vol}$ (where K_{vol} is in absolute terms).
  Allowable values: Any non-negative number.
- [index] `Underlying`: Underlying index.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [event] `ValuationSchedule`: The schedule defining the (daily) observation period for the variance accrual.
  Allowable values: See Section 19.3.4.
- [bool] `SquaredPayoff`: Flag indicating whether the trade is a variance swap (*True*) or a volatility swap (*False*).
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.
- [number] `UpperBarrierLevel`: The agreed upper barrier price level.
  Allowable values: Any non-negative real number.
- [number] `LowerBarrierLevel`: The agreed lower barrier price level.
  Allowable values: Any non-negative real number.
- [bool] `CountBothObservations`: Whether the variance/volatility is accrued based on both the current and previous underlying prices falling within the corridor (*True*) or only on the previous underlying price (*False*).
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.
- [bool] `AccrualAdjustment`: Whether the strike will be scaled relative to the number of days that the underlying traded within the corridor. See $\widetilde{K}_{var}$ (for variance swaps) and $\widetilde{K}_{vol}$ (for volatility swaps) in section Capped/Floored Corridor Variance Swap of the Product Description for Exotic Variance and Volatility Swaps. Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc.
- [number] `Cap`: The cap on the realised variance (or volatility), as a factor of the `Strike` after the accrual adjustment is applied (only when `AccrualAdjustment` is *True*). For example, if `Cap` is 2.5, then the cap level will be $2.5^2 \times \widetilde{K}_{var}$ for variance swaps, and $2.5 \times \widetilde{K}_{vol}$ for volatility swaps. For trades with no cap, set `Cap` to zero.
  Allowable values: Any non-negative number.
- [number] `Floor`: The floor on the realised variance (or volatility), as a factor of the `Strike` after the accrual adjustment is applied (only when `AccrualAdjustment` is *True*). For example, if `Floor` is 0.1, then the floor level will be $0.1^2 \times \widetilde{K}_{var}$ for variance swaps, and $0.1 \times \widetilde{K}_{var}$ for volatility swaps. For trades with no floor, set `Floor` to zero.
  Allowable values: Any non-negative number. The full set of allowable values is given in Table 36.

- [event] `SettlementDate`: The date on which the swap payoff is settled.
  Allowable values: See `Date` in Table 20.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Corridor Variance Swap with KI/KO Barrier

The traditional trade representation is as follows, using an EQ underlying in this example:

```
<Trade id="EQ_VarianceSwap_KIKO_Corridor">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
     .....
  </Envelope>
  <KIKOCorridorVarianceSwapData>
    <LongShort type="longShort">Long</LongShort>
    <Strike type="number">0.19</Strike>
    <Notional type="number">1380</Notional>
    <Underlying type="index">EQ-RIC:.SPX</Underlying>
    <ValuationSchedule type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2020-11-26</StartDate>
          <EndDate>2021-09-18</EndDate>
          <Tenor>1D</Tenor>
          <Convention>Following</Convention>
          <TermConvention>Following</TermConvention>
          <Calendar>USA</Calendar>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
    </ValuationSchedule>
    <CorridorUpperBarrierLevel type="number">3469.18</CorridorUpperBarrierLevel>
    <CorridorLowerBarrierLevel type="number">2207.66</CorridorLowerBarrierLevel>
    <KIKOBarrierType type="barrierType">UpOut</KIKOBarrierType>
    <KIKOBarrierLevel type="number">3469.18</KIKOBarrierLevel>
    <CountBothObservations type="bool">true</CountBothObservations>
    <AccrualAdjustment type="bool">true</AccrualAdjustment>
    <Cap type="number">0</Cap>
    <Floor type="number">0</Floor>
    <SettlementSchedule type="event">
      <DerivedSchedule>
        <BaseSchedule>ValuationSchedule</BaseSchedule>
        <Shift>2D</Shift>
        <Calendar>USA</Calendar>
        <Convention>Following</Convention>
      </DerivedSchedule>
    </SettlementSchedule>
    <PayCcy type="currency">USD</PayCcy>
  </KIKOCorridorVarianceSwapData>
</Trade>
```

The KIKOCorridorVarianceSwap script referenced in the trade above is shown in listing 182

The meanings and allowable values for the `KIKOCorridorVarianceSwapData` node below.

- [longShort] `LongShort`: Own party position in the swap. *Long* corresponds to paying out on the fixed/strike variance and receiving on the floating/realised variance. In other words, a long position has positive value if the realised variance exceeds the variance strike.
  Allowable values: *Long, Short*.

- [number] `Strike`: The volatility strike K_{vol} of the variance swap quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.
- [number] `Notional`: The vega notional amount. If the swap was struck in terms of a variance notional N_{var}, the corresponding vega notional is given by $N_{vol} = N_{var} \cdot 2 \cdot 100 \cdot K_{vol}$ (where K_{vol} is in absolute terms).
  Allowable values: Any non-negative number.
- [index] `Underlying`: Underlying index. Currently only EQ, FX and COMM underlyings are supported.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [event] `ValuationSchedule`: The schedule defining the (daily) observation period for the variance accrual.
  Allowable values: See Section 19.3.4.
- [number] `CorridorUpperBarrierLevel`: The agreed upper barrier price level for the corridor.
  Allowable values: Any non-negative real number.
- [number] `CorridorLowerBarrierLevel`: The agreed lower barrier price level for the corridor.
  Allowable values: Any non-negative real number.
- [barrierType] `KIKOBarrierType`: Whether the barrier is a knock-in (*DownIn* or *UpIn*) or a knock-out (*DownOut* or *UpOut*) barrier. For trades with no barrier, which is equivalent to a capped/floored corridor variance swap, set `KIKOBarrierType` to *UpIn* and `KIKOBarrierLevel` to zero, as in the sample trade representation above.
  Allowable values: *DownIn, UpIn, DownOut, UpOut*.
- [number] `KIKOBarrierLevel`: The agreed knock-in/knock-out barrier price level.
  Allowable values: Any non-negative real number.
- [bool] `CountBothObservations`: Whether the variance is accrued based on both the current and previous underlying prices falling within the corridor (*True*) or only on the previous underlying price (*False*).
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.
- [bool] `AccrualAdjustment`: Whether the strike will be scaled relative to the number of days that the underlying traded within the corridor. See $\widetilde{K}_{var}$ in section Capped/Floored Corridor Variance Swap of the Product Description for Exotic Variance and Volatility Swaps. Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc.
- [number] `Cap`: The cap on the realised variance, as a factor of the `Strike`. For example, if `Cap` is 2.5, then the cap level will be $2.5^2 \times$ `Strike`^{2}. For trades with no cap, set `Cap` to zero.
  Allowable values: Any non-negative number.
- [number] `Floor`: The floor on the realised variance, as a factor of the `Strike`. For

example, if `Floor` is 0.1, then the floor level will be $0.1^2 \times$ `Strike`. For trades with no floor, set `Floor` to zero.
Allowable values: Any non-negative number.

- [event] `SettlementSchedule`: The settlement dates derived from the `ValuationSchedule` using a settlement lag to reflect settlement after a knock-out event, or after the final variance observation date if no knock-out occurs. If the barrier is knock-in, this represents the settlement date, which is the final valuation date plus the settlement lag.
  Allowable values: See section 19.3.4 Schedule Data and Dates, or DerivedSchedule (see the scripted trade documentation in vre/docs/scripted_trade).
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Conditional Variance Swap 01

The traditional trade representation is as follows, using an FX underlying in this example:

```
<Trade id="FX_VarianceSwap_Conditional_01">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    .....
  </Envelope>
  <ConditionalVarianceSwap01Data>
    <LongShort type="longShort">Long</LongShort>
    <Strike type="number">0.02</Strike>
    <Notional type="number">50000</Notional>
    <Underlying type="index">FX-ECB-EUR-USD</Underlying>
    <ValuationSchedule type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2018-12-31</StartDate>
          <EndDate>2019-05-05</EndDate>
          <Tenor>1D</Tenor>
          <Convention>F</Convention>
          <TermConvention>F</TermConvention>
          <Calendar>US</Calendar>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
    </ValuationSchedule>
    <SquaredPayoff type="bool">true</SquaredPayoff>
    <BarrierType type="barrierType">UpIn</BarrierType>
    <BarrierLevel type="number">0</BarrierLevel>
    <CountBothObservations type="bool">true</CountBothObservations>
    <Cap type="number">2.5</Cap>
    <Floor type="number">0</Floor>
    <AccrualAdjustment type="bool">false</AccrualAdjustment>
    <SettlementDate type="event">2019-05-06</SettlementDate>
    <PayCcy type="currency">USD</PayCcy>
  </ConditionalVarianceSwap01Data>
</Trade>
```

The ConditionalVarianceSwap01 script referenced in the trade above is shown in listing 185

The payout formula is:

$$Payout = 100^2 * varianceAmount * \left[\min\left(\max(realisedVariance^2, \texttt{Floor}), \texttt{Cap} \right) - \texttt{Strike}^2 \right],$$

where the $realisedVariance$ only consists of variance accrual contributions for when the underlying price/level was trading within the pre-defined window.

The meanings and allowable values for the `ConditionalVarianceSwapData01` node below.

- [longShort] `LongShort`: Own party position in the swap. *Long* corresponds to paying out on the fixed/strike variance (volatility) and receiving on the floating/realised variance (volatility). In other words, a long position has positive value if the realised variance (volatility) exceeds the variance (volatility) strike.
  Allowable values: *Long, Short*.
- [number] `Strike`: The volatility strike K_{vol} of the variance swap quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.
- [number] `Notional`: The vega notional amount. If the swap was struck in terms of a variance notional N_{var}, the corresponding vega notional is given by $N_{vol} = N_{var} \cdot 2 \cdot 100 \cdot K_{vol}$ (where K_{vol} is in absolute terms).
  Allowable values: Any non-negative number.
- [index] `Underlying`: Underlying index.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [event] `ValuationSchedule`: The schedule defining the (daily) observation period for the variance accrual.
  Allowable values: See Section 19.3.4.
- [bool] `SquaredPayoff`: Flag indicating whether the trade is a variance swap (*True*) or a volatility swap (*False*).
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.
- [barrierType] `BarrierType`: Whether the instrument is an up-variance swap (*UpIn* or *DownOut*) or a down-variance swap (*DownIn* or *UpOut*).
  Allowable values: *DownIn, UpIn, DownOut, UpOut*.
- [number] `BarrierLevel`: The agreed barrier price level.
  Allowable values: Any non-negative real number.
- [bool] `CountBothObservations`: Whether the variance/volatility is accrued based on both the current and previous underlying prices falling within the range (*True*) or only on the previous underlying price (*False*).
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.
- [bool] `AccrualAdjustment`: Whether the strike will be scaled relative to the number of days that the underlying traded within the range. See $\widetilde{K}_{var}$ (for variance swaps) and $\widetilde{K}_{vol}$ (for volatility swaps) in section Capped/Floored Conditional Variance Swap of the Product Description for Exotic Variance and Volatility Swaps. Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc.
- [number] `Cap`: The cap on the realised variance (or volatility), as a factor of the

`Strike` after the accrual adjustment is applied (only when `AccrualAdjustment` is *True*). For example, if `Cap` is 2.5, then the cap level will be $2.5^2 \times \widetilde{K}_{var}$ for variance swaps, and $2.5 \times \widetilde{K}_{vol}$ for volatility swaps. For trades with no cap, set `Cap` to zero. Allowable values: Any non-negative number.

- [number] `Floor`: The floor on the realised variance (or volatility), as a factor of the `Strike` after the accrual adjustment is applied (only when `AccrualAdjustment` is *True*). For example, if `Floor` is 0.1, then the floor level will be $0.1^2 \times \widetilde{K}_{var}$ for variance swaps, and $0.1 \times \widetilde{K}_{var}$ for volatility swaps. For trades with no floor, set `Floor` to zero. Allowable values: Any non-negative number. The full set of allowable values is given in Table 36.
- [event] `SettlementDate`: The date on which the swap payoff is settled. Allowable values: See `Date` in Table 20.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade. Allowable values: See Table 22 for allowable currency codes.

Conditional Variance Swap 02

The traditional trade representation is as follows, using an FX underlying in this example:

```
<Trade id="FX_VarianceSwap_Conditional_02">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    .....
  </Envelope>
  <ConditionalVarianceSwap02Data>
    <LongShort type="longShort">Long</LongShort>
    <Strike type="number">0.02</Strike>
    <Notional type="number">50000</Notional>
    <VarianceReference type="number">0.03</VarianceReference>
    <Underlying type="index">FX-ECB-EUR-USD</Underlying>
    <ValuationSchedule type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2018-12-31</StartDate>
          <EndDate>2019-05-05</EndDate>
          <Tenor>1D</Tenor>
          <Convention>F</Convention>
          <TermConvention>F</TermConvention>
          <Calendar>US</Calendar>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
    </ValuationSchedule>
    <SquaredPayoff type="bool">true</SquaredPayoff>
    <BarrierType type="barrierType">UpIn</BarrierType>
    <BarrierLevel type="number">0</BarrierLevel>
    <CountBothObservations type="bool">true</CountBothObservations>
    <Cap type="number">2.5</Cap>
    <Floor type="number">0</Floor>
    <AccrualAdjustment type="bool">false</AccrualAdjustment>
    <SettlementDate type="event">2019-05-06</SettlementDate>
    <PayCcy type="currency">USD</PayCcy>
  </ConditionalVarianceSwap02Data>
</Trade>
```

The ConditionalVarianceSwap02 script referenced in the trade above is shown in listing 186

The payout formula is:

$$Payout = 100^2 * varianceAmount * \left[\min\left(\max(realisedVariance^2, \texttt{Floor}), \texttt{Cap} \right) - \texttt{Strike}^2 \right],$$

where the $realisedVariance$ only consists of variance accrual contributions for when the underlying price/level was trading within the pre-defined window, and the "variance strike" used to obtain the varianceAmount is given by the variance reference strike (instead of the actual strike).

The meanings and allowable values for the `ConditionalVarianceSwapData02` node below.

- [longShort] `LongShort`: Own party position in the swap. *Long* corresponds to paying out on the fixed/strike variance (volatility) and receiving on the floating/realised variance (volatility). In other words, a long position has positive value if the realised variance (volatility) exceeds the variance (volatility) strike.
  Allowable values: *Long, Short*.
- [number] `Strike`: The volatility strike K_{vol} of the variance swap quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.
- [number] `Notional`: The vega notional amount. If the swap was struck in terms of a variance notional N_{var}, the corresponding vega notional is given by $N_{vol} = N_{var} \cdot 2 \cdot 100 \cdot K_{vol}$ (where K_{vol} is in absolute terms).
  Allowable values: Any non-negative number.
- [number] `VarianceReference`: The (volatility) strike value used (in place of the actual strike) to scale down the vega notional in order to obtain the variance amount. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any non-negative number.
- [index] `Underlying`: Underlying index.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [event] `ValuationSchedule`: The schedule defining the (daily) observation period for the variance accrual.
  Allowable values: See Section 19.3.4.
- [bool] `SquaredPayoff`: Flag indicating whether the trade is a variance swap (*True*) or a volatility swap (*False*).
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.
- [barrierType] `BarrierType`: Whether the instrument is an up-variance swap (*UpIn* or *DownOut*) or a down-variance swap (*DownIn* or *UpOut*).
  Allowable values: *DownIn, UpIn, DownOut, UpOut*.
- [number] `BarrierLevel`: The agreed barrier price level.
  Allowable values: Any non-negative real number.

- [bool] `CountBothObservations`: Whether the variance/volatility is accrued based on both the current and previous underlying prices falling within the range (*True*) or only on the previous underlying price (*False*).
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.
- [bool] `AccrualAdjustment`: Whether the strike will be scaled relative to the number of days that the underlying traded within the range. See $\widetilde{K}_{var}$ (for variance swaps) and $\widetilde{K}_{vol}$ (for volatility swaps) in section Capped/Floored Conditional Variance Swap of the Product Description for Exotic Variance and Volatility Swaps. Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc.
- [number] `Cap`: The cap on the realised variance (or volatility), as a factor of the `Strike` after the accrual adjustment is applied (only when `AccrualAdjustment` is *True*). For example, if `Cap` is 2.5, then the cap level will be $2.5^2 \times \widetilde{K}_{var}$ for variance swaps, and $2.5 \times \widetilde{K}_{vol}$ for volatility swaps. For trades with no cap, set `Cap` to zero.
  Allowable values: Any non-negative number.
- [number] `Floor`: The floor on the realised variance (or volatility), as a factor of the `Strike` after the accrual adjustment is applied (only when `AccrualAdjustment` is *True*). For example, if `Floor` is 0.1, then the floor level will be $0.1^2 \times \widetilde{K}_{var}$ for variance swaps, and $0.1 \times \widetilde{K}_{var}$ for volatility swaps. For trades with no floor, set `Floor` to zero.
  Allowable values: Any non-negative number. The full set of allowable values is given in Table 36.
- [event] `SettlementDate`: The date on which the swap payoff is settled.
  Allowable values: See `Date` in Table 20.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Pairwise Variance Swap

The `FxPairwiseVarianceSwap` and `EquityPairwiseVarianceSwap` trade types have trade data containers (respectively):

- `FxPairwiseVarianceSwapData`
- `EquityPairwiseVarianceSwapData`

Listing 187 shows the structure of example trades for an Equity underlying.

Listing 187: EquityPairwiseVarianceSwap data

```
<Trade id="EQ_VarianceSwap_Pairwise">
  <TradeType>EquityPairwiseVarianceSwap</TradeType>
  <Envelope>
    .......
  </Envelope>
  <EquityPairwiseVarianceSwapData>
    <LongShort>Long</LongShort>
    <Underlyings>
      <Value>EQ-RIC:.STOXX50E</Value>
      <Value>EQ-RIC:.SPX</Value>
    </Underlyings>
    <UnderlyingStrikes>
      <Value>0.33859119894055129</Value>
      <Value>0.39039467209479178</Value>
    </UnderlyingStrikes>
    <UnderlyingNotionals>
      <Value>577.75847822419905</Value>
      <Value>603.54235516510619</Value>
    </UnderlyingNotionals>
    <BasketNotional>1339.2898822637151</BasketNotional>
    <BasketStrike>0.31612972020991642</BasketStrike>
    <ValuationSchedule>
      <Rules>
      .......
      </Rules>
    </ValuationSchedule>
    <LaggedValuationSchedule>
      <Derived>
       .......
      </Derived>
    </LaggedValuationSchedule>
    <AccrualLag>3</AccrualLag>
    <PayoffLimit>5</PayoffLimit>
    <Cap>2.5</Cap>
    <Floor>0</Floor>
    <SettlementDate>2022-06-29</SettlementDate>
    <PayCcy>USD</PayCcy>
  </EquityPairwiseVarianceSwapData>
</Trade>
```

The payout formula is:

$$Payout = \min\Big(\max(equityAmount1 + equityAmount2 + equityAmountBasket, LowerLimit), UpperLimi$$

where

$$\begin{aligned} UpperLimit &= \texttt{PayoffLimit} * \big(|notional1| + |notional2|\big) \\ LowerLimit &= - \texttt{PayoffLimit} * \big(|notional1| + |notional2|\big) \end{aligned}$$

The meanings and allowable values of elements in the `EquityPairwiseVarianceSwapData` node follow below.

- `LongShort`: Own party position.
  Allowable values: *Long*, *Short*
- `Underlyings`: The basket of underlyings.

Allowable values: The format follows that of a scripted trade underlying. See the scripted trade documentation in vre/docs/scripted_trade (section Data Node / Index) for allowable values.

- `UnderlyingStrikes`: The volatility strikes $K_{vol}^{1,2}$ of the underlyings quoted in absolute terms. If the swap was struck in terms of variance, the square roots of the variances should be used here.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.

- `UnderlyingNotionals`: The vega notional amount. If the swap was struck in terms of variance notionals N_{var}, the corresponding vega notionals are given by $N_{vol}^{1,2} = N_{var}^{1,2} \cdot 2 \cdot 100 \cdot K_{vol}^{1,2}$ (where $K_{vol}^{1,2}$ is in absolute terms).
  Allowable values: Any non-negative number.

- `BasketNotional`: The basket vega notional amount.
  Allowable values: Any non-negative number.

- `ValuationSchedule`: The start dates of the variance accrual schedule. This node can also be used to derive the variance accrual schedule (or vice versa) using a DerivedSchedule (e.g. with a two-day lag, *-2D*). For the standard accrual period lengths of one day, set `Shift` to *-1D*.
  Allowable values: See Section 19.3.4. If this uses a DerivedSchedule, the `Shift` must be less than or equal to *-1D*.

- `LaggedValuationSchedule`: The end dates of the variance accrual schedule. This node can be derived from the `ValuationSchedule` (or vice versa) using a DerivedSchedule (e.g. with a two-day lag, *2D*). For the standard accrual period lengths of one day, set `Shift` to *1D*.
  Allowable values: See section 19.3.4. If this uses a DerivedSchedule, the `Shift` must be greater than or equal to *1D*. If left blank or omitted, defaults to a derived schedule, with `Shift` equal to *1D*.

- `AccrualLag`: The length (in days) of each variance accrual period. For classic variance swaps, set this to *1*.
  Allowable values: Any integer greater than or equal to *1*. If left blank or omitted, defaults to *1*.

- `PayoffLimit`: The factor used to determine the maximum/minimum payoff under the swap. This corresponds to C in the final equation in the Product Description for Pairwise Variance Swap.
  Allowable values: Any non-negative number. If left blank or omitted, defaults to zero, i.e. no payoff limit.

- `Cap`: The cap on the realised variances, as a factor of the corresponding strikes. For example, if `Cap` is 2.5, then the cap level will be $2.5^2 \times \hat{K}_{var}$ for the basket realised variance. For trades with no cap, set `Cap` to zero.
  Allowable values: Any non-negative number.

- `Floor`: The floor on the realised variances, as a factor of the corresponding strikes. For example, if `Floor` is 0.1, then the floor level will be $0.1^2 \times \hat{K}_{var}$ for the basket realised variance. For trades with no floor, set `Floor` to zero.
  Allowable values: Any non-negative number.

- `SettlementDate`: The date on which the swap payoff is settled.
  Allowable values: See `Date` in Table 20.
- `PayCcy`: The settlement currency.
  Allowable values: See Table 22 `Currency`.

Pairwise variance swaps can alternatively be represented as *scripted trades*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction.

```
<Trade id="EQ_VarianceSwap_Pairwise">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    .....
  </Envelope>
  <PairwiseVarianceSwapData>
    <LongShort type="longShort">Long</LongShort>
    <Underlyings type="index">
      <Value>EQ-RIC:.STOXX50E</Value>
      <Value>EQ-RIC:.SPX</Value>
    </Underlyings>
    <UnderlyingStrikes type="number">
      <Value>0.3385911989405513</Value>
      <Value>0.3903946720947918</Value>
    </UnderlyingStrikes>
    <UnderlyingNotionals type="number">
      <Value>577.7584782241991</Value>
      <Value>603.5423551651062</Value>
    </UnderlyingNotionals>
    <BasketNotional type="number">1339.289882263715</BasketNotional>
    <BasketStrike type="number">0.3161297202099164</BasketStrike>
    <ValuationSchedule type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2021-06-25</StartDate>
          <EndDate>2022-06-25</EndDate>
          <Tenor>3D</Tenor>
          <Convention>F</Convention>
          <TermConvention>F</TermConvention>
          <Calendar>US</Calendar>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
    </ValuationSchedule>
    <LaggedValuationSchedule type="event">
      <DerivedSchedule>
        <BaseSchedule>ValuationSchedule</BaseSchedule>
        <Shift>2D</Shift>
        <Calendar>US</Calendar>
        <Convention>F</Convention>
      </DerivedSchedule>
    </LaggedValuationSchedule>
    <AccrualLag type="number">3</AccrualLag>
    <PayoffLimit type="number">5</PayoffLimit>
    <Cap type="number">2.5</Cap>
    <Floor type="number">0</Floor>
    <SettlementDate type="event">2022-06-29</SettlementDate>
    <PayCcy type="currency">USD</PayCcy>
  </PairwiseVarianceSwapData>
</Trade>
```

The PairwiseVarianceSwap script referenced in the trade above is shown in listing 188

The meanings and allowable values for the `PairwiseVarianceSwapData` node below.

- [longShort] `LongShort`: Own party position in the swap. *Long* corresponds to paying out on the fixed/strike variance (volatility) and receiving on the floating/realised variance (volatility). In other words, a long position has positive value if the realised variance (volatility) exceeds the variance (volatility) strike.

Allowable values: *Long, Short*.

- [number] `UnderlyingStrikes`: The volatility strikes $K_{vol}^{1,2}$ of the underlyings quoted in absolute terms. If the swap was struck in terms of variance, the square roots of the variances should be used here.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.

- [number] `UnderlyingNotionals`: The vega notional amount. If the swap was struck in terms of variance notionals N_{var}, the corresponding vega notionals are given by $N_{vol}^{1,2} = N_{var}^{1,2} \cdot 2 \cdot 100 \cdot K_{vol}^{1,2}$ (where $K_{vol}^{1,2}$ is in absolute terms).
  Allowable values: Any non-negative number.

- [index] `Underlyings`: Underlying index.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.

- [number] `BasketStrike`: The basket volatility strike $\hat{K}_{vol}$ quoted in absolute terms.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.

- [number] `BasketNotional`: The basket vega notional amount.
  Allowable values: Any non-negative number.

- [event] `ValuationSchedule`: The base schedule which defines the start dates of the variance accrual schedule. This node is also used to derive the variance accrual schedule.
  Allowable values: See Section 19.3.4.

- [event] `LaggedValuationSchedule`: The end dates of the variance accrual schedule. This can be derived from the `Schedule` using a DerivedSchedule (e.g. with a two-day lag, *2D*). For the standard accrual period lengths of one day, set `Shift` to *1D*.
  Allowable values: See section 19.3.4 Schedule Data and Dates, or DerivedSchedule (see the scripted trade documentation in vre/docs/scripted_trade).

- [number] `AccrualLag`: The length (in days) of each variance accrual period. For classic variance swaps with no lag, set this to 1.
  Allowable values: Any integer greater than or equal to 1.

- [number] `PayoffLimit`: The factor used to determine the maximum/minimum payoff under the swap. This corresponds to C in the final equation in the Product Description for Pairwise Variance Swap.
  Allowable values: Any non-negative number.

- [number] `Cap`: The cap on the realised variances, as a factor of the corresponding strikes. For example, if `Cap` is 2.5, then the cap level will be $2.5^2 \times \hat{K}_{var}$ for the basket realised variance. For trades with no cap, set `Cap` to zero.
  Allowable values: Any non-negative number.

- [number] `Floor`: The floor on the realised variances, as a factor of the corresponding strikes. For example, if `Floor` is 0.1, then the floor level will be $0.1^2 \times \hat{K}_{var}$ for the basket realised variance. For trades with no floor, set `Floor` to zero.
  Allowable values: Any non-negative number.

- [event] `SettlementDate`: The date on which the swap payoff is settled.

Allowable values: See `Date` in Table 20.

- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade. Allowable values: See Table 22 for allowable currency codes.

Variance Dispersion Swap

The traditional trade representation is as follows, using EQ underlyings in this example:

```
<Trade id="EQ_VarianceDispersionSwap">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    .....
  </Envelope>
  <VarianceDispersionSwapData>
    <LongShort type="longShort">Long</LongShort>
    <Underlyings1 type="index">
      <Value>EQ-RIC:HSBA.L</Value>
      <Value>EQ-RIC:MSFT.OQ</Value>
      <Value>EQ-RIC:AMZN.O</Value>
    </Underlyings1>
    <Weights1 type="number">
        .....
    </Weights1>
    <Strikes1 type="number">
        .....
    </Strikes1>
    <Spreads1 type="number">
        .....
    </Spreads1>
    <Notionals1 type="number">
        .....
    </Notionals1>
    <Caps1 type="number">
        .....
    </Caps1>
    <Floors1 type="number">
        .....
    </Floors1>
    <Underlyings2 type="index">
      <Value>EQ-RIC:.STOXX50E</Value>
      <Value>EQ-RIC:.SPX</Value>
    </Underlyings2>
    <Weights2 type="number">
        .....
    </Weights2>
    <Strikes2 type="number">
        .....
    </Strikes2>
    <Spreads2 type="number">
        .....
    </Spreads2>
    <Notionals2 type="number">
        .....
    </Notionals2>
    <Caps2 type="number">
        .....
    </Caps2>
    <Floors2 type="number">
        .....
    </Floors2>
    <ValuationSchedule type="event">
      <ScheduleData>
        .....
      </ScheduleData>
    </ValuationSchedule>
```

```
    <DividendAdjustment type="bool">false</DividendAdjustment>
    <SettlementDate type="event">2021-01-10</SettlementDate>
    <PayCcy type="currency">USD</PayCcy>
  </VarianceDispersionSwapData>
</Trade>
```

The VarianceDispersionSwap script referenced in the trade above is shown in listing 189

The meanings and allowable values for the `VarianceDispersionSwapData` node below.

- [longShort] `LongShort`: Own party position in the swap. *Long* corresponds to selling volatility on the `Underlyings1` basket and buying volatility on `Underlyings2`.
  Allowable values: *Long, Short*.
- [index] `Underlyings1`: The basket of underlyings whose volatility is bought in the *Long* position.
  Allowable values: For each underlying, see 19.3.29.
- [number] `Weights1`: List of weights applied to the final realised volatility of each underlying in the `Underlyings1` basket.
  Allowable values: Any positive number.
- [number] `Strikes1`: The volatility strike $K_{1,u,vol}$ of the variance swap for each underlying u in the `Underlyings1` basket, quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any positive number, as a percentage expressed in decimal form.
- [number] `Spreads1`: Additional spread to the realised variance, for each underlying in the `Underlyings1` basket.
  Allowable values: Any real number.
- [number] `Notionals1`: For each underlying u in the `Underlyings1` basket, the vega notional amount. If the swap was struck in terms of variance notionals $N_{1,u,var}$, the corresponding vega notionals are given by $N_{1,u,vol} = N_{1,u,var} \cdot 2 \cdot 100 \cdot K_{1,u,vol}$ (where $K_{1,u,vol}$ is in absolute terms).
  Allowable values: Any real number.
- [number] `Caps1`: For each underlying u in the `Underlyings1` basket, the cap on the realised variance as a factor of the strike. For example, if the value in `Caps1` is 2.5, then the cap level will be $2.5^2 \times K_{1,u,\text{var}}$. For underlyings with no cap, set the value in `Caps1` to zero.
  Allowable values: Any non-negative number.
- [number] `Floors1`: For each underlying u in the `Underlyings1` basket, the floor on the realised variance as a factor of the strike. For example, if the value in `Floors1` is 0.1, then the floor level will be $0.1^2 \times K_{1,u,\text{var}}$. For underlyings with no floor, set the value in `Floors1` to zero.
  Allowable values: Any non-negative number.
- [index] `Underlyings2`: The basket of underlyings whose volatility is sold in the *Long* position.
  Allowable values: For each underlying, see 19.3.29.
- [number] `Weights2`: List of weights applied to the final realised volatility of each underlying in the `Underlyings2` basket.

Allowable values: Any positive number.

- [number] `Strikes2`: The volatility strike $K_{2,u,vol}$ of the variance swap for each underlying u in the `Underlyings2` basket, quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any positive number, as a percentage expressed in decimal form.
- [number] `Spreads2`: Additional spread to the realised variance, for each underlying in the `Underlyings2` basket.
  Allowable values: Any real number.
- [number] `Notionals2`: For each underlying u in the `Underlyings2` basket, the vega notional amount. If the swap was struck in terms of variance notionals $N_{2,u,var}$, the corresponding vega notionals are given by $N_{2,u,vol} = N_{2,u,var} \cdot 2 \cdot 100 \cdot K_{2,u,vol}$ (where $K_{2,u,vol}$ is in absolute terms).
  Allowable values: Any real number.
- [number] `Caps2`: For each underlying u in the `Underlyings2` basket, the cap on the realised variance as a factor of the strike. For example, if the value in `Caps2` is 2.5, then the cap level will be $2.5^2 \times K_{2,u,\text{var}}$. For underlyings with no cap, set the value in `Caps2` to zero.
  Allowable values: Any non-negative number.
- [number] `Floors2`: For each underlying u in the `Underlyings2` basket, the floor on the realised variance as a factor of the strike. For example, if the value in `Floors2` is 0.1, then the floor level will be $0.1^2 \times K_{2,u,\text{var}}$. For underlyings with no floor, set the value in `Floors2` to zero.
  Allowable values: Any non-negative number.
- [bool] `DividendAdjustment`: Whether or not the underlying price is adjusted for dividend payments. This feature is not yet supported. Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.
- [event] `ValuationSchedule`: The schedule defining the (daily) observation period for the variance accrual.
  Allowable values: See Section 19.3.4.
- [event] `SettlementDate`: The date on which the swap payoff is settled.
  Allowable values: See `Date` in Table 20.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Corridor Variance Dispersion Swap

This instrument is a variance dispersion swap with a corridor feature, where variance is accrued only when the the price of the first underlying of each pair of underlyings (between the first and second basket) falls within the 'corridor'. This means that the number of underlyings in each basket should be the same, and each pair is determined based on the order that they are provided in the trade XML, as in the example below.

The traditional trade representation is as follows, using EQ underlyings in this example:

```
<Trade id="EQ_VarianceDispersionSwap_Corridor">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    .....
  </Envelope>
  <CorridorVarianceDispersionSwapData>
    <LongShort type="longShort">Long</LongShort>
    <Weights type="number">
        .....
    </Weights>
    <Underlyings1 type="index">
      <Value>EQ-RIC:HSBA.L</Value>
      <Value>EQ-RIC:MSFT.OQ</Value>
      <Value>EQ-RIC:AMZN.O</Value>
    </Underlyings1>
    <Strikes1 type="number">
        .....
    </Strikes1>
    <Spreads1 type="number">
        .....
    </Spreads1>
    <Notionals1 type="number">
        .....
    </Notionals1>
    <Caps1 type="number">
        .....
    </Caps1>
    <Floors1 type="number">
        .....
    </Floors1>
    <Underlyings2 type="index">
      <Value>EQ-RIC:.STOXX50E</Value>
      <Value>EQ-RIC:.SPX</Value>
      <Value>EQ-RIC:.SPX</Value>
    </Underlyings2>
    <Strikes2 type="number">
        .....
    </Strikes2>
    <Spreads2 type="number">
        .....
    </Spreads2>
    <Notionals2 type="number">
        .....
    </Notionals2>
    <Caps2 type="number">
        .....
    </Caps2>
    <Floors2 type="number">
        .....
    </Floors2>
    <UpperBarrierLevels type="number">
        .....
    </UpperBarrierLevels>
    <LowerBarrierLevels type="number">
        .....
    </LowerBarrierLevels>
    <CountBothObservations type="bool">true</CountBothObservations>
    <AccrualAdjustment type="bool">true</AccrualAdjustment>
    <DividendAdjustment type="bool">false</DividendAdjustment>
    <ValuationSchedule type="event">
      <ScheduleData>
        .....
      </ScheduleData>
    </ValuationSchedule>
    <SettlementDate type="event">2021-01-10</SettlementDate>
    <PayCcy type="currency">USD</PayCcy>
  </CorridorVarianceDispersionSwapData>
</Trade>
```

The CorridorVarianceDispersionSwap script referenced in the trade above is shown in listing

190

The meanings and allowable values for the `CorridorVarianceDispersionSwapData` node below.

- [longShort] `LongShort`: Own party position in the swap. *Long* corresponds to selling volatility on the `Underlyings1` basket and buying volatility on `Underlyings2`.
  Allowable values: *Long, Short*.
- [number] `Weights`: List of weights applied to the final realised volatility of each underlying in the `Underlyings1` and `Underlyings2` baskets.
  Allowable values: Any positive number.
- [index] `Underlyings1`: The basket of underlyings whose volatility is bought in the *Long* position.
  Allowable values: For each underlying, see 19.3.29.
- [number] `Strikes1`: The volatility strike $K_{1,u,vol}$ of the variance swap for each underlying u in the `Underlyings1` basket, quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any positive number, as a percentage expressed in decimal form.
- [number] `Spreads1`: Additional spread to the realised variance, for each underlying in the `Underlyings1` basket.
  Allowable values: Any real number.
- [number] `Notionals1`: For each underlying u in the `Underlyings1` basket, the vega notional amount. If the swap was struck in terms of variance notionals $N_{1,u,var}$, the corresponding vega notionals are given by $N_{1,u,vol} = N_{1,u,var} \cdot 2 \cdot 100 \cdot K_{1,u,vol}$ (where $K_{1,u,vol}$ is in absolute terms).
  Allowable values: Any real number.
- [number] `Caps1`: For each underlying u in the `Underlyings1` basket, the cap on the realised variance as a factor of the strike. For example, if the value in `Caps1` is 2.5, then the cap level will be $2.5^2 \times K_{1,u,\text{var}}$. For underlyings with no cap, set the value in `Caps1` to zero.
  Allowable values: Any non-negative number.
- [number] `Floors1`: For each underlying u in the `Underlyings1` basket, the floor on the realised variance as a factor of the strike. For example, if the value in `Floors1` is 0.1, then the floor level will be $0.1^2 \times K_{1,u,\text{var}}$. For underlyings with no floor, set the value in `Floors1` to zero.
  Allowable values: Any non-negative number.
- [index] `Underlyings2`: The basket of underlyings whose volatility is sold in the *Long* position.
  Allowable values: For each underlying, see 19.3.29.
- [number] `Strikes2`: The volatility strike $K_{2,u,vol}$ of the variance swap for each underlying u in the `Underlyings2` basket, quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any positive number, as a percentage expressed in decimal form.
- [number] `Spreads2`: Additional spread to the realised variance, for each underlying in

the Underlyings2 basket.
Allowable values: Any real number.

- [number] Notionals2: For each underlying u in the Underlyings2 basket, the vega notional amount. If the swap was struck in terms of variance notionals $N_{2,u,var}$, the corresponding vega notionals are given by $N_{2,u,vol} = N_{2,u,var} \cdot 2 \cdot 100 \cdot K_{2,u,vol}$ (where $K_{2,u,vol}$ is in absolute terms).
Allowable values: Any real number.

- [number] Caps2: For each underlying u in the Underlyings2 basket, the cap on the realised variance as a factor of the strike. For example, if the value in Caps2 is 2.5, then the cap level will be $2.5^2 \times K_{2,u,\text{var}}$. For underlyings with no cap, set the value in Caps2 to zero.
Allowable values: Any non-negative number.

- [number] Floors2: For each underlying u in the Underlyings2 basket, the floor on the realised variance as a factor of the strike. For example, if the value in Floors2 is 0.1, then the floor level will be $0.1^2 \times K_{2,u,\text{var}}$. For underlyings with no floor, set the value in Floors2 to zero.
Allowable values: Any non-negative number.

- [number] UpperBarrierLevels: The agreed upper barrier price levels for each underlying in Underlyings1.
Allowable values: Any real number.

- [number] LowerBarrierLevels: The agreed lower barrier price levels for each underlying in Underlyings1.
Allowable values: Any real number.

- [bool] CountBothObservations: Whether the variance is accrued based on both the current and previous underlying prices (of each underlying in Underlyings1) falling within the corridor (*True*) or only on the previous underlying price (*False*).
Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.

- [bool] AccrualAdjustment: Whether the strike will be scaled relative to the number of days that the underlying traded within the corridor. See, for example, $\widetilde{K}_{var}$ in section Corridor Variance Swap of the Product Description for Exotic Variance and Volatility Swaps.

- [bool] DividendAdjustment: Whether or not the underlying price is adjusted for dividend payments. This feature is not yet supported. Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.

- [event] ValuationSchedule: The schedule defining the (daily) observation period for the variance accrual.
Allowable values: See Section 19.3.4.

- [event] SettlementDate: The date on which the swap payoff is settled.
Allowable values: See Date in Table 20.

- [currency] PayCcy: The payment currency. For FX, where the underlying is provided in the form FX-SOURCE-CCY1-CCY2 (see Table 28) this should be CCY2. If CCY1 or the

currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section “Payment Currency” in vre/docs/scripted_trade. Allowable values: See Table 22 for allowable currency codes.

KO Corridor Variance Dispersion Swap

This instrument is a variance dispersion swap with a double-barrier knock-out and a corridor feature.

With the corridor feature, variance is accrued only when the the price of the first underlying of each pair of underlyings (between the first and second basket) falls within the ‘corridor’. This means that the number of underlyings in each basket should be the same, and each pair is determined based on the order that they are provided in the trade XML, as in the example below.

The traditional trade representation is as follows, using EQ underlyings in this example:

```
<Trade id="EQ_VarianceDispersionSwap_Corridor_KO">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    .....
  </Envelope>
  <KOCorridorVarianceDispersionSwapData>
    <LongShort type="longShort">Long</LongShort>
    <Weights type="number">
        .....
    </Weights>
    <Underlyings1 type="index">
      <Value>EQ-RIC:HSBA.L</Value>
      <Value>EQ-RIC:MSFT.OQ</Value>
      <Value>EQ-RIC:AMZN.O</Value>
    </Underlyings1>
    <Strikes1 type="number">
        .....
    </Strikes1>
    <Spreads1 type="number">
        .....
    </Spreads1>
    <Notionals1 type="number">
        .....
    </Notionals1>
    <Caps1 type="number">
        .....
    </Caps1>
    <Floors1 type="number">
        .....
    </Floors1>
    <Underlyings2 type="index">
      <Value>EQ-RIC:.STOXX50E</Value>
      <Value>EQ-RIC:.SPX</Value>
      <Value>EQ-RIC:.SPX</Value>
    </Underlyings2>
    <Strikes2 type="number">
        .....
    </Strikes2>
    <Spreads2 type="number">
        .....
    </Spreads2>
    <Notionals2 type="number">
        .....
    </Notionals2>
    <Caps2 type="number">
        .....
    </Caps2>
    <Floors2 type="number">
        .....
    </Floors2>
```

```
    <CorridorUpperBarrierLevels type="number">
        .....
    </CorridorUpperBarrierLevels>
    <CorridorLowerBarrierLevels type="number">
        .....
    </CorridorLowerBarrierLevels>
    <KOUpperBarrierLevels type="number">
        .....
    <KOUpperBarrierLevels/>
    <KOLowerBarrierLevels type="number">
        .....
    </KOLowerBarrierLevels>
    <CountBothObservations type="bool">true</CountBothObservations>
    <AccrualAdjustment type="bool">true</AccrualAdjustment>
    <DividendAdjustment type="bool">false</DividendAdjustment>
    <KnockOutSchedule type="event">
      <ScheduleData>
        .....
      </ScheduleData>
    </KnockOutSchedule>
    <VarianceAccrualStartDate type="event">2020-12-02</VarianceAccrualStartDate>
    <SettlementSchedule type="event">
      <DerivedSchedule>
        <BaseSchedule>KnockOutSchedule</BaseSchedule>
          .....
      </DerivedSchedule>
    </SettlementSchedule>
    <PayCcy type="currency">USD</PayCcy>
  </KOCorridorVarianceDispersionSwapData>
</Trade>
```

The KOCorridorVarianceDispersionSwap script referenced in the trade above is shown in listing 191

The meanings and allowable values for the `KOCorridorVarianceDispersionSwapData` node below.

- [longShort] `LongShort`: Own party position in the swap. *Long* corresponds to selling volatility on the `Underlyings1` basket and buying volatility on `Underlyings2`.
  Allowable values: *Long, Short*.
- [number] `Weights`: List of weights applied to the final realised volatility of each underlying in the `Underlyings1` and `Underlyings2` baskets.
  Allowable values: Any positive number.
- [index] `Underlyings1`: The basket of underlyings whose volatility is bought in the *Long* position.
  Allowable values: For each underlying, see 19.3.29.
- [number] `Strikes1`: The volatility strike $K_{1,u,vol}$ of the variance swap for each underlying u in the `Underlyings1` basket, quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any positive number, as a percentage expressed in decimal form.
- [number] `Spreads1`: Additional spread to the realised variance, for each underlying in the `Underlyings1` basket.
  Allowable values: Any real number.
- [number] `Notionals1`: For each underlying u in the `Underlyings1` basket, the vega notional amount. If the swap was struck in terms of variance notionals $N_{1,u,var}$, the corresponding vega notionals are given by $N_{1,u,vol} = N_{1,u,var} \cdot 2 \cdot 100 \cdot K_{1,u,vol}$ (where $K_{1,u,vol}$ is in absolute terms).

Allowable values: Any real number.

- [number] `Caps1`: For each underlying u in the `Underlyings1` basket, the cap on the realised variance as a factor of the strike. For example, if the value in `Caps1` is 2.5, then the cap level will be $2.5^2 \times K_{1,u,\text{var}}$. For underlyings with no cap, set the value in `Caps1` to zero.
  Allowable values: Any non-negative number.
- [number] `Floors1`: For each underlying u in the `Underlyings1` basket, the floor on the realised variance as a factor of the strike. For example, if the value in `Floors1` is 0.1, then the floor level will be $0.1^2 \times K_{1,u,\text{var}}$. For underlyings with no floor, set the value in `Floors1` to zero.
  Allowable values: Any non-negative number.
- [index] `Underlyings2`: The basket of underlyings whose volatility is sold in the *Long* position.
  Allowable values: For each underlying, see 19.3.29.
- [number] `Strikes2`: The volatility strike $K_{2,u,vol}$ of the variance swap for each underlying u in the `Underlyings2` basket, quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any positive number, as a percentage expressed in decimal form.
- [number] `Spreads2`: Additional spread to the realised variance, for each underlying in the `Underlyings2` basket.
  Allowable values: Any real number.
- [number] `Notionals2`: For each underlying u in the `Underlyings2` basket, the vega notional amount. If the swap was struck in terms of variance notionals $N_{2,u,var}$, the corresponding vega notionals are given by $N_{2,u,vol} = N_{2,u,var} \cdot 2 \cdot 100 \cdot K_{2,u,vol}$ (where $K_{2,u,vol}$ is in absolute terms).
  Allowable values: Any real number.
- [number] `Caps2`: For each underlying u in the `Underlyings2` basket, the cap on the realised variance as a factor of the strike. For example, if the value in `Caps2` is 2.5, then the cap level will be $2.5^2 \times K_{2,u,\text{var}}$. For underlyings with no cap, set the value in `Caps2` to zero.
  Allowable values: Any non-negative number.
- [number] `Floors2`: For each underlying u in the `Underlyings2` basket, the floor on the realised variance as a factor of the strike. For example, if the value in `Floors2` is 0.1, then the floor level will be $0.1^2 \times K_{2,u,\text{var}}$. For underlyings with no floor, set the value in `Floors2` to zero.
  Allowable values: Any non-negative number.
- [number] `CorridorUpperBarrierLevels`: The agreed corridor upper barrier price levels for each underlying in `Underlyings1`.
  Allowable values: Any real number.
- [number] `CorridorLowerBarrierLevels`: The agreed corridor lower barrier price levels for each underlying in `Underlyings1`.
  Allowable values: Any real number.

- [number] `KOUpperBarrierLevels`: The agreed knock-out upper barrier price levels for each underlying in `Underlyings1`.
  Allowable values: Any real number.
- [number] `KOLowerBarrierLevels`: The agreed knock-out lower barrier price levels for each underlying in `Underlyings1`.
  Allowable values: Any real number.
- [bool] `CountBothObservations`: Whether the variance is accrued based on both the current and previous underlying prices (of each underlying in `Underlyings1`) falling within the corridor (*True*) or only on the previous underlying price (*False*).
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.
- [bool] `AccrualAdjustment`: Whether the strike will be scaled relative to the number of days that the underlying traded within the corridor. See, for example, $\widetilde{K}_{var}$ in section Corridor Variance Swap of the Product Description for Exotic Variance and Volatility Swaps.
- [bool] `DividendAdjustment`: Whether or not the underlying price is adjusted for dividend payments. This feature is not yet supported. Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. The full set of allowable values is given in Table 36.
- [event] `KnockOutSchedule`: The schedule defining the (daily) knock-out observation period.
  Allowable values: See Section 19.3.4.
- [event] `VarianceAccrualStartDate`: The date on which variance observation/accrual starts. The last variance observation date is the same as the last date in the `KnockOutSchedule`.
  Allowable values: See `Date` in Table 20.
- [event] `SettlementSchedule`: The settlement dates derived from the `KnockOutSchedule` using a settlement lag to reflect settlement after a knock-out event, or after the final variance observation date if no knock-out occurs..
  Allowable values: See section 19.3.4 Schedule Data and Dates, or DerivedSchedule (see (see the scripted trade documentation in vre/docs/scripted_trade).
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Gamma Swap

The traditional trade representation is as follows, using EQ underlyings in this example:

```
<Trade id="EQ_GammaSwap">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
   .....
  </Envelope>
    <GammaSwapData>
  <LongShort type="longShort">Long</LongShort>
```

```
  <Strike type="number">0.19</Strike>
  <Notional type="number">1380</Notional>
  <SettlementDate type="event">2021-09-25</SettlementDate>
  <Underlying type="index">EQ-RIC:.SPX</Underlying>
  <ValuationSchedule type="event">
    <ScheduleData>
      <Rules>
    <StartDate>2020-11-26</StartDate>
    <EndDate>2021-09-18</EndDate>
    <Tenor>1D</Tenor>
    <Convention>Following</Convention>
    <TermConvention>Following</TermConvention>
    <Calendar>USA</Calendar>
    <Rule>Forward</Rule>
       </Rules>
     </ScheduleData>
  </ValuationSchedule>
  <PayCcy type="currency">USD</PayCcy>
  </GammaSwapData>
</Trade>
```

The GammaSwap script referenced in the trade above is shown in listing 192

The meanings and allowable values for the `GammaSwapData` node below.

- [longShort] `LongShort`: Own party position in the swap. *Long* corresponds to paying out on the fixed/strike variance (volatility) and receiving on the floating/realised variance (volatility). In other words, a long position has positive value if the realised variance (volatility) exceeds the variance (volatility) strike.
  Allowable values: *Long, Short*.
- [number] `Strike`: The volatility strike K of the underlying quoted in absolute terms. If the swap was struck in terms of variance, the square roots of the variances should be used here.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.
- [number] `Notional`: The vega notional amount. If the swap was struck in terms of a variance notional N_{var}, the corresponding vega notional is given by $N_{vol} = N_{var} \cdot 2 \cdot 100 \cdot K$ (where K is in absolute terms).
  Allowable values: Any non-negative number.
- [event] `SettlementDate`: The date on which the swap payoff is settled.
  Allowable values: See `Date` in Table 20.
- [index] `Underlying`: Underlying index.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [event] `ValuationSchedule`: The base schedule which defines the start dates of the variance accrual schedule. This node is also used to derive the variance accrual schedule. Note that the end date (or last date) of this schedule must be the final period end date of the instrument.
  Allowable values: See Section 19.3.4.
- [currency] `PayCcy`: The payment currency. For FX, where the underlying is provided in the form `FX-SOURCE-CCY1-CCY2` (see Table 28) this should be `CCY2`. If `CCY1` or the currency of the underlying (for EQ and COMM underlyings), this will result in a quanto payoff. Notice section "Payment Currency" in vre/docs/scripted_trade.
  Allowable values: See Table 22 for allowable currency codes.

Basket Variance Swap

The `FxBasketVarianceSwap` and `EquityBasketVarianceSwap` `CommodityBasketVarianceSwap` trade types have trade data containers (respectively):

- `FxBasketVarianceSwapData`
- `EquityBasketVarianceSwapData`
- `CommodityBasketVarianceSwapData`

Listing 193 shows the structure of example trades for an Equity underlying.

Listing 193: EquityBasketVarianceSwap data

```
<Trade id="Equity_VarianceSwap_Basket">
  <TradeType>EquityBasketVarianceSwap</TradeType>
  <Envelope>
    .......
  </Envelope>
  <EquityBasketVarianceSwapData>
    <LongShort>Long</LongShort>
    <Strike>0.31613</Strike>
    <Notional>1340.0</Notional>
    <Underlyings>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.SPX</Name>
        <Weight>0.75</Weight>
      </Underlying>
      <Underlying>
        <Type>Equity</Type>
        <Name>RIC:.STOXX50E</Name>
        <Weight>0.25</Weight>
      </Underlying>
    </Underlyings>
    <SquaredPayoff>true</SquaredPayoff>
    <ValuationSchedule>
        <Rules>
        .......
        </Rules>
    </ValuationSchedule>
    <Cap>2.5</Cap>
    <SettlementDate>2022-06-29</SettlementDate>
    <Currency>USD</Currency>
  </EquityBasketVarianceSwapData>
</Trade>
```

The payout formula is the same as for a standard capped/floored variance swap, with the realised variance calculated as follows:

$$RealisedVariance = \frac{252}{D}\sum_{i=1}^{D}\left[\sum_{n=1}^{N} w_n \ln\left(\frac{X_{i,n}}{X_{i-1,n}}\right)\right]^2,$$

for D variance accrual/valuation dates over the life of the trade, where

- $X_{i,n}$ denotes the fixing for the n-th underlying at t_i.
- w_n is the basket weight for the n-th underlying.

The meanings and allowable values of elements in the `EquityBasketVarianceSwapData` node follow below.

- `LongShort`: Own party position.
  Allowable values: *Long*, *Short*

- `Strike`: The volatility strike K_{vol} of the variance swap quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.

- `Notional`: The vega notional amount. If the swap was struck in terms of a variance notional N_{var}, the corresponding vega notional is given by $N_{vol} = N_{var} \cdot 2 \cdot 100 \cdot K_{vol}$ (where K_{vol} is in absolute terms).
  Allowable values: Any non-negative number.

- `Underlyings`: The basket of underlyings.
  For `FxBasketVarianceSwap`, `Type` is set to *FX* and `Name` is a string of the form *SOURCE-CCY1-CCY2* where *CCY1* is the foreign currency, *CCY2* is the domestic currency, and *SOURCE* is the fixing source, see Table 28. (section Data Node / Index) for allowable values.

  For `EquityBasketVarianceSwap`, `Type` is set to *Equity* and `Name` and other fields are as outlined in 19.3.29.

  For `CommodityBasketVarianceSwap`, `Type` is set to *Commodity* and `Name` is an identifier of the commodity as outlined in 19.3.29 and in Table 32

  Allowable values: For each underlying, an `Underlying` node as outlined in 19.3.29. All underlyings must be from the same asset class.

- `SquaredPayoff` [Optional]: Flag indicating whether the trade is a variance swap (*True*) or a volatility swap (*False*).
  Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. Defaults to *False* if left blank or omitted. The full set of allowable values is given in Table 36.

- `ValuationSchedule`: The schedule defining the (daily) observation period for the variance accrual.
  Allowable values: See Section 19.3.4.

- `Cap` [Optional]: The cap on the realised variance, as a factor of the corresponding strike. For example, if `Cap` is 2.5, then the cap level will be $2.5^2 \times \hat{K}_{var}$ for the basket realised variance. For trades with no cap, set `Cap` to zero.
  Allowable values: Any non-negative number. Defaults to zero if left blank or omitted.

- `Floor` [Optional]: The floor on the realised variance, as a factor of the corresponding strike. For example, if `Floor` is 0.1, then the floor level will be $0.1^2 \times \hat{K}_{var}$ for the basket realised variance. For trades with no floor, set `Floor` to zero.
  Allowable values: Any non-negative number. Defaults to zero if left blank or omitted.

- `SettlementDate`: The date on which the swap payoff is settled.
  Allowable values: See `Date` in Table 20.

- `Currency`: The settlement currency.
  Allowable values: See Table 22 `Currency`.

Basket variance swaps can alternatively be represented as *scripted trades*, refer to the scripted trade documentation in vre/docs/scripted_trade for an introduction.

```
<Trade id="EQ_VarianceSwap_Basket">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    ......
  </Envelope>
  <BasketVarianceSwapData>
    <LongShort type="longShort">Long</LongShort>
    <Underlyings type="index">
      <Value>EQ-RIC:.STOXX50E</Value>
      <Value>EQ-RIC:.SPX</Value>
    </Underlyings>
    <Weights type="number">
      <Value>0.25</Value>
      <Value>0.75</Value>
    </Weights>
    <Strike type="number">0.31613</Strike>
    <Notional type="number">1339.0</Notional>
    <SquaredPayoff type="bool">true</SquaredPayoff>
    <ValuationSchedule type="event">
      <ScheduleData>
        <Rules>
         ......
        </Rules>
      </ScheduleData>
    </ValuationSchedule>
    <Cap type="number">2.5</Cap>
    <Floor type="number">0</Floor>
    <SettlementDate type="event">2022-06-29</SettlementDate>
    <PayCcy type="currency">USD</PayCcy>
  </BasketVarianceSwapData>
</Trade>
```

The PairwiseVarianceSwap script referenced in the trade above is shown in listing 194

The meanings and allowable values for the `BasketVarianceSwapData` node below.

- [longShort] `LongShort`: Own party position in the swap. *Long* corresponds to paying out on the fixed/strike variance (volatility) and receiving on the floating/realised variance (volatility). In other words, a long position has positive value if the realised variance (volatility) exceeds the variance (volatility) strike.
  Allowable values: *Long, Short*.
- [index] `Underlyings`: List of underlying indices.
  Allowable values: See vre/docs/scripted_trade's Index Section for allowable values.
- [number] `Strike`: The volatility strike K_{vol} of the variance swap quoted in absolute terms. If the swap was struck in terms of variance, the square root of that variance should be used here.
  Allowable values: Any non-negative number, as a percentage expressed in decimal form.
- [number] `Notional`: The vega notional amount. If the swap was struck in terms of a variance notional N_{var}, the corresponding vega notional is given by $N_{vol} = N_{var} \cdot 2 \cdot 100 \cdot K_{vol}$ (where K_{vol} is in absolute terms).
  Allowable values: Any non-negative number.
- `SquaredPayoff`: Flag indicating whether the trade is a variance swap (*True*) or a volatility swap (*False*).

Allowable values: Boolean node, allowing *Y*, *N*, *1*, *0*, *true*, *false*, etc. Defaults to *False* if left blank or omitted. The full set of allowable values is given in Table 36.

- [event] `ValuationSchedule`: The schedule defining the (daily) observation period for the variance accrual.
  Allowable values: See Section 19.3.4.
- [number] `Cap`: The cap on the realised variance, as a factor of the corresponding strike. For example, if `Cap` is 2.5, then the cap level will be $2.5^2 \times \hat{K}_{var}$ for the basket realised variance. For trades with no cap, set `Cap` to zero.
  Allowable values: Any non-negative number.
- [number] `Floor`: The floor on the realised variance, as a factor of the corresponding strike. For example, if `Floor` is 0.1, then the floor level will be $0.1^2 \times \hat{K}_{var}$ for the basket realised variance. For trades with no floor, set `Floor` to zero.
  Allowable values: Any non-negative number.
- [event] `SettlementDate`: The date on which the swap payoff is settled.
  Allowable values: See `Date` in Table 20.
- [currency] `PayCcy`: The settlement currency.
  Allowable values: See Table 22 for allowable currency codes.

19.2.84 Generic Scripted Products

The products in sections 19.2.70 to 19.2.80 are internally represented as *Scripted Trades*, but "wrapped" such that their input XML format still looks like "classic" VRE XML.

With 19.2.82 and 19.2.83 we have seen two examples of the generic Scripted Trade input format.

The Scripted Trade module allows flexible definition of new payoffs across five of the six asset classes covered in VRE, just by way of defining the payoff script. The payoff script can be embedded into the trade XML or can be placed into a separate script library.

Refer to the stand-alone Scripted Trade documentation in vre/docs/scripted_trade or section 20 for an introduction.

19.2.85 Flexi Swap

The `FlexiSwapData` node is the trade data container for trade type Flexi Swap. A Flexi Swap is a two-legged swap with optional and customisable pre-payments. Flexi Swaps are typically used for representing swaps linked to Asset Backed Securities with flexible amortisation. A Flexi Swap must have two legs, one fixed and one floating. The floating leg must have a pay frequency that is a multiple of the fixed leg frequency and corresponding floating and fixed leg periods must have the same notional. The legs typically have an amortising notional and are represented by `LegData` trade component sub-nodes, described in section 19.3.3. The `FlexiSwapData` node also contains a `OptionLongShort` node indicating the holder of the prepayment option and a node describing the optional prepayments, see below.

An example structure of a `FlexiSwapData` node is shown in Listing 195. In this case the optional pre-payments are given by a subnode `LowerNotionalBounds` meaning that the notional of the swap can be reduced to any value between the given lower bound and the original notional in each fixed leg period.

Listing 195: Flexi Swap data

```
<FlexiSwapData>
  <LowerNotionalBounds>
        <Notional>451389557.145667</Notional>
        <Notional>427876791.621303</Notional>
        <Notional>404435982.369285</Notional>
        <Notional>379353200.32956</Notional>
        ...
  </LowerNotionalBounds>
  <OptionLongShort>Short</OptionLongShort>
  <LegData>
    <LegType>Fixed</LegType>
    ...
  </LegData>
  <LegData>
    <LegType>Floating</LegType>
    ...
  </LegData>
</FlexiSwapData>
```

Alternatively the optional pre-payments can be described by a subnode `NotionalDecreases` which is more general than the description via LowerNotionalBounds (using the reduction type RedutionToLowerBound, see below for more details on this), see Listing 196 for an example.

Listing 196: Flexi Swap data

```
<FlexiSwapData>
  <Prepayment>
    <NoticePeriod>5D</NoticePeriod>
    <NoticeCalendar>TARGET</NoticeCalendar>
    <NoticeConvention>F</NoticePeriod>
    <PrepaymentOptions>
      <PrepaymentOption>
        <ExerciseDate>2015-02-01</ExerciseDate>
        <Type>ReductionUpToLowerBound</Type>
        <Value>404435982.369285</Value>
      </PrepaymentOption>
      <PrepaymentOption>
        <ExerciseDate>2016-02-01</ExerciseDate>
        <Type>ReductionByAbsoluteAmount</Type>
        <Value>100000.0</Value>
      </PrepaymentOption>
      <PrepaymentOption>
        <ExerciseDate>2017-02-01</ExerciseDate>
        <Type>ReductionUpToAbsoluteAmount</Type>
        <Value>50000.0</Value>
      </PrepaymentOption>
    <PrepaymentOptions>
  </Prepayment>
  <OptionLongShort>Short</OptionLongShort>
  <LegData>
    <LegType>Fixed</LegType>
    ...
  </LegData>
  <LegData>
    <LegType>Floating</LegType>
    ...
  </LegData>
</FlexiSwapData>
```

The meanings and allowable values of the elements in the `FlexiSwapData` node follow below.

- OptionLongShort: Specifies which party has the right to pre-pay the notional down to the lower notional bound. *Short* means that for pricing purposes pre-payments are assumed to be done in such a way to maximise the value of the Flexi Swap for the "other" counterparty, *Long* means that the Flexi Swap value is maximised from "our" point of view.

  Allowable values: *Long* or *Short*

- LegData: This is a trade component sub-node outlined in section 19.3.3. A Flexi Swap must have two `LegData` nodes and the LegType element must be set to *Floating* on one leg and *Fixed* on the other. The two legs must have the same `Currency`. The float leg pay frequency must be a multiple of the fixed leg frequency.

The optional prepayments are described by either a `LowerNotionalBounds` node or a `Prepyment` node.

In case the optional prepayments are described by a `LowerNotionalBounds` node, the

minimum level to which the notional can be amortised down to must be given as a notional schedule. The schedule can be specified as described in 19.3.3, i.e. using a sequence of `Notional` subnodes or using the `startDate` attribute to specify notional changes. The given schedule must be given for the fixed leg periods since the notional can be decreased for each whole fixed leg period and the corresponding floating leg periods (remember that the floating leg frequency must be a multiple of the fixed leg frequency). Each lower notional bound child element can take a positive real number that cannot exceed the notional amount of the corresponding coupon period on either leg and (from the second fixed coupon period on) the lower notional bound of the previous coupon period.

In case the optional prepayments are described by a `Prepayment` node, the the single exercise opportunities are described by a `PrepaymentOptions` subnode that contains one or several `PerpaymentOption` subnodes, each of which comprises the following elements:

- ExerciseDate: The date on which the notional can be decreased.
- Type: The type of the allowed notional reduction. The allowable types are
  - ReductionUpTpLowerBound: The notional can be reduced to any value between the current notional and the lower bound given in the Value node.
  - ReductionByAbsoluteAmount: The notional can be reduced by an absolute amount given in the Value node. If this value is greater than the current notional, the reduction amount is equal to the current notional.
  - ReductionUpToAbsoluteAmount: The notional can be reduced by any value between zero and a given absolute amount (given in the Value node).
- Value: The value that together with the type describes the amount by which the notional can be decreased.

In addition the `Prepayment` node contains the following optional subnodes describing the conventions for deriving the option notice date from the exercise date:

- NoticePeriod [Optional]: The notice period defining the date (relative to the exercise date) on which the exercise decision has to be taken. If not given the notice period defaults to 0D, i.e. the notice date is identical to the exercise date.
- NoticeCalendar [Optional]: The calendar used to compute the notice date from the exercise date. If not given defaults to the null calendar (no holidays, weekends are no holidays either).
- NoticeConvention [Optional]: The convention used to compute the notice date from the exercise date. Defaults to Unadjusted if not given.

19.2.86 Balance Guaranteed Swap (BGS)

BGS are priced in VRE using an auxiliary Flexi Swap as a proxy. The amortization schedule of the Flexi Swap is set up as the notional schedule of the BGS assuming a zero CPR (Conditional Prepayment Rate). The lower notional bound of the Flexi Swap is constructed assuming a MaxCPR (Maximum Conditional Prepayment Rate) which is dependent on the Reference Security. The MaxCPR is estimated on the basis of the current CPR, historical CPRs and / or expert judgement as to provide a (hypothetical) sufficiently realistic hedge for

the BGS. The option holder in the Flexi Swap is the payer of the structured leg (i.e. the leg replicating the payments of the reference security) in the BGS.

The `BalanceGuaranteedSwapData` node is the trade data container for trade type *BalanceGuaranteedSwap*. A BGS must have two legs, one fixed and one floating. Each leg typically has an amortising notional and is represented by a `LegData` trade component sub-node, described in section 19.3.3. The `BalanceGuaranteedSwapData` node also contains a `ReferenceSecurity` sub-node specifying the Asset Backed Security to which the notional schedule of the BGS is linked. An example structure of a `BalanceGuaranteedSwapData` node is shown in Listing 197.

Listing 197: Balance Guaranteed Swap data

```
<BalanceGuaranteedSwapData>
  <ReferenceSecurity>ISIN:XS0983610930</ReferenceSecurity>
  <Tranches>
    <Tranche>
      <Description>Class A</Description>
      <SecurityId>ISIN:XS0983610930</SecurityId>
      <Seniority>1</Seniority>
      <Notionals>
      ...
      </Notionals>
    </Tranche>
    <Tranche>
      <Description>Class B</Description>
      <SecurityId>ISIN:XS0983610931</SecurityId>
      <Seniority>2</Seniority>
      <Notionals>
      ...
      </Notionals>
    </Tranche>
    <ScheduleData>
    ...
    </ScheduleData>
  </Tranches>
  <LegData>
        <LegType>Fixed</LegType>
         ...
  </LegData>
  <LegData>
        <LegType>Floating</LegType>
         ...
  </LegData>
<BalanceGuaranteedSwapData>
```

The meanings and allowable values of the elements in the `BalanceGuaranteedSwapData` node follow below.

- ReferenceSecurity: The ISIN of the Asset Backed Security tranche to which the BGS is linked.

  Allowable values: The prefix `ISIN:` followed by an ISIN code for the Reference Security.

- Tranches: A description of the Asset Backed Security tranche notionals. Each Tranche is identified by a `SecurityId` and an optional `Description`. Each Tranche has a `Seniority` given as a positive integer value where lower values mean higher seniority, i.e. 1 is the most senior tranche (e.g. "class A") followed by 2 (e.g. "class B") etc. The notionals are given in a sub-node `Notionals` as described in section 19.3.3 w.r.t. a schedule given in `ScheduleData` which is shared across all tranches. There must be exactly one tranche with a security id matching the reference security.
- LegData: This is a trade component sub-node outlined in section 19.3.3. A BGS must have two `LegData` nodes and the LegType element must be set to *Floating* on one leg and *Fixed* on the other. The two legs must have the same `Currency`.

The notionals of the swap and the referenced tranche must be consistent. Furthermore, notionals for periods with a start date in the past must be given with their actual value, i.e. including actual prepayments that were made in the previous periods. Notionals for periods with a start dats in the future on the other hand must be given assuming a zero conditional prepayment rate. For the latter periods a prepayment model is used to generate suitable notional schedules when pricing the swap. The prepayment model assumes that tranches with higher seniority are amortised first, i.e. in the example here the class A tranche is amortised before the class B tranche.

19.3 Trade Components

Trade components are XML sub-nodes used within the trade data containers to define sets of trade data that more than one trade type can have in common, such as a leg or a schedule. A trade data container can include multiple trade components such as a swap with multiple legs, and a trade component can itself contain further trade components in a nested way.

An example of a `SwapData` trade data container, including two `LegData` trade components which in turn include further trade components such as `FixedLegData`, `ScheduleData` and `FloatingLegData` is shown in Listing 198.

Listing 198: Trade Components Example

```
<SwapData>
    <LegData>
        <Payer>true</Payer>
        <LegType>Fixed</LegType>
        <Currency>EUR</Currency>
        <PaymentConvention>Following</PaymentConvention>
        <DayCounter>30/360</DayCounter>
        <Notionals>
            <Notional>1000000</Notional>
        </Notionals>
        <ScheduleData>
        ...
        </ScheduleData>
        <FixedLegData>
            <Rates>
                <Rate>0.035</Rate>
            </Rates>
        </FixedLegData>
    </LegData>
    <LegData>
        ...
        <ScheduleData>
            ...
        </ScheduleData>
        <FloatingLegData>
            ...
        </FloatingLegData>
    </LegData>
</SwapData>
```

Descriptions of all trade components supported in VRE follow below.

19.3.1 Option Data

This trade component node is used within the `SwaptionData` and `FXOptionData` trade data containers. It contains the `ExerciseDates` sub-node which includes `ExerciseDate` child elements. An example structure of the `OptionData` trade component node is shown in Listing 199.

Listing 199: Option data

```
<OptionData>
  <LongShort>Long</LongShort>
  <OptionType>Call</OptionType>
  <Style>Bermudan</Style>
  <NoticePeriod>5D</NoticePeriod>
  <NoticeCalendar>TARGET</NoticeCalendar>
  <NoticeConvention>F</NoticePeriod>
  <Settlement>Cash</Settlement>
  <SettlementMethod>CollateralizedCashPrice</SettlementMethod>
  <MidCouponExercise>false</MidCouponExercise>
  <PayOffAtExpiry>true</PayOffAtExpiry>
  <ExerciseFees>
    <ExerciseFee type="Percentage">0.0020</ExerciseFee>
    <ExerciseFee type="Absolute" startDate="2020-04-20">25000</ExerciseFee>
  </ExerciseFees>
  <ExerciseFeeSettlementPeriod>2D</ExerciseFeeSettlementPeriod>
  <ExerciseFeeSettlementConvention>F</ExerciseFeeSettlementConvention>
  <ExerciseFeeSettlementCalendar>TARGET</ExerciseFeeSettlementCalendar>
  <ExerciseDates>
    <ExerciseDate>2019-04-20</ExerciseDate>
    <ExerciseDate>2020-04-20</ExerciseDate>
  </ExerciseDates>
  <!-- Alternative format for exercise dates using Schedule format -->
  <ExerciseSchedule>
    <Rules>
      <StartDate>2019-04-20</StartDate>
      <EndDate>2024-04-20</EndDate>
      <Tenor>3M</Tenor>
    </Rules>
  </ExerciseSchedule>
  <Premiums>
    <Premium>
      <Amount>100000</Amount>
      <Currency>EUR</Currency>
      <PayDate>2018-05-07</PayDate>
    </Premium>
  </Premiums>
  <AutomaticExercise>...</AutomaticExercise>
  <ExerciseData>
    <Date>...</Date>
    <Price>...</Price>
  </ExerciseData>
  <PaymentData>...</PaymentData>
</OptionData>
```

The meanings and allowable values of the elements in the `OptionData` node follow below.

- LongShort: Specifies whether the option position is *long* or *short*. Note that for Swaptions, Callable Swaps, and Index CDS Options setting `LongShort` to *short* makes the `Payer` indicator on the underlying Swap / Index CDS to be set from the perspective of the Counterparty.

  Allowable values: *Long, L* or *Short, S*

- OptionType: Specifies whether it is a call or a put option. Optional for trade types

Swaption and CallableSwap.

Allowable values: *Call* or *Put*

The meaning of Call and Put values depend on the trade type and asset class of the option, see Table 92.

Asset Class and Trade Type	**Call / Put Specifications**
Equity/ Commodity/Bond Option	*Call*: The right to buy the underlying equity/-commodity/bond at the strike price. *Put*: The right to sell the underlying equity/-commodity/bond at the strike price.
IR Swaption, CallableSwap, Commodity Swaption	*Call/Put* values are ignored, and the OptionType field is optional. Payer/Receiver swaption is determined by the `Payer` fields in the Leg Data nodes of the underlying swap.
FX Options (all variants, except Touch, Digital, Asian)	*Call*: Bought and Sold currencies/amounts stay as determined in the trade data node. *Put*: Bought and Sold currencies/amounts are switched compared to the trade data node. Note that barriers are not switched / unaffected.
Index CDS Option	*Call/Put* values are ignored, and the OptionType field is optional. The `Payer` field in the underlying Index CDS leg determines if the option is to buy or sell protection.
Asian FX Options	*Call*: The right to buy/receive the underlying currency at the strike price. *Put*: The right to sell/pay the underlying currency at the strike price.
Digital FX Options	*Call*: The digital payout will occur if the fx rate at the expiry date is above the given strike, *Put*: The digital payout will occur if the fx rate at the expiry date is below the given strike.
FX Single Touch Options	*Call/Put* values are ignored, and are instead inferred from the BarrierData type, and the OptionType field is optional.
FX Double Touch Options	*Call/Put* values are ignored, and and the OptionType field is optional.
Ascot	*Call* has payout: $\max(0, convertiblePrice - Strike)$ *Put* has payout: $\max(0, Strike - convertiblePrice)$

Table 92: Specification of Option Type Call / Put

- PayoffType [Optional, except for trade types detailed below]: Specifies a detailed payoff type for exotic options. Only applicable to specific trade types as indicated in parentheses:

  Allowable values:

  - *Accumulator, Decumulator* (applies to trade types EquityAccumulator, FxAccumulator, CommodityAccumulator only)
  - *TargetFull, TargetExact, TargetTruncated* (applies to trade types EquityTaRF, FxTaRF, CommodityTaRF only)
  - *BestOfAssetOrCash, WorstOfAssetOrCash, MaxRainbow, MinRainbow* (applies to trade types EquityRainbowOption, FxRainbowOption, CommodityRainbowOption only)
  - *Vanilla, Asian, AverageStrike, LookbackCall, LookbackPut* (applies to trade types EquityBasketOption, FxBasketOption, CommodityBasketOption only)
  - *Asian* (applies to trade types EquityAsianOption, FxAsianOption only)
  - *Vanilla, AssetOrNothing, CashOrNothing* (applies to trade type FxGenericBarrierOption, EquityGenericBarrierOption, CommodityGenericBarrierOption)

- Style: The exercise style of the option.

  Allowable values: *European* or *American* or *Bermudan*.

  Note that trade types IR Swaption and CallableSwap can have all three styles: *European*, *Bermudan*, or *American*.

  FX, Equity and Commodity vanilla options can have styles *European* or *American*, but not *Bermudan*.

  Exotic FX, Equity and Commodity options can generally only have style *European*, see each trade type for details.

  Commodity Swaption and Commodity Average Price Options must have style *European*.

  Index CDS Options must have style *European*.

  Ascots must have style *American*.

- PayoffType2 [Optional]: Subtype for payoff of exotic options. Only applicable to specific trade types as indicated in parantheses:

  Allowable values:

  - *Arithmetic, Geometric* (applies to trade types EquityAsianOption, FxAsianOption only, if not given it defaults to Arithmetic)

- NoticePeriod [Optional]: The notice period defining the date (relative to the exercise date) on which the exercise decision has to be taken. If not given the notice period defaults to 0D, i.e. the notice date is identical to the exercise date. Only supported for Swaptions and Callable Swaps currently.

- NoticeCalendar [Optional]: The calendar used to compute the notice date from the exercise date. If not given defaults to the null calendar (no holidays, weekends are no holidays either).
- NoticeConvention [Optional]: The convention used to compute the notice date from the exercise date. Defaults to Unadjusted if not given.
- Settlement: Delivery type. Note that Settlement is not required for Asian options.

  Allowable values: *Cash* or *Physical*
- SettlementMethod [Optional]: Specifies the method to calculate the settlement amount for Swaptions and CallableSwaps.

  Allowable values: *PhysicalOTC*, *PhysicalCleared*, *CollateralizedCashPrice*, *ParYieldCurve*.

  Defaults to *ParYieldCurve* if Settlement is *Cash* and defaults to *PhysicalOTC* if Settlement is *Physical*.

  PhysicalOTC = OTC traded swaptions with physical settlement
  PhysicalCleared = Cleared swaptions with physical settlement
  CollateralizedCashPrice = Cash settled swaptions with settlement price calculation using zero coupon curve discounting
  ParYieldCurve = Cash settled swaptions with settlement price calculation using par yield discounting [5] [6]
- MidCouponExercise [Optional]: Relevant for Swaptions and CallableSwaps. If *false*, the exercise-into underlying comprises all coupons with accrual start date greater or equal to notification date. I.e. one exercises into the next coupon, not the current one. If *true*, the exercise-into underlying comprises all coupons with accrual end date greater than the effective exercise date which is computed from the notification date by adding the notice period. The accrual paid for such coupons on exercise is calculated from the effective exercise date to the accrual end date (short coupon).

  Allowable values: *true*, *false*. If omitted, defaults to *false* for European and Bermudan swaptions/callableswaps and *true* for American swaptions/callableswaps.
- PayOffAtExpiry [Optional]: Relevant for options with early exercise, i.e. the exercise occurs before expiry; *true* indicates payoff at expiry, whereas *false* indicates payoff at exercise. Defaults to *true* if left blank or omitted.

  Allowable values: *true*, *false*.

  Note that for `IndexCreditDefaultSwapOption` PayOffAtExpiry must be set to *false* as only payoff at exercise is supported.
- Premiums [Optional]: Option premium amounts paid by the option buyer to the option seller.

  Allowable values: See section 19.3.2
- ExerciseDates: This node contains child elements of type `ExerciseDate` or an `ExerciseSchedule` node.

[5] https://www.isda.org/book/2006-isda-definitions/
[6] https://www.isda.org/a/TlAEE/Supplement-No-58-to-ISDA-2006-Definitions.pdf

Options of style *European* require a single exercise date expressed by one single `ExerciseDate` child element.

American style options must have exactly two `ExerciseDate` child elements representing the start and end of the American exercise period.

Bermudan style options must have two or more `ExerciseDate` child elements. One can alternatively use `ExerciseSchedule` to specify the option exercise dates for *Bermudan* style options.

- ExerciseSchedule [Optional]: This node can be provided instead of `ExerciseDates` and should be specified in the same format as a Schedule (see Section 19.3.4), e.g. for a list of Bermudan exercise dates.
- ExerciseFees [Optional]: This node contains child elements of type ExerciseFee. Similar to a list of notionals (see 19.3.3) the fees can be given either
  - as a list where each entry corresponds to an exercise date and the last entry is used for all remaining exercise dates if there are more exercise dates than exercise fee entries, or
  - using the `startDate` attribute to specify a change in a fee from a certain day on (w.r.t. the exercise date schedule)

  Fees can either be given as an absolute amount or relative to the current notional of the period immediately following the exercise date using the `type` attribute together with specifiers `Absolute` resp. `Percentage`. If not given, the type defaults to `Absolute`.

  If a fee is given as a positive number the option holder has to pay a corresponding amount if they exercise the option. If the fee is negative on the other hand, the option holder receives an amount on the option exercise.

  Only supported for Swaptions and Callable Swaps currently.
- ExerciseFeeSettlementPeriod [Optional]: The settlement lag for exercise fee payments. Defaults to *0D* if not given. This lag is relative to the exercise date (as opposed to the notice date).

  Allowable values: A number followed by *D, W, M, or Y*
- ExerciseFeeSettlementCalendar [Optional]: The calendar used to compute the exercise fee settlement date from the exercise date. If not given defaults to the *NullCalendar* (no holidays, weekends are no holidays either).

  Allowable values: See Table 24 Calendar.
- ExerciseFeeSettlementConvention [Optional]: The convention used to compute the exercise fee settlement date from the exercise date. Defaults to *Unadjusted* if not given.

  Allowable values: See Table 21 Roll Convention.
- AutomaticExercise [Optional]: Used if the option expiry date is on the current date or in the past, and the payment date is in the future - so that there still is an outstanding cashflow if the option was in the money on the expiry date. In this case, if AutomaticExercise is applied, the FX / Commodity / Equity fixing on the expiry date is

used to automatically determine the payoff and thus whether the option was exercised or not.

Currently, this field is only used for vanilla European cash settled FX, equity and commodity options. It is a boolean flag indicating if Automatic Exercise is applicable for the option trade. A value of *true* indicates that Automatic Exercise is applicable and a value of *false* indicates that it is not.

Allowable values: A boolean value given in Table 36. If not provided, the default value is *false*.

- ExerciseData [Optional]: Currently, this node is only used for vanilla European cash settled FX, equity and commodity options where *Automatic Exercise* is not applicable. It has the structure shown in Listing 199 i.e. a child `Date` and `Price` node. It is used to supply the price at which an option was exercised and the date of exercise. For a European option, the supplied date clearly has to match the single option `ExerciseDate`. It is needed where the cash settlement date is after the `ExerciseDate`. If this node is not supplied, and the `ExerciseDate` is in the past relative to the valuation date, the option is assumed to have expired unexercised.

  Allowable values: The `Date` node should be a valid date as outlined in Table 20 and the `Price` node should be a valid price as a real number.

- PaymentData [Optional]: This node is used to supply the date on which the option is cash settled if it is exercised. There are two methods in which this data may be supplied:

  1. The first method is an explicit list of dates as shown in Listing 200. The `Date` node should be a valid date as outlined in Table 20. Obviously, for European options, there should be exactly one date supplied.

  2. The second method is a set of rules that are used to generate the settlement date relative to either the exercise date of the option or the expiry date of the option. The structure of the `PaymentData` node in this case is given in Listing 201. The optional `RelativeTo` node must be either `Expiry` or `Exercise`. If it is `Expiry`, the expiry date is taken as the base date from which the rules are applied. If it is `Exercise`, the exercise date is taken as the base date from which the rules are applied. These two dates are the same in the case of a European option. If not provided, `Expiry` is assumed. The `Lag` node is a non-negative integer giving the number of days from the base date to the cash settlement date. The `Calendar` gives the business day calendar for the cash settlement date and should be a valid calendar code as outlined in Table 24. The `Convention` gives the roll convention for the cash settlement date and should be a valid roll convention as outlined in Table 21.

Listing 200: Dates based `PaymentData`

```
<PaymentData>
  <Dates>
    <Date>...</Date>
  </Dates>
</PaymentData>
```

Listing 201: Rules based `PaymentData`

```
<PaymentData>
  <Rules>
    <Lag>...</Lag>
    <Calendar>...</Calendar>
    <Convention>...</Convention>
    <RelativeTo>...</RelativeTo>
  </Rules>
</PaymentData>
```

19.3.2 Premiums

The `Premiums` node holds data of one or more premiums to be paid. It is used in different trade types, notably in caps / floors (see section 19.2.3) and more generally in the option data component (see section 19.3.1). Listing 202 shows an example for a Premiums data block representing two premiums.

Listing 202: Premiums Node

```
<Premiums>
  <Premium>
    <Amount>1000</Amount>
    <Currency>EUR</Currency>
    <PayDate>2021-01-27</PayDate>
  </Premium>
  <Premium>
    <Amount>5000</Amount>
    <Currency>USD</Currency>
    <PayDate>2023-01-27</PayDate>
  </Premium>
</Premiums>
```

The meanings and allowable values of the elements in the `Premium` node follow below.

- Amount: Option premium amounts paid by the option buyer to the option seller. A positive amount is considered to be paid by the option holder to the option seller and thus results in a negative contribution to the NPV of a long option.
  Allowable values: arbitrary number
- Currency: Currency of the premium to be paid
  Allowable values: See Table 22 `Currency`.
- PayDate: Date of the premium payment.
  Allowable values: See `Date` in Table 20.

We support a deprecated schema to represent a single premium as shown in listing 203 for backwards compatibility. The 3 nodes PremiumAmount, PremiumCurrency, PremiumPayDate can be used on the same level as the new Premiums node to represent a single premium payment. The deprecated and new schema may not be mixed.

Listing 203: Deprecated Single Premium Representation

```
<PremiumAmount>1000</PremiumAmount>
<PremiumCurrency>EUR</PremiumCurrency>
<PremiumPayDate>2021-01-27</PremiumPayDate>
```

19.3.3 Leg Data and Notionals

The `LegData` trade component node is used within the `CapFloorData`, `SwapData` , `SwaptionData` and `EquitySwapData` trade data containers. It contains a `ScheduleData` trade component sub-node, and a sub-node that depends on the value of the `LegType` element, e.g.: `FixedLegData` for `LegType` *Fixed* or `FloatingLegData` for `LegType` *Floating*. The `LegData` node also includes a `Notionals` sub-node with `Notional` child elements described below. An example structure of a `LegData` node of `LegType` *Floating* is shown in Listing 204.

Listing 204: Leg data

```
<LegData>
    <Payer>false</Payer>
    <LegType>Floating</LegType>
    <Currency>EUR</Currency>
    <PaymentConvention>Following</PaymentConvention>
    <DayCounter>30/360</DayCounter>
    <Notionals>
        <Notional>1000000</Notional>
    </Notionals>
    <ScheduleData>
        ...
    </ScheduleData>
    <FloatingLegData>
        ...
    </FloatingLegData>
</LegData>
```

The meanings and allowable values of the elements in the `LegData` node follow below.

- LegType: Determines which of the available sub-nodes must be used.

  Allowable values: *Fixed, Floating, Cashflow, CMS, CMB, DigitalCMS, DurationAdjustedCMS, CMSSpread, DigitalCMSSpread, Equity, CPI, YY, ZeroCouponFixed, FormulaBased, CommodityFloating, CommodityFixed, EquityMargin*

- Payer: The flows of the leg are paid to the counterparty if *true*, and received if *false*.

  Allowable values: *true, false*

- Currency: The currency of the leg.

  Allowable values: See Table 22 `Currency`. When `LegType` is *Equity*, Minor Currencies in Table 22 are also allowable.

- PaymentCalendar [Optional]: The payment calendar of the leg coupons. The `PaymentCalendar` is used in conjunction with the `PaymentConvention`,

PaymentLag and NotionalPaymentLag to determine the payments dates, unless the PaymentDates node is used which defines the payment dates explicitly.

Allowable values: See Table 24 Calendar. If left blank or omitted, defaults to the calendar in the ScheduleData node, unless LegType is *Floating* and Index is OIS, in which case this defaults to the index calendar.

The PaymentCalendar calendar field is currently only supported for LegType *Floating* (with an IBOR, BMA or OIS underlying index), *CMS*, *CMSSpread*, *DigitalCMSSpread*, *Equity*, *YY*, *CPI*, *Fixed*, *ZeroCouponFixed*, *DigitalCMS*. For unsupported legs it defaults to the schedule calendar, and if no calendar is set in the ScheduleData node (for dates-based schedules the calendar field is optional), the *NullCalendar* is used.

- PaymentConvention: The payment convention of the leg coupons.

  Allowable values: See Table 21.

- PaymentLag [optional]: The payment lag applies to the coupons on Fixed legs, Equity legs, and Floating legs with Ibor and OIS indices (but not to BMA/SIFMA indices), as well as CMS legs, CMSSpread legs, CPI legs and Zero Coupon Fixed legs. PaymentLag is also not supported for CapFloor Floating legs that have Ibor coupons with sub periods (HasSubPeriods = *true*), nor for CapFloor Floating legs with averaged ON coupons (IsAveraged = *true*).

  Allowable values: Any valid period, i.e. a non-negative whole number, optionally followed by *D* (days), *W* (weeks), *M* (months), *Y* (years). Defaults to *0D* if left blank or omitted. If a whole number is given and no letter, it is assumed that it is a number of *D* (days).

- NotionalPaymentLag [optional]: The notional payment lag (in days) applied to any notional exchanges.

  Allowable values: Any non-negative integer. Defaults to zero if left blank or omitted.

- DayCounter: The day count convention of the leg coupons. Note that DayCounter is mandatory for all leg types except *Equity*.

  Allowable values: See DayCount Convention in Table 25. For *Equity* legs, if left blank or omitted, it defaults to *ACT/365*.

- Notionals: This node contains child elements of type Notional. If the notional is fixed over the life of the leg only one notional value should be entered. If the notional is amortising or accreting, this is represented by entering multiple notional values, each represented by a Notional child element. The first notional value corresponds to the first coupon, the second notional value corresponds to the second coupon, etc. If the number of coupons exceeds the number of notional values, the notional will be kept flat at the value of last entered notional for the remaining coupons. The number of entered notional values cannot exceed the number of coupons.

  Allowable values: Each child element can take any positive real number.

  An example of a Notionals element for an amortising leg with four coupons is shown in Listing 205.

Listing 205: Notional list

```
<Notionals>
    <Notional>65000000</Notional>
    <Notional>65000000</Notional>
    <Notional>55000000</Notional>
    <Notional>45000000</Notional>
</Notionals>
```

Another allowable specification of the notional schedule is shown in Listing 206.

Listing 206: Notional list with dates

```
<Notionals>
    <Notional>65000000</Notional>
    <Notional startDate='2016-01-02'>65000000</Notional>
    <Notional startDate='2017-01-02'>55000000</Notional>
    <Notional startDate='2021-01-02'>45000000</Notional>
</Notionals>
```

The first notional must not have a start date, it will be associated with the schedule's start, The subsequent notionals must either all or none have a start date specified from which date onwards the new notional is applied. This allows specifying notionals only for dates where the notional changes.

An initial exchange, a final exchange and an amortising exchange can be specified using an `Exchanges` child element with `NotionalInitialExchange`, `NotionalFinalExchange` and `NotionalAmortizingExchange` as subelements, see Listing 207. The `Exchanges` element is typically used in cross-currency swaps and inflation swaps, but can also be used in other trade and leg types. Note that for cross-currency swaps, the `NotionalInitialExchange` must be set to the same value on both legs. The `NotionalFinalExchange` must also be set to the same value on both legs, i.e. *true* on both, or *false* on both.

Allowable values for `NotionalInitialExchange`, `NotionalFinalExchange` and `NotionalAmortizingExchange`: *true, false*. Defaults to *false* if omitted, or if the entire `Exchanges` block is omitted.

Listing 207: Notional list with exchange

```
<Notionals>
    <Notional>65000000</Notional>
    <Exchanges>
      <NotionalInitialExchange>true</NotionalInitialExchange>
      <NotionalFinalExchange>true</NotionalFinalExchange>
      <NotionalAmortizingExchange>true</NotionalAmortizingExchange>
    </Exchanges>
</Notionals>
```

FX Resets, used for Rebalancing Cross-currency swaps, can be specified using an `FXReset` child element with the following subelements: See Listing 208 for an example.

- ForeignCurrency: The foreign currency the notional of the leg resets to.

  Allowable values: See Table 22 `Currency`.

- ForeignAmount: The notional amount in the foreign currency that the notional of the leg resets to.

  Allowable values: Any positive real number.

- FXIndex: A reference to an FX Index source for the FX reset fixing.

  Allowable values: A string on the form FX-SOURCE-CCY1-CCY2.

Listing 208: Notional list with fx reset

```
<Currency>USD</Currency>
<Notionals>
    <Notional>65000000</Notional> <!-- in USD -->
    <FXReset>
      <ForeignCurrency> EUR </ForeignCurrency>
      <ForeignAmount> 60000000 </ForeignAmount>
      <FXIndex> FX-ECB-USD-EUR </FXIndex>
    </FXReset>
</Notionals>
```

- StrictNotionalDates [Optional]: If given and set to true, notional changes specified by startDate will be interpreted as taking place on the exact given date, even if that date falls into a calculation (accrual) period. Otherwise the notional change is applied for the next calculation period. Supported only for fixed and floating legs with IBOR / RFR term rate coupons.
- ScheduleData: This is a trade component sub-node outlined in section 19.3.4 Schedule Data and Dates.
- `PaymentSchedule` [Optional]: This node allows for the specification of an explicit payment schedule, see 19.3.4. Supported in commodity trades, fixed legs and floating legs with underlying OIS and IBOR indices.
- `PaymentDates` [Deprecated]: This node allows for the specification of a list of explicit payment dates. The usage is deprecated, use PaymentSchedule instead.
- FixedLegData: This trade component sub-node is required if `LegType` is set to *Fixed* It is outlined in section 19.3.5.
- FloatingLegData: This trade component sub-node is required if `LegType` is set to *Floating* It is outlined in section 19.3.6 Floating Leg Data and Spreads.
- CashflowLegData: This trade component sub-node is required if `LegType` is set to *Cashflow*. It is outlined in section 19.3.9.

- CMSLegData: This trade component sub-node is required if `LegType` is set to *CMS* (Constant Maturity Swap). It is outlined in section 19.3.10.
- CMBLegData: This trade component sub-node is required if `LegType` is set to *CMB* (Constant Maturity Bond). It is outlined in section 19.3.11.
- DigitalCMSLegData: This trade component sub-node is required if `LegType` is set to *DigitalCMS*. It is outlined in section 19.3.12.
- DurationAdjustedCMSLegData: This trade component sub-node is required if `LegType` is set to *DurationAdjustedCMS*. It is outlined in section 19.3.13.
- CMSSpreadLegData: This trade component sub-node is required if `LegType` is set to *CMSSpread*. It is outlined in section 19.3.14.
- DigitalCMSSpreadLegData: This trade component sub-node is required if `LegType` is set to *DigitalCMSSpread*. It is outlined in section 19.3.15.
- EquityLegData: This trade component sub-node is required if `LegType` is set to *Equity*. It is outlined in section 19.3.16.
- CPILegData: This trade component sub-node is required if `LegType` is set to *CPI*. It is outlined in section 19.3.17.
- YYLegData: This trade component sub-node is required if `LegType` is set to *YY*. It is outlined in section 19.3.18.
- ZeroCouponFixedLegData: This trade component sub-node is required if `LegType` is set to *ZeroCouponFixed*. It is outlined in section 19.3.19.
- FormulaBasedLegData: This trade component sub-node is required if `LegType` is set to *FormulaBased*. It is outlined in section 19.3.35.
- CommodityFloatingLegData: This trade component sub-node is required if `LegType` is set to *CommodityFloating* It is outlined in section 19.3.24.
- CommodityFixedLegData: This trade component sub-node is required if `LegType` is set to *CommodityFixed* It is outlined in section 19.3.21.
- EquityMarginLegData: This trade component sub-node is required if `LegType` is set to *EquityMargin* It is outlined in section 19.3.26.

19.3.4 Schedule Data (Rules, Dates and Derived)

The `ScheduleData` trade component node is used within the `LegData` trade component. The Schedule can be rules based (at least one `Rules` sub-node exists), dates based (at least one `Dates` sub-node exists, where the schedule is determined directly by `Date` child elements), or derived from another schedule in the same leg (at least one `Derived` sub-node exists). In rules based schedules, the schedule dates are generated from a set of rules based on the entries of the sub-node Rules, having the elements `StartDate`, `EndDate`, `Tenor`, `Calendar`, `Convention`, `TermConvention`, and `Rule`. Example structures of `ScheduleData` nodes based on rules, dates and derived from a base schedule are shown in Listing 209, Listing 210, and Listing 211 respectively.

Listing 209: Schedule data, rules based

```
<ScheduleData>
  <Rules>
    <StartDate>2013-02-01</StartDate>
    <EndDate>2030-02-01</EndDate>
    <Tenor>1Y</Tenor>
    <Calendar>UK</Calendar>
    <Convention>MF</Convention>
    <TermConvention>MF</TermConvention>
    <Rule>Forward</Rule>
  </Rules>
</ScheduleData>
```

Listing 210: Schedule data, date based

```
<ScheduleData>
  <Dates>
    <Calendar>NYB</Calendar>
    <Convention>Following</Convention>
    <Tenor>3M</Tenor>
    <EndOfMonth>false</EndOfMonth>
    <EndOfMOnthConvention>Following</EndOfMOnthConvention>
    <IncludeDuplicateDates>false</IncludeDuplicateDates>
    <Dates>
      <Date>2012-01-06</Date>
      <Date>2012-04-10</Date>
      <Date>2012-07-06</Date>
      <Date>2012-10-08</Date>
      <Date>2013-01-07</Date>
      <Date>2013-04-08</Date>
    </Dates>
  </Dates>
</ScheduleData>
```

Listing 211: Schedule data, derived

```
<ScheduleData>
  <Derived>
    <BaseSchedule>ScheduleData</BaseSchedule>
    <Shift>3M</Shift>
    <Calendar>GBP</Calendar>
    <Convention>Following</Convention>
  </Derived>
</ScheduleData>
```

The ScheduleData section can contain any number and combination of <Dates>, <Rules> and <Derived> sections. The resulting schedule will then be an ordered concatenation of individual schedules.

The meanings and allowable values of the elements in a <Rules> based section of the ScheduleData node follow below.

- `Rules`: a sub-node that determines whether the schedule is set by specifying rules that generate the schedule. If existing, the following entries are required: `StartDate`, `EndDate`, `Tenor`, `Calendar`, `Convention`, and `Rule`. `EndDateConvention` is optional. If not existing, a `Dates` or `Derived` sub-node is required.
- `StartDate`: The schedule start date.

  Allowable values: See `Date` in Table 20.
- `EndDate`: The schedule end date. This can be omitted to indicate a perpetual schedule. Notice that perpetual schedule are only supported by specific trade types (e.g. Bond).

  Allowable values: See `Date` in Table 20.
- `AdjustEndDateToPreviousMonthEnd` [Optional]: Only relevant for commodity legs. Allows for the `EndDate` to be on a date other than the end of the month. If set to *true* the given `EndDate` is restated to the end date of the previous month.

  Allowable values: *true* or *false*. Defaults to false if left blank or omitted.
- `Tenor`: The tenor used to generate schedule dates.

  Allowable values: A string where the last character must be *D* or *W* or *M* or *Y*. The characters before that must be a positive integer.
  D = Day, *W* = Week, *M* = Month, *Y* = Year

  Note that *0D* and *1T* are equivalent valid values, and both cause there to be no intermediate dates between `StartDate` and `EndDate`.
- `Calendar`: The calendar used to generate schedule dates. Also used to determine payment dates (except for compounding OIS index legs, which use the index calendar to determine payment dates).

  Allowable values: See Table 24 Calendar.
- `Convention`: Determines the adjustment of the schedule dates with regards to the selected calendar, i.e. the roll convention.

  Allowable values: See Table 21 Roll Convention.
- `TermConvention` [Optional]: Determines the adjustment of the final schedule date with regards to the selected calendar. If left blank or omitted, defaults to the value of `Convention`.

  Allowable values: See Table 21 Roll Convention.
- `Rule` [Optional]: Rule for the generation of the schedule using given start and end dates, tenor, calendar and roll conventions.

  Allowable values and descriptions: See Table 23 Rule. Defaults to *Forward* if omitted. Cannot be left blank.
- `EndOfMonth` [Optional]: Specifies whether the date generation rule is different for end of month, so that the last date of each month is generated, regardless of number of days in the month.

  If `EndOfMonth` is *true*, and `EndOfMonthConvention` is omitted:

- the date is set to the last calendar day in a month if the roll convention is *Unadjusted*, and
- the date is set to the last business day in a month if the roll convention is anything other than *Unadjusted*

Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36. Defaults to *false* if left blank or omitted. Must be set to *false* or omitted if the date generation Rule is set to *CDS* or *CDS2015*.

- `EndOfMonthConvention` [Optional]: Determines the adjustment of the end-of-month schedule dates with regards to the selected calendar. This field is only used when `EndOfMonth` is *true*. If left blank or omitted, then the default *Preceding / MF* convention is applied (i.e. end-of-month dates will never be adjusted over to the beginning of the next month)

  Allowable values: See Table 21 Roll Convention.

  Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36. Defaults to *false* if left blank or omitted. Must be set to *false* or omitted if the date generation Rule is set to *CDS* or *CDS2015*.

- `FirstDate` [Optional]: Date for initial stub period, determining the end date of the first period. If omitted the first period will follow the date generation rule. Note that for date generation rules *CDS* and *CDS2015*, the FirstDate has no impact and the schedule is built from IMM dates.

  Allowable values: See `Date` in Table 20. The FirstDate cannot be before the StartDate of the Schedule, and cannot be after the EndDate of the Schedule.

- `LastDate` [Optional]: Date for final stub period, determining the start date of the last period. If omitted the last period will follow the date generation rule. For date generation rules *CDS* and *CDS2015*, the LastDate has no impact and the schedule is built from IMM dates.

  Allowable values: See `Date` in Table 20. The LastDate cannot be after the EndDate of the Schedule, and cannot be before the StartDate of the Schedule.

- `RemoveFirstDate` [Optional]: If true the first date will be removed from the schedule. Useful to define a payment schedule using the rules for a calculation schedule.

  Allowable values: true, false

- `RemoveLastDate` [Optional]: If true the last date will be removed from the schedule. Useful to define a fixing or reset schedule using the rules for a calculation schedule.

  Allowable values: true, false

The meanings and allowable values of the elements in a <`Dates`> based section of the `ScheduleData` node follow below.

- `Dates`: a sub-node that determines that the schedule is set by specifying schedule dates explicitly.
- `Calendar` [Optional]: Calendar used to determine the accrual schedule dates. Also used to determine payment dates (except for compounding OIS index legs, which use

the index calendar), and also to compute day count fractions for irregular periods when day count convention is ActActISMA and the schedule is dates based.

Allowable values: See Table 24 Calendar. Defaults to *NullCalendar* if omitted, i.e. no holidays at all, not even on weekends.

- `Convention` [Optional]: Roll Convention to determine the accrual schedule dates, and also used to compute day count fractions for irregular periods when day count convention is ActActISMA and the schedule is dates based.

  Allowable values: See Table 21 Roll Convention. Defaults to *Unadjusted* if omitted.

- `Tenor` [Optional]: Tenor used to compute day count fractions for irregular periods when day count convention is ActActISMA and the schedule is dates based.

  Allowable values: A string where the last character must be *D* or *W* or *M* or *Y*. The characters before that must be a positive integer.
  D = Day, *W* = Week, *M* = Month, *Y* = Year

  Defaults to *null* if omitted.

- `EndOfMonth` [Optional]: Specifies whether the end of month convention is applied when calculating reference periods for irregular periods when the day count convention is ActActICMA and the schedule is dates based.

  Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36. Defaults to *false* if left blank or omitted.

- `EndOfMonthConvention` [Optional]: Whenever the `EndOfMonth` logic is applied, this is used as the roll convention along with the `Calendar`for any date adjustments.

  Allowable values: See Table 21 Roll Convention. Defaults to *Preceding* if omitted.

- `IncludeDuplicateDates` [Optional]: If set to *false* the resulting schedule will have unqiue set of dates and all duplicates will be removed. Default to *false*.

- `Dates`: This is a sub-sub-node and contains child elements of type `Date`. In this case the schedule dates are determined directly by the `Date` child elements. At least two `Date` child elements must be provided. Dates must be provided in chronological order. Note that if no calendar and roll convention is given, the specified dates must be given as adjusted dates.

  Allowable values: Each `Date` child element can take the allowable values listed in `Date` in Table 20.

The meanings and allowable values of the elements in a <`Derived`> section of the `ScheduleData` node follow below.

- `BaseSchedule`: The schedule from which the derived schedule will be deduced.

  Allowable values: Must be the node name of another schedule in a given leg data node.

- `Shift` [Optional]: The tenor/period offset to be applied to each date in the base schedule in order to obtain the derived schedule.

  Allowable values: A string where the last character must be *D* or *W* or *M* or *Y*. The characters before that must be a positive integer.

D = Day, W = Week, M = Month, Y = Year. If left blank or omitted, defaults to *0D*.

- `Calendar` [Optional]: The calendar adjustment to be applied to each date in the base schedule in order to obtain the derived schedule.

  Allowable values: See Table 24 Calendar. Defaults to *NullCalendar* if left blank or omitted, i.e. no holidays at all, not even on weekends.

- `Convention` [Optional]: The roll convention to be applied to each date in the base schedule in order to obtain the derived schedule.

  Allowable values: See Table 21 Roll Convention. Defaults to *Unadjusted* if left blank or omitted.

- `RemoveFirstDate` [Optional]: If true the first date will be removed from the schedule. Useful to define a payment schedule based on a calculation schedule.

  Allowable values: true, false

- `RemoveLastDate` [Optional]: If true the last date will be removed from the schedule. Useful to define a fixing or reset schedule based on a calculation schedule.

  Allowable values: true, false

19.3.5 Fixed Leg Data and Rates

The `FixedLegData` trade component node is used within the `LegData` trade component when the `LegType` element is set to *Fixed*. The `FixedLegData` node only includes the `Rates` sub-node which contains the rates of the fixed leg as child elements of type `Rate`.

An example of a `FixedLegData` node for a fixed leg with constant notional is shown in Listing 212.

Listing 212: Fixed leg data

```
<FixedLegData>
      <Rates>
          <Rate>0.05</Rate>
      </Rates>
</FixedLegData>
```

The meanings and allowable values of the elements in the `FixedLegData` node follow below.

- Rates: This node contains child elements of type `Rate`. If the rate is constant over the life of the fixed leg, only one rate value should be entered. If two or more coupons have different rates, multiple rate values are required, each represented by a `Rate` child element. The first rate value corresponds to the first coupon, the second rate value corresponds to the second coupon, etc. If the number of coupons exceeds the number of rate values, the rate will be kept flat at the value of last entered rate for the remaining coupons. The number of entered rate values cannot exceed the number of coupons.

  Allowable values: Each child element can take any real number. The rate is expressed in decimal form, e.g. 0.05 is a rate of 5%.

As in the case of notionals, the rate schedule can be specified with dates as shown in Listing 213.

Listing 213: Fixed leg data with 'dated' rates

```
<FixedLegData>
      <Rates>
          <Rate>0.05</Rate>
          <Rate startDate='2016-02-04'>0.05</Rate>
          <Rate startDate='2019-02-05'>0.05</Rate>
      </Rates>
</FixedLegData>
```

19.3.6 Floating Leg Data, Spreads, Gearings, Caps and Floors

The `FloatingLegData` trade component node is used within the `LegData` trade component when the `LegType` element is set to *Floating*. It is also used directly within the `CapFloor` trade data container. The `FloatingLegData` node includes elements specific to a floating leg.

An example of a `FloatingLegData` node is shown in Listing 214.

Listing 214: Floating leg data

```
<FloatingLegData>
    <Index>USD-LIBOR-3M</Index>
    <IsInArrears>false</IsInArrears>
    <IsAveraged>false</IsAveraged>
    <HasSubPeriods>false</HasSubPeriods>
    <IncludeSpread>false</IncludeSpread>
    <FixingDays>2</FixingDays>
    <Spreads>
        <Spread>0.005</Spread>
    </Spreads>
    <Gearings>
        <Gearing>2.0</Gearing>
    </Gearings>
    <Caps>
        <Cap>0.05</Cap>
    </Caps>
    <Floors>
        <Floor>0.01</Floor>
    </Floors>
    <NakedOption>false</NakedOption>
    <LocalCapFloor>false</LocalCapFloor>
    <HistoricalFixings>
                    <Fixing fixingDate="2016-02-01">0.2</Fixing>
                </HistoricalFixings>
</FloatingLegData>
```

The meanings and allowable values of the elements in the `FloatingLegData` node follow below.

- Index: The combination of currency, index and term that identifies the relevant fixings and yield curve of the floating leg.

Allowable values: An alphanumeric string of the form CCY-INDEX-TENOR. CCY, INDEX and TENOR must be separated by dashes (-). CCY and INDEX must be among the supported currency and index combinations. TENOR must be an integer followed by D, W, M or Y. See Table 26. TENOR is not required for Overnight indices, but can be set to *1D*.

- IsAveraged [Optional]: For cases where there are multiple index fixings over a period *true* indicates that the average of the fixings is used to calculate the coupon. *false* indicates that the coupon is calculated by compounding the fixings. IsAveraged only applies to Overnight indices and Sub Periods Coupons.

  Allowable values: *true, false*. Defaults to *false* if left blank or omitted.

- HasSubPeriods [Optional]: For cases where several Ibor fixings result in a single payment for a period, e.g. if the Ibor tenor is 3M and the schedule tenor is 6M, two fixings are used to compute the amount of the semiannual coupon payments. *true* indicates that an average (IsAveraged = true) or a compounded (IsAveraged=false) value of the fixings is used to determine the payment rate. *false* indicates that the initial index period fixing determines the payment rate for the full tenor, i.e. no further fixings, no averaging and no compounding. IsAveraged is ignored for Ibor legs when HasSubPeriods is set to *false*. HasSubPeriods does not apply to Overnight indices.

  Allowable values: *true, false*. Defaults to *false* if left blank or omitted.

- IncludeSpread [Optional]: Only applies to Sub Periods and (compounded) OIS Coupons. If *true* the spread is included in the compounding, otherwise it is excluded.

  Allowable values: *true, false*. Defaults to *false* if left blank or omitted.

  A Zero Coupon Floating leg with compounding that includes spread can be set up using a rules-based schedule as shown in Listing 215. Note that the `Tenor` in the rules-based schedule is not used when `Rule` is set to *Zero*.

Listing 215: Zero Coupon Floating Leg - Rules-based

```
<LegData>
    <LegType>Floating</LegType>
    <Payer>false</Payer>
    <Currency>USD</Currency>
    <Notionals>
        <Notional>200000.0000</Notional>
    </Notionals>
    <DayCounter>A360</DayCounter>
    <PaymentConvention>MF</PaymentConvention>
    <ScheduleData>
        <Rules>
            <StartDate>2020-01-14</StartDate>
            <EndDate>2020-07-14</EndDate>
            <Tenor>3M</Tenor>
            <Calendar>USD</Calendar>
            <Convention>MF</Convention>
            <TermConvention>MF</TermConvention>
            <Rule>Zero</Rule>
        </Rules>
    </ScheduleData>
    <FloatingLegData>
        <Index>USD-LIBOR-3M</Index>
        <IsAveraged>false</IsAveraged>
        <HasSubPeriods>true</HasSubPeriods>
        <IncludeSpread>true</IncludeSpread>
        <Spreads>
            <Spread>0.006500</Spread>
        </Spreads>
        <IsInArrears>false</IsInArrears>
        <FixingDays>2</FixingDays>
    </FloatingLegData>
</LegData>
```

A Zero Coupon Floating leg with compounding that includes spread can also be set up using a dates-based schedule with two dates (start and end) as shown in Listing 216.

Listing 216: Zero Coupon Floating Leg - Dates-based

```
<LegData>
    <LegType>Floating</LegType>
    <Payer>false</Payer>
    <Currency>USD</Currency>
    <Notionals>
        <Notional>200000.0000</Notional>
    </Notionals>
    <DayCounter>A360</DayCounter>
    <PaymentConvention>MF</PaymentConvention>
    <ScheduleData>
        <Dates>
            <Calendar>USD</Calendar>
            <Convention>MF</Convention>
            <Dates>
                <Date>2020-01-14</Date>
                <Date>2020-07-14</Date>
            </Dates>
        </Dates>
    </ScheduleData>
    <FloatingLegData>
        <Index>USD-LIBOR-3M</Index>
        <IsAveraged>false</IsAveraged>
        <HasSubPeriods>true</HasSubPeriods>
        <IncludeSpread>true</IncludeSpread>
        <Spreads>
            <Spread>0.006500</Spread>
        </Spreads>
        <IsInArrears>false</IsInArrears>
        <FixingDays>2</FixingDays>
    </FloatingLegData>
</LegData>
```

- IsInArrears [Optional]: *true* indicates that fixing is in arrears, *false* indicates that fixing is in advance.
  - For Ibor coupons, "in arrears" means that the fixing gap is calculated in relation to the current period end date, while "in advance" means that the fixing gap is calculated in relation to the period start date.
  - For OIS coupons, "in arrears" means that the compounding (or averaging) of ON rates is done over the current period (with period as defined in ScheduleData), while "in advance" means that the compounding (averaging) is done over the previous period. For the first period, a virtual previous period will be constructed based on the schedule construction rules. In the context of RFRs there are two common "in advance" variants:
    - "Last Recent" which means the length of the period used for compounding / averaging is independent of the original period. This former period is specified in the LastRecentPeriod field.
    - "Last Reset" which means the original period will be used for compounding / averaging. This variant is indicated by omitting the LastRecentPeriod field.

    Notice that the use of the LastRecentPeriod field is not restricted to "in advance"

OIS coupons, i.e. it can also be used in combination with “in arrears”.

Allowable values: *true, false*. Defaults to *false* for Ibor and to *true* for OIS coupons, if left blank or omitted.

- LastRecentPeriod [Optional]: Only applies to OIS coupons. If given, the compounding / averaging of ON rates will not be done over the usual reference period derived from the accrual period and the Lookback, FixingDays and IsInArrears parameters, but instead over a period determined by the end date of this usual period and the LastRecentPeriod parameter as [EndDate - LastRecentPeriod, EndDate]. The calendar used to compute EndDate - LastRecentPeriod is the schedule calendar unless a specific LastRecentPeriodCalendar is specified. To represent SOFR 30D, 90D, 180D average indices, the LastRecentPeriodCalendar should be set to NullCalendar, since these averages refer to rolling averages over 30, 90, 180 calendar days.
  Allowable values: any valid period, e.g. 30D, 90D, 180D, 1M, 2M, 6M
- LastRecentPeriodCalendar [Optional]: The calendar used to compute the LastRecentPeriod, see this field for more details. If not given, defaults to the schedule calendar.
  Allowable values: See Table 24 Calendar.
- FixingDays [Optional]: The fixing gap. For Ibor coupons this is the number of business days before the accrual period’s *reference* date to observe the index fixing. Here, the accrual period reference date is the accrual start date for an in advanced fixed coupon and the accrual end date for in arrears fixed coupon.
  For overnight coupons this is the number of business days by which the value dates are shifted into the past to get the fixing observation dates. In the context of RFRs the FixingDays parameter is sometimes also called “obervation lag”.

  The calendar used for the fixing gap, is the calendar associated with the floating index, as defined in the conventions for the index.

  Allowable values: A non-negative whole number. Defaults to the index’s fixing days if blank or omitted. See defaults per index in Table 27.
- Lookback [Optional]: Only applicable to OIS legs. A period by which the value dates schedule of (averaged, compounded) OIS legs is shifted into the past. On top of this the gap defined by the FixingDays is applied to get the final fixing date for an original date in the OIS value dates schedule. In the context of RFRs the Lookback parameter is sometimes also called “shift”. With this terminology, first the shift and then the observation lag is applied to get the fixing date for an original value date of an overnight coupon.

  Allowable values: any valid period, e.g. 2D, 3M, 1Y
- RateCutoff [Optional]: Only applicable to OIS legs. The number of fixing dates at the end of the fixing period for which the fixing value is held constant and set to the previous value. Defaults to 0.

  Allowable values: any non-negative whole number
- Spreads [Optional]: This node contains child elements of type `Spread`. If the spread is constant over the life of the floating leg, only one spread value should be entered. If two or more coupons have different spreads, multiple spread values are required, each

represented by a `Spread` child element. The first spread value corresponds to the first coupon, the second spread value corresponds to the second coupon, etc. If the number of coupons exceeds the number of spread values, the spread will be kept flat at the value of last entered spread for the remaining coupons. The number of entered spread values cannot exceed the number of coupons.

Allowable values: Each child element can take any real number. The spread is expressed in decimal form, e.g. 0.005 is a spread of 0.5% or 50 bp.

For the <`Spreads`> section, the same applies as for notionals and rates - a list of changing spreads can be specified without or with individual start dates as shown in Listing 217.

Listing 217: 'Dated' spreads

```
<Spreads>
    <Spread>0.005</Spread>
    <Spread startDate='2017-03-05'>0.007</Spread>
    <Spread startDate='2019-03-05'>0.009</Spread>
</Spreads>
```

If the entire <`Spreads`> section is omitted, it defaults to a spread of *0%*.

- Gearings [Optional]: This node contains child elements of type `Gearing` indicating that the coupon rate is multiplied by the given factors. The mode of specification is analogous to spreads, see above.

  If the entire <`Gearings`> section is omitted, it defaults to a gearing of *1*.

- Caps [Optional]: This node contains child elements of type `Cap` indicating that the coupon rate is capped at the given rate (after applying gearing and spread, if any). The mode of specification is analogous to spreads, see above. Caps / Floors are supported for Ibor, SIFMA, compounded / averaged OIS coupons, but not for coupons with subperiods.

  For OIS coupons notice how the gearing g and spread s enter the calculation of the coupon amount A dependent on the IncludeSpread and LocalCapFloor flags and the cap rate C, floor rate F, daily rates f_i, daily accrual fractions τ_i and the coupon accrual fraction τ. Notice that the gearing must be 1 if include spread is set to true for capped / floored coupons. The cases for compounded coupons are:

  – IncludeSpread = false, LocalCapFloor = false:

  $$A = \min\left(\max\left(g \cdot \frac{\prod(1+\tau_i f_i) - 1}{\tau} + s, F\right), C\right)$$

  – IncludeSpread = true, LocalCapFloor = false:

  $$A = \min\left(\max\left(\cdot\frac{\prod(1+\tau_i(f_i+s)) - 1}{\tau}, F\right), C\right)$$

  – IncludeSpread = false, LocalCapFloor = true:

$$A = g \cdot \frac{\prod(1 + \tau_i \min(\max(f_i, F), C)) - 1}{\tau} + s$$

– IncludeSpread = true, LocalCapFloor = true:

$$A = \cdot \frac{\prod(1 + \tau_i \min(\max(f_i + s, F), C)) - 1}{\tau}$$

The cases for Averaged coupons are:

– IncludeSpread = false, LocalCapFloor = false:

$$A = \min\left(\max\left(g \cdot \frac{\sum(\tau_i f_i)}{\tau} + s, F\right), C\right)$$

– IncludeSpread = true, LocalCapFloor = false:

$$A = \min\left(\max\left(\frac{\sum(\tau_i f_i)}{\tau} + s, F\right), C\right)$$

– IncludeSpread = false, LocalCapFloor = true:

$$A = g \cdot \frac{\sum(\tau_i \min(\max(f_i, F), C))}{\tau} + s$$

– IncludeSpread = true, LocalCapFloor = true:

$$A = \cdot \frac{\sum(\tau_i \min(\max(f_i + s, F), C))}{\tau}$$

- Floors [Optional]: This node contains child elements of type `Floor` indicating that the coupon rate is floored at the given rate (after applying gearing and spread, if any). The mode of specification is analogous to spreads, see above.
- NakedOption [Optional]: Optional node, if *true* the leg represents only the embedded floor, cap or collar. By convention the embedded floor (or cap) are considered long if the leg is a receiver leg, otherwise short. For a collar the floor is long and the cap is short if the leg is a receiver leg. Notice that this is opposite to the definition of a collar in 19.2.3.

  Allowable values: *true*, *false* . Defaults to *false* if left blank or omitted.
- LocalCapFloor [Optional]: Optional node, if *true* a cap (floor) will be applied to the daily rates of a compounded / averaged overnight coupon. If *false* the effective period rate will be capped (floored). The flag is ignored for coupons other than overnight coupons.

  Allowable values: *true*, *false* . Defaults to *false* if left blank or omitted.
- `FixingSchedule` [Optional]: This node allows for the specification of an explicit fixing schedule, see 19.3.4. Supported for underlying IBOR / term rate index. A given fixing will become effective as specified by FixingDays relative to the fixing schedule or by an explicit ResetSchedule.

- `ResetSchedule` [Optional]: This node allows for the specification of an explicit reset schedule, see 19.3.4, i.e. the dates on which fixings become effective. Supported for underlying IBOR / term rate index. Can be given together with FixingSchedule or FixingDays. In the latter case, the fixing dates are derived from the reset schedule.
- `HistoricalFixings` [Optional]: This node allows for the specification of an custom trade specific fixings. Supported for underlying OIS / IBOR / term rate index. If a historical fixing for date in the provided list is needed for pricing, the custom fixings will be used instead of an exisiting global index fixings.

19.3.7 Leg Data with Amortisation Structures

Amortisation structures can (optionally) be added to a leg as indicated in the following listing 218, within a block of information enclosed by <Amortizations> and </Amortizations> tags. Note that <Amortizations> structures are not supported for trade type CapFloor.

Listing 218: Amortisation data

```
<LegData>
  <LegType> ... </LegType>
  <Payer> ... </Payer>
  <Currency> ... </Currency>
  <Notionals>
    <Notional>10000000</Notional>
  </Notionals>
  <Amortizations>
    <AmortizationData>
      <Type>FixedAmount</Type>
      <Value>1000000</Value>
      <StartDate>20170203</StartDate>
      <Frequency>1Y</Frequency>
      <Underflow>false</Underflow>
    </AmortizationData>
    <AmortizationData>
      ...
    </AmortizationData>
  </Amortizations>
  ...
</LegData>
```

The user can specify a sequence of `AmortizationData` items in order to switch from one kind of amortisation to another etc. Within each `AmortisationData` block the meaning of elements is

- Type: Amortisation type with allowable values *FixedAmount, RelativeToInitialNotional, RelativeToPreviousNotional, Annuity, LinearToMaturity.*
- Value [optional]: Interpreted depending on `Type`, see below. Required for all types except LinearToMaturity.
- StartDate [optional]: Amortisation starts on first schedule date on or beyond StartDate. If not given, amortisation starts in first schedule period. If more than one AmortizationData block is specified, the StartDate is mandatory for all blocks except the first.

- EndDate [optional]: Amortization is applied for schedule periods with start date before EndDate. If more than one AmortizationData block is specified, the EndDate is mandatory for all blocks except the last.
- Frequency, entered as a period [optional]: Frequency of amortisations. If not given, an amortization is applied in each schedule period, otherwise in each *n*th period, where *n* is determined from Frequency. Amortizations are always applied to whole periods though, i.e. not within a period. The frequency is ignored for type Annuity, in which case an amortisation is applied in each period.
- Underflow [optional]: Allow amortisation below zero notional if `true`, otherwise amortisation stops at zero notional. Defaults to false;

The amortisation data block's `Value` element is interpreted depending on the chosen `Type`:

- FixedAmount: The value is interpreted as a notional amount to be subtracted from the current notional on each amortisation date.
- RelativeToInitialNotional: The value is interpreted as a fraction of the **initial** notional to be subtraced from the current notional on each amortisation date.
- RelativeToPreviousNotional: The value is interpreted as a fraction of the **previous** notional to be subtraced from the previous notional to get the current notional on each amortisation date.
- Annuity: The value is interpreted as annuity amount (redemption plus coupon).
- LinearToMaturity: The value is not relevant, and does not need to be provided.

Annuity type amortisation is supported for fixed rate legs as well as floating (ibor) legs.

Note:

- Floating annuities require at least one previous vanilla coupon in order to work out the first amortisation amount.
- Floating legs with annuity amortisation currently do not allow switching the amortisation type, i.e. only a single block of `AmortizationData`.

19.3.8 Indexings

This trade component can be used as an optional node within the `LegData` component to scale the notional of the coupons of a leg by one or several index prices. This feature is typically used within equity swaps with notional reset to align the notional of the funding leg with the one from the equity leg for all return periods. See 19.2.25 for the specific usage in equity swaps. Notice that typically it is sufficient to set the `FromAssetLeg` flag to *true* in the `Indexings` node definition, i.e.

```
<LegData>
    <LegType>Floating</LegType>
    <!-- no notionals node required -->
    <ScheduleData> ... </ScheduleData>
    <Indexings>
        <FromAssetLeg>true</FromAssetLeg>
    </Indexings>
```

```
    <FloatingLegData> ... </FloatingLegData>
</LegData>
```

which will cause the trade builder to pull all the indexing details from the asset leg (the equity leg in an equity swap) and populate the indexing data on the funding leg accordingly. Notice that no definition of a `Notionals` node is required in this case, this will also generated automatically.

In what follows we will describe the full syntax of the Indexings node below for reference. The Indexing component can be used in combination with the following leg types:

- Fixed
- Floating
- CMS
- DigitalCMS
- CMSSpread
- DigitalCMSSpread

If specified the notional of the single coupons in the leg is scaled by one or several index prices and a quantity. The indices can be equity or FX indices. Notice that if notional exchanges are enabled on a leg with the `FromAssetLeg` flag set to *true*, the notional exchanges are *not* influenced by the indexing definitions. In general we assume that notional exchanges are not enabled in combination with `FromAssetLeg` *true*, but it is not forbidden technically. Listing 219 shows an example of a Floating leg indexed by both an equity price and a FX rate.

Another use case for Indexings is for non-deliverable IR and Cross Currency Swaps. A non-deliverable IR Swap has Currency set to the deliverable currency on both legs, Notional in the non-deliverable currency on both legs, and Indexings with an FX Index between the deliverable and non-deliverable currency on both legs. See the Swap section for an example non-deliverable IR swap where USD is the payment currency and CLP is the non-deliverable currency.

A non-deliverable Cross Currency Swap has Settlement set to *Cash*, and one leg is a regular leg in the deliverable currency without Indexings. The other leg has Currency set to the deliverable currency, Notional in the non-deliverable currency and Indexings with an FX Index between the deliverable and non-deliverable currency. See the Swap section for an example USD-CLP non-deliverable cross currency swap where CLP is the non-deliverable currency.

Listing 219: Indexings node

```
<LegData>
    <LegType>Floating</LegType>
    <Notionals> ... </Notionals>
    <ScheduleData> ... </ScheduleData>
    <Indexings>
        <FromAssetLeg>false</FromAssetLeg>
        <Indexing>
            <Quantity>1000</Quantity>
            <Index>EQ-RIC:.STOXX50E</Index>
            <InitialFixing>2937.36</InitialFixing>
            <ValuationSchedule>
              <Dates>...</Dates>
              <Rules>...</Rules>
            </ValuationSchedule>
            <FixingDays>0</FixingDays>
            <FixingCalendar/>
            <FixingConvention>U</FixingConvention>
            <IsInArrears>false</IsInArrears>
        </Indexing>
        <Indexing>
            <Index>FX-ECB-EUR-USD</Index>
            <InitialFixing>1.1469</InitialFixing>
            <ValuationSchedule> ... </ValuationSchedule>
            <FixingDays>0</FixingDays>
            <FixingCalendar/>
            <FixingConvention>U</FixingConvention>
            <IsInArrears>false</IsInArrears>
        </Indexing>
    </Indexings>
    <FloatingLegData> ... </FloatingLegData>
</LegData>
```

The Indexings node contains the following elements:

- `FromAssetLeg` [Optional]: If *true*, and the trade type supports this, the notionals on the funding leg (i.e. the leg with the `FromAssetLeg` field) will be derived from the respective asset leg. Internally, the trade builder will add `Indexing` blocks automatically reflecting the necessary indexings (equity price and FX in the case of an equity swap) from the notional reset feature of the asset leg. Also, the Notionals node of the funding leg will internally be set to a single notional 1. The actual Notionals node in the XML on the funding leg is not required and can be omitted.

  `FromAssetLeg` is supported for the following trade types:

  - `EquitySwap`: Setting `FromAssetLeg` to *true*, aligns the notionals for all return periods on the non-equity funding leg, to the equity leg by deriving equity price, quantity and FX from the equity leg.
    Note that `FromAssetLeg` is only supported if `NotionalReset` is *true* on the equity leg - `FromAssetLeg` is ignored otherwise. Also `FromAssetLeg` is only supported when Quantity is given on the equity leg, not InitialPrice and Notional.
  - `BondTRS`: Setting `FromAssetLeg` to *true*, aligns the notionals for all return periods on the funding leg (in the `FundingData` block), to the total return leg (in

the `TotalReturnData` block) by deriving bond price, bond notional and FX from the total return leg, bond data and the reference bond.

Allowable values: *true*, *false*. Defaults to *false* if left blank or omitted.

- `Indexing` [Optional, an arbitrary number can be given]: Each Indexing node describes one indexing as follows:
  - Quantity [Optional]: The quantity that applies. For equity that should be the number of shares, for FX it should be 1, i.e. for FX this field can be omitted. The notional of each coupon is in general determined as
    Original Coupon Notional x Quantity x Equity Price x FX Rate
    depending on which indexing types are given. Typically, the original coupon notional will be set to 1.

    Allowable values: Any number. Defaults to *1* if left blank or omitted.
  - Index: The relevant index. This is either an equity or FX index. For an FX index, one of the currencies of the index must match the leg currency. It is then ensured that the FX conversion is applied using the correct direction, i.e. if the foreign currency of the index matches the leg currency, the reciprocals of the index fixings are used as a multiplier.

    Allowable values: This is "FX-SOURCE-CCY1-CCY" for FX, see 19.3.29 and 28 for details, or "EQ-NAME" for Equity with "Name" being the general string representation for equity underlyings IdentifierType:Name:Currency:Exchange, see 19.3.29.
  - Dirty [Optional]: Only used for bond indices. Indicates whether to use dirty (*true*) or clean (*false*) prices.

    Allowable values: *true*, *false*. Defaults to *true* if left blank or omitted.
  - Relative [Optional]: Only used for bond indices. Indicates whether to use relative (*true*) or absolute prices (*false*). The absolute price is the dirty or clean npv as of the settlement date of the bond in absolute "dollar" terms using the bond details (in particular the notional) from the reference data. The relative price is the absolute price divided by the current notional as of the settlement date.

    Allowable values: *true*, *false*. Defaults to *true* if left blank or omitted.
  - ConditionalOnSurvival [Optional]: Only used for bond indices. Indicates whether to forecast bond prices conditional on survival (*true*) or including the default probability from today until the fixing date (*false*).

    Allowable values: *true*, *false*. Defaults to *true* if left blank or omitted.
  - InitialFixing [Optional]: If given the index fixing value to apply on the fixing date of the first coupon. If not given the value is read from the relevant fixing history.

    Allowable values: any number
  - InitialNotionalFixing [Optional]: If given the index fixing value to apply on the fixing date of the first notional exchange flow. This is used in non-deliverable XCCY swaps. If not given the value is read from the relevant fixing history.

Allowable values: any number

- ValuationSchedule [Optional]: If given the schedule from which the fixing dates are deduced. If not given, it defaults to the original leg's schedule.

  If the valuation schedule has the same size as the original leg's schedule, it is assumed that the periods correspond one to one, i.e. the *i*th fixing date is derived from the *i*th (inArrears = false) or $i + 1$th (inArrears = true) date in the valuation schedule using the FixingDays, FixingCalendar and FixingConvention.

  If the valuation schedule has a different size than the original leg's schedule, the relevant valuation date for the *i*th original leg's coupon is determined as the latest valuation date that is less or equal to accrual start date (inArrears = false) resp. accrual end date (inArrears = true) of that coupon. The fixing date is derived from the relevant valuation date as above, i.e. using the FixingDays, FixingCalendar and FixingConvention.

  Allowable values: a valid schedule definition, see 19.3.4

- FixingDays [Optional]: If given defines the number of fixing days to apply when deriving the fixing dates from the valuation schedule (see above).

  Allowable values: Any non-negative whole number. Defaults to *0* if left blank or omitted.

- FixingCalendar [Optional, defaults to NullCalendar (no holidays): If given defines the fixing calendar to use when deriving the fixing dates from the valuation schedule (see above).

  Allowable values: Allowable values: See Table 24 Calendar. Defaults to the *NullCalendar* (no holidays) if left blank or omitted.

- FixingConvention [Optional]: If given defines the business day convention to use when deriving the fixing dates from the valuation schedule (see above). Defaults to *Preceding* if left blank or omitted.

  Allowable values: Any valid roll convention, e.g. (*F, MF, P, MP, U*). See Table 21.

- IsInArrears [Optional]: If *true*, the fixing dates are derived from the period end dates, otherwise from the period start dates as described for `ValuationSchedule` above.

  Allowable values: *true*, *false*. Defaults to *false* if left blank or omitted.

19.3.9 Cashflow Leg Data

A Cashflow leg is used to represent one or more custom cashflows, with specified dates and amounts. Listing 220 shows an example for a leg of type Cashflow.

Listing 220: Cashflow leg data

```
<LegData>
  <Payer>false</Payer>
  <LegType>Cashflow</LegType>
  <Currency>EUR</Currency>
  <PaymentConvention>ModifiedFollowing</PaymentConvention>
  <CashflowData>
    <Cashflow>
      <Amount date="2024-12-15">105000</Amount>
    </Cashflow>
  </CashflowData>
</LegData>
```

The CashflowData block contains the following elements:

- Cashflow: This node contains child elements of type `Amount`, each representing a cashflow. Each child element should include the date of the cashflow using the form:

  ```
  <Amount date="YYYY-MM-DD">[amount]</Amount>
  ```

  Allowable values: Each child element can take any real number.

19.3.10 CMS Leg Data

Listing 221 shows an example for a leg of type CMS.

Listing 221: CMS leg data

```
<LegData>
  <LegType>CMS</LegType>
  <Payer>false</Payer>
  <Currency>EUR</Currency>
  <Notionals>
    <Notional>10000000</Notional>
  </Notionals>
  <DayCounter>ACT/ACT</DayCounter>
  <PaymentConvention>Following</PaymentConvention>
  <ScheduleData>
    ...
  </ScheduleData>
  <CMSLegData>
    <Index>EUR-CMS-10Y</Index>
    <Spreads>
      <Spread>0.0010</Spread>
    </Spreads>
    <Gearings>
      <Gearing>2.0</Gearing>
    </Gearings>
    <Caps>
      <Cap>0.05</Cap>
    </Caps>
    <Floors>
      <Floor>0.01</Floor>
    </Floors>
    <NakedOption>false</NakedOption>
  </CMSLegData>
</LegData>
```

The CMSLegData block contains the following elements:

- Index: The underlying CMS index.

  Allowable values: See 26, a string on the form CCY-CMS-TENOR, where the CMS part stays constant and TENOR is an integer followed by Y.

- Spreads [Optional]: The spreads applied to index fixings. As usual, this can be a single value, a vector of values or a dated vector of values.

  Allowable values: Each child spread element can take any real number. The spread is expressed in decimal form, e.g. 0.005 is a spread of 0.5% or 50 bp.

- IsInArrears [Optional]: *true* indicates that fixing is in arrears, i.e. the fixing gap is calculated in relation to the current period end date.
  false indicates that fixing is in advance, i.e. the fixing gap is calculated in relation to the previous period end date.

  Allowable values: *true, false*. Defaults to *false* if left blank or omitted.

- FixingDays [Optional]: This is the fixing gap, i.e. the number of days before the period end date an index fixing is taken. Defaults to the index's fixing gap.

  Allowable values: A non-negative whole number. Defaults to the fixing days of the Ibor index the swap references if blank or omitted. See defaults per index in Table 27.

- Gearings [Optional]: This node contains child elements of type `Gearing` indicating that the coupon rate is multiplied by the given factors. The mode of specification is analogous to spreads, see above.

  If the entire <`Gearings`> section is omitted, it defaults to a gearing of *1*.

- Caps [Optional]: This node contains child elements of type `Cap` indicating that the coupon rate is capped at the given rate (after applying gearing and spread, if any). The mode of specification is analogous to spreads, see above.

- Floors [Optional]: This node contains child elements of type `Floor` indicating that the coupon rate is floored at the given rate (after applying gearing and spread, if any). The mode of specification is analogous to spreads, see above.

- NakedOption [Optional]: If *true* the leg represents only the embedded floor, cap or collar. By convention the embedded floor (or cap) are considered long if the leg is a receiver leg, otherwise short. For a collar the floor is long and the cap is short if the leg is a receiver leg.

  Allowable values: *true*, *false* . Defaults to *false* if left blank or omitted.

19.3.11 Constant Maturity Bond Leg Data

In close analogy to the CMS leg one can create a leg that is linked to a *Constant Maturity Bond* yield index. The associated leg type is *CMB*.

Listing 222 shows an example for a leg of type CMB.

Listing 222: CMB leg data

```
<LegData>
  <LegType>CMB</LegType>
  <Payer>false</Payer>
  <Currency>EUR</Currency>
  <Notionals>
    <Notional>10000000</Notional>
  </Notionals>
  <DayCounter>ACT/ACT</DayCounter>
  <PaymentConvention>Following</PaymentConvention>
  <ScheduleData>
    ...
  </ScheduleData>
  <CMBLegData>
    <Index>CMB-US-TBILL-HD-13W</Index>
    <FixingDays>2</FixingDays>
    <Spreads>
      <Spread>0.0010</Spread>
    </Spreads>
    <Gearings>
      <Gearing>2.0</Gearing>
    </Gearings>
  </CMBLegData>
</LegData>
```

The CMBLegData block contains the following elements:

- Index: The underlying CMB index.

  Allowable values: A string of the form CMB-FAMILY-TENOR, where FAMILY might consist of several tokens separated by "-" as e.g. in CMB-US-TBILL-HD or CMB-DE-BUND, and TENOR is a valid period.

- Spreads [Optional]: The spreads applied to index fixings. As usual, this can be a single value, a vector of values or a dated vector of values.

  Allowable values: Each child spread element can take any real number. The spread is expressed in decimal form, e.g. 0.005 is a spread of 0.5% or 50 bp. Defaults to zero if left blank or omitted.

- FixingDays: This is the fixing gap, i.e. the number of days before the period end date an index fixing is taken. Defaults to the index's fixing gap.

  Allowable values: A non-negative whole number. Defaults to the fixing days of the Ibor index the swap references if blank or omitted. See defaults per index in Table 27.

- IsInArrears [Optional]: *true* indicates that fixing is in arrears, i.e. the fixing gap is calculated in relation to the current period end date.
  false indicates that fixing is in advance, i.e. the fixing gap is calculated in relation to the previous period end date.

  Allowable values: *true, false*. Defaults to *false* if left blank or omitted.

- Gearings [Optional]: This node contains child elements of type `Gearing` indicating that the coupon rate is multiplied by the given factors. The mode of specification is analogous to spreads, see above.

  If the entire <`Gearings`> section is omitted, it defaults to a gearing of *1*.

Note:

- For each CMB index name one needs to maintain bond reference data with ID equal to the index name. This reference data is used to project the CMB index fixings as follows: For a future index period (from future date to future date + index tenor), a forward starting bond is constructed using the schedule frequency defined in the reference data and with constant maturity (future date + tenor). The forward bond is priced using the reference yield curve and credit curve defined in the reference data. And the bond price is then converted into a bond yield using the bond yield conventions (compounding, frequncy, price type) maintained for that same ID. If the conventions are not set up, then default values are used (compounded, annual, clean).
- For periods with start dates in the past, historical index fixings need to be provided, as for interest rate indices.
- No convexity adjustment is applied here yet, in contrast to CMS index projections.
- The CMB leg does not support Caps or Floors yet, in contrast to the CMS leg.

19.3.12 Digital CMS Leg Data

Listing 223 shows an example for a leg of type *DigitalCMS*.

Listing 223: Digital CMS leg data

```
<LegData>
  <LegType>DigitalCMS</LegType>
  <Payer>false</Payer>
  <Currency>GBP</Currency>
  <Notionals>
    <Notional>10000000</Notional>
  </Notionals>
  <DayCounter>ACT/ACT</DayCounter>
  <PaymentConvention>Following</PaymentConvention>
  <ScheduleData>
    ...
  </ScheduleData>
  <DigitalCMSLegData>
    <CMSLegData>
      <Index>EUR-CMS-10Y</Index>
      <FixingDays>2</FixingDays>
      <Gearings>
        <Gearing>3.0</Gearing>
      </Gearings>
      <Spreads>
        <Spread>0.0010</Spread>
      </Spreads>
      <NakedOption>false</NakedOption>
    </CMSLegData>
    <CallPosition>Long</CallPosition>
    <IsCallATMIncluded>false</IsCallATMIncluded>
    <CallStrikes>
      <Strike>0.003</Strike>
    </CallStrikes>
    <CallPayoffs>
      <Payoff>0.003</Payoff>
    </CallPayoffs>
    <PutPosition>Short</PutPosition>
    <IsPutATMIncluded>false</IsPutATMIncluded>
    <PutStrikes>
      <Strike>0.05</Strike>
    </PutStrikes>
    <PutPayoffs>
      <Payoff>0.05</Payoff>
    </PutPayoffs>
  </DigitalCMSLegData>
</LegData>
```

The `DigitalCMSLegData` block contains the following elements:

- CMSLegData: a `CMSLegData` block describing the underlying Digital CMS leg (see 19.3.10). Caps and floors in the underlying CMS leg are not supported for Digital CMS Options. The `NakedOption` flag in the `CMSLegData` block is supported and can be used to separate the digital option payoff from the underlying CMS coupon.
- CallPosition: Specifies whether the call option position is long or short.
- IsCallATMIncluded: inclusion flag on the call payoff if the call option ends at-the-money

- CallStrikes: strike rate for the call option
- CallPayoffs: digital call option payoff rate. If included the option is cash-or-nothing, if excluded the option is asset-or-nothing
- PutPosition: Specifies whether the put option position is long or short.
- IsPutATMIncluded: inclusion flag on the put payoff if the put option ends at-the-money
- PutStrikes: strike rate for the put option
- PutPayoffs: digital put option payoff rate. If included the option is cash-or-nothing, if excluded the option is asset-or-nothing

19.3.13 Duration Adjusted CMS Leg Data

Listing 224 shows an example for a leg of type DurationAdjustedCMS.

Listing 224: Duration Adjusted CMS leg data

```
<LegData>
  <LegType>DurationAdjustedCMS</LegType>
  <Payer>false</Payer>
  <Currency>EUR</Currency>
  <Notionals>
    <Notional>21000000</Notional>
  </Notionals>
  <DayCounter>Simple</DayCounter>
  <PaymentConvention>Following</PaymentConvention>
  <PaymentLag>0</PaymentLag>
  <PaymentCalendar>TARGET</PaymentCalendar>
  <DurationAdjustedCMSLegData>
    <Index>EUR-CMS-20Y</Index>
    <Duration>10</Duration>
    <Spreads>
      <Spread>0</Spread>
    </Spreads>
    <Gearings>
      <Gearing>1</Gearing>
    </Gearings>
    <IsInArrears>false</IsInArrears>
    <FixingDays>2</FixingDays>
    <Caps>
      <Cap>0.015</Cap>
    </Caps>
    <NakedOption>true</NakedOption>
  </DurationAdjustedCMSLegData>
  <ScheduleData>
    <Rules>
      <StartDate>2019-12-31</StartDate>
      <EndDate>2023-12-31</EndDate>
      <Tenor>3M</Tenor>
      <Calendar>EUR</Calendar>
      <Convention>ModifiedFollowing</Convention>
      <TermConvention>ModifiedFollowing</TermConvention>
      <Rule>Backward</Rule>
    </Rules>
  </ScheduleData>
</LegData>
```

The DurationAdjustedCMSLegData is identical to the CMSLegData block (see 19.3.10). In addition to this it contains a field defining the duration adjustment:

- Duration [Optional]: A non-negative whole number n that defines the duration adjustment for the coupons. If γ is the underlying CMS index fixing for a coupon the duration adjustment δ is defined as

$$\delta = \sum_{i=1}^{n} \frac{1}{(1+\gamma)^i}$$

  If n is zero or the duration is not given, δ is defined as 1, i.e. no adjustment is applied in this case and the coupon will be an ordinary CMS coupon. The coupon amount A for a duration adjusted coupon with a spread s, a gearing g, a cap c, a floor f and with

nominal N and accrual fraction τ is given by:

$$A = \delta \cdot N \cdot \tau \cdot \max(\min(g \cdot \gamma + s, c), f)$$

Allowable values: A non-negative whole number.

19.3.14 CMS Spread Leg Data

A CMS Spread leg consists of coupons that are linked to differences (spreads) of CMS index fixings with different maturities. When these are capped and/or floored, then the leg (assuming NakedOption *true*) contains a sequence of CMS Spread Options with payoff

$$N \cdot [(CMS_m - CMS_n - K)]^+ = N \cdot \max(0, CMS_m - CMS_n - K)$$

where N is the notional amount, $CMS_{n/m}$ is the n/m year CMS rate, and K is the strike.

Listing 225 shows an example for a leg of type CMSSpread.

Listing 225: CMS Spread leg data

```
<LegData>
  <LegType>CMSSpread</LegType>
  <Payer>false</Payer>
  <Currency>GBP</Currency>
  <Notionals>
    <Notional>10000000</Notional>
  </Notionals>
  <DayCounter>ACT/ACT</DayCounter>
  <PaymentConvention>Following</PaymentConvention>
  <ScheduleData>
    ...
  </ScheduleData>
  <CMSSpreadLegData>
    <Index1>EUR-CMS-10Y</Index1>
    <Index2>EUR-CMS-2Y</Index2>
    <Spreads>
      <Spread>0.0010</Spread>
    </Spreads>
    <Gearings>
      <Gearing>8.0</Gearing>
    </Gearings>
    <Caps>
      <Cap>0.05</Cap>
    </Caps>
    <Floors>
      <Floor>0.01</Floor>
    </Floors>
    <NakedOption>false</NakedOption>
  </CMSSpreadLegData>
</LegData>
```

The elements of the CMSSpreadLegData block are identical to those of the CMSLegData (see 19.3.10), except for the index which is defined by two CMS indices as the difference between `Index1` and `Index2`.

The payout for each coupon is thus:

$N \cdot (gearing \cdot (Index1 - Index2) + Spread) \cdot daycountfraction$

Adding a cap, and assuming no spread, a gearing of 1, a daycount fraction of 1, and a notional of 1, the payout becomes:

$min(Cap; Index1 - Index2)$

If there is a floor instead of a cap, the payout is:

$max(Floor; Index1 - Index2)$

Note that a CMS Spread Option can be created by setting NakedOption to *true*. With this setting, the payout for the CMS Spread leg with a cap becomes an option with the cap rate as strike:

$max(0; (Index1 - Index2) - Cap)$

And the payout for a CMS Spread leg with a floor, and with NakedOption set to *true* is :

$max(0; Floor - (Index1 - Index2))$

19.3.15 Digital CMS Spread Leg Data

Listing 226 shows an example for a leg of type *DigitalCMSSpread*.

Listing 226: Digital CMS Spread leg data

```
<LegData>
  <LegType>DigitalCMSSpread</LegType>
  <Payer>false</Payer>
  <Currency>GBP</Currency>
  <Notionals>
    <Notional>10000000</Notional>
  </Notionals>
  <DayCounter>ACT/ACT</DayCounter>
  <PaymentConvention>Following</PaymentConvention>
  <ScheduleData>
    ...
  </ScheduleData>
  <DigitalCMSSpreadLegData>
      <CMSSpreadLegData>
        <Index1>EUR-CMS-10Y</Index1>
        <Index2>EUR-CMS-2Y</Index2>
        <Spreads>
          <Spread>0.0010</Spread>
        </Spreads>
        <Gearings>
          <Gearing>8.0</Gearing>
        </Gearings>
        <NakedOption>false</NakedOption>
      </CMSSpreadLegData>
      <CallPosition>Long</CallPosition>
      <IsCallATMIncluded>false</IsCallATMIncluded>
      <CallStrikes>
          <Strike>0.0001</Strike>
      </CallStrikes>
      <CallPayoffs>
          <Payoff>0.0001</Payoff>
      </CallPayoffs>
      <PutPosition>Long</PutPosition>
      <IsPutATMIncluded>false</IsPutATMIncluded>
      <PutStrikes>
          <Strike>0.001</Strike>
      </PutStrikes>
      <PutPayoffs>
          <Payoff>0.001</Payoff>
      </PutPayoffs>
  </DigitalCMSSpreadLegData>
</LegData>
```

The `DigitalCMSSpreadLegData` block contains the following elements:

- CMSSpreadLegData: a `CMSSpreadLegData` block describing the underlying Digital CMS Spread leg (see 19.3.14). Caps and floors in the underlying CMS Spread leg are not supported for Digital CMS Spread Options. The `NakedOption` flag in the `CMSSpreadLegData` block is supported and can be used to separate the digital option payoff from the underlying CMS Spread coupon.
- CallPosition: Specifies whether the call option position is long or short.
- IsCallATMIncluded: inclusion flag on the call payoff if the call option ends at-the-money

- CallStrikes: strike rate for the call option
- CallPayoffs: digital call option payoff rate. If included the option is cash-or-nothing, if excluded the option is asset-or-nothing
- PutPosition: Specifies whether the put option position is long or short.
- IsPutATMIncluded: inclusion flag on the put payoff if the put option ends at-the-money
- PutStrikes: strike rate for the put option
- PutPayoffs: digital put option payoff rate. If included the option is cash-or-nothing, if excluded the option is asset-or-nothing

19.3.16 Equity Leg Data

Listing 227 shows an example of a leg of type Equity. Note that a resetting Equity Leg (NotionalReset set to *true*) must have either:
a) a Quantity, or
b) an InitialPrice and a Notional in the leg

The EquityLegData block contains the following elements:

- Quantity[Optional with one exception]: The number of shares. Either a Notional or the Quantity must be given for the leg, but not both at the same time.

  Quantity is optional with the exception that when FXTerms is used and NotionalReset is set to *true*, and the InitialPriceCurrency differs from the leg currency, Quantity must be given, and Notional cannot be used.

  Allowable values: Any positive real number

- ReturnType: *Price* indicates that the coupons on the equity leg are determined by the price movement of the underlying equity, i.e.: $Notional \cdot \frac{FinalPrice - InitialPrice}{InitialPrice}$,
  Total indicates that coupons are determined by the total return of the underlying equity including dividends, i.e.:
  $Notional \cdot \frac{(FinalPrice + dividends * DividendFactor) - InitialPrice}{InitialPrice}$,
  Dividend indicates that the coupons are determined by the dividened paid on the underlying equity.

  Allowable values: *Price, PRICE*, *Total, TOTAL*, *Dividend, DIVIDEND*

- Name: The identifier of the underlying equity or equity index.

  Allowable values: See `Name` for equity trades in Table 31.

- Underlying: This node may be used as an alternative to the `Name` node to specify the underlying equity. This in turn defines the equity curve used for pricing. The `Underlying` node is described in further detail in Section 19.3.29.

- InitialPrice [Optional]: Initial Price of the equity, if not present, the first valuation date is used to determine the initial price. If InitialPrice is zero then each coupon's price is just the discounted fixing from the coupon's FixingEndDate. For any divisions we assume the value is one, i.e. when NotionalReset = true we have instead Quantity = Notional. The Initial price can be either given in the currency of the equity or in the leg currency, see InitialPriceCurrency.

Allowable values: Any positive real number. If omitted or left blank it defaults to the equity price of the fixing at the valuation date associated with the start date. Note that when this valuation date is in the future the forward equity price is used.

- InitialPriceCurrency [Optional]: If an initial price is given, it can be either given in the original equity ccy or the leg currency (if these are different). This field determines in which currency the initial price is given. If omitted, it is assumed that the initial price is given in equity currency.

  Allowable values: A valid currency code, See Fiat Currencies and Minor Currencies in Table 22.

- NotionalReset [Optional]: Defaults to *true*. Notional resets only affect the equity leg. If NotionalReset is set to *true* the quantity or number of shares of the underlying equity is fixed for all the coupons on the equity leg and the Notional for a period is computed as

  Notional = Quantity x (share price at valuation date for period) x (FX conversion rate at valuation date for period)

  Notice that either a) the Quantity or b) a Notional and an explicit InitialPrice must be given in the leg data for a resettable leg. In the latter case the Quantity is computed as

  Quantity = Notional / InitialPrice

  No FX conversion is allowed if the Quantity has to be derived from the Notional and the InitialPrice.

  If NotionalReset is set to *false* the quantity of the underlying equity varies per period, as per:

  Quantity = Notional / (Equity Price at valuation date for the period)

  For the first period, the InitialPrice is the Equity Price at valuation date. Here, the Notional is taken to be the Notional specified in the leg or - if the Quantity is given - to be

  Notional = Quantity x InitialPrice

  where again the InitialPrice must be explicitly given in the leg data and no FX conversion is allowed in this case.

  Allowable values: *true* or *false*

- DividendFactor [Optional]: Factor of dividend to be included in return. Note that the DividendFactor is only relevant when the ReturnType is set to *Total*. It is not used if the ReturnType is set to *Price*.

  Allowable values: $0 <$ DividendFactor ≤ 1. Defaults to *1* if left blank or omitted.

- ValuationSchedule [Optional]: Schedule of dates for equity valuation. If used, fixing dates for equity valuation will come from the `ValuationSchedule` instead of the `ScheduleData`.

  Allowable values: A node of the same form as `ScheduleData`, (see 19.3.4). Note that the number of dates (and periods) in the `ValuationSchedule` must be the same as in the `ScheduleData`. If omitted, equity valuation dates follow the schedule of the equity leg adjusted for FixingDays.

- FixingDays [Optional]: The number of days before payment date for equity valuation. *N.B.* Only used when no valuation schedule present. The calendar used when applying the fixing days is the equity curve calendar (defined in the equity reference data for the underlying equity) combined with the FxIndex calendar (if applicable).

  Allowable values: Any non-negative integer. Defaults to *0* if left blank or omitted.

- FXTerms [Mandatory when leg and equity currencies differ]: For the case when the currency the underlying equity is quoted in, is different from the leg currency. The `FXTerm` node contains the following elements:

  – EquityCurrency [Mandatory within `FXTerms`]: Currency underlying equity is quoted in. Required if FXTerms is present.

    Allowable values: See Fiat Currencies and Minor Currencies in Table 22.

  – FXIndex [Mandatory within `FXTerms`]: Name of the index for FX fixings for the leg vs equity currency pair, e.g. FX-TR20H-EUR-USD for Thomson Reuters 20:00 EURUSD FX fixing. Required if FXTerms present.

    Allowable values: See Table 28

Listing 227: Equity leg data

```
<LegData>
  <LegType>Equity</LegType>
  <Payer>false</Payer>
  <Currency>EUR</Currency>
  <DayCounter>ACT/ACT</DayCounter>
  <PaymentConvention>Following</PaymentConvention>
  <ScheduleData>
    <Rules>
      <StartDate>2026-03-01</StartDate>
      <EndDate>2028-03-01</EndDate>
      <Tenor>3M</Tenor>
      <Calendar>TARGET</Calendar>
      <Convention>ModifiedFollowing</Convention>
      <Rule>Forward</Rule>
    </Rules>
  </ScheduleData>
  <EquityLegData>
    <Quantity>1000.0</Quantity>
    <ReturnType>Price</ReturnType>
    <Underlying>
      <Type>Equity</Type>
      <Name>.SPX</Name>
      <IdentifierType>RIC</IdentifierType>
    </Underlying>
    <InitialPrice>100</InitialPrice>
    <NotionalReset>true</NotionalReset>
    <DividendFactor>1</DividendFactor>
    <ValuationSchedule>
      <Dates>
        <Calendar>USD</Calendar>
        <Convention>ModifiedFollowing</Convention>
        <Dates>
          <Date>2026-03-01</Date>
          <Date>2026-06-01</Date>
          <Date>2026-09-01</Date>
          <Date>2026-12-01</Date>
          <Date>2027-03-01</Date>
          <Date>2027-06-01</Date>
          <Date>2027-09-01</Date>
          <Date>2027-12-01</Date>
          <Date>2028-03-01</Date>
        </Dates>
      </Dates>
    </ValuationSchedule>
    <FixingDays>0</FixingDays>
    <FXTerms>
      <EquityCurrency>USD</EquityCurrency>
      <FXIndex>FX-TR20H-EUR-USD</FXIndex>
    <FXTerms>
  </EquityLegData>
</LegData>
```

19.3.17 CPI Leg Data

A CPI leg contains a series of CPI-linked coupon payments $N\, r\, (I(t)/I_0)\, \delta$ and, if `NotionalFinalExchange` is set to *true*, a final inflation-linked redemption $(I(t)/I_0)\, N$. Each coupon and the final redemption can be subtracting the (un-inflated) notional N, i.e. $(I(t)/I_0 - 1)\, N$, see below.

Note that CPI legs with just a final redemption and no coupons, can be set up with a dates-based Schedule containing just a single date - representing the date of the final redemption flow. In this case `NotionalFinalExchange` must be set to *true*, otherwise the whole leg is empty, and the Rate is not used and can be set to any value.

Listing 228 shows an example for a leg of type CPI with annual coupons, and 229 shows an example for a leg of type CPI with just the final redemption.

The `CPILegData` block contains the following elements:

- Index: The underlying zero inflation index.

  Allowable values: See `Inflation CPI Index` in Table 29.

- Rates: The contractual fixed real rate(s) of the leg, r. As usual, this can be a single value, a vector of values or a dated vector of values.

  Note that a CPI leg coupon payment at time t is:

  $$N\, r\, \frac{I(t)}{I_0}\, \delta$$

  where:

  - N: notional
  - r: the contractual fixed real rate
  - $I(t)$: the relevant CPI fixing for time t
  - I_0: the BaseCPI
  - δ: the day count fraction for the accrual period up to time t

  Allowable values: Each rate element can take any real number. The rate is expressed in decimal form, e.g. 0.05 is a rate of 5%.

- BaseCPI [Optional]: The base CPI value I_0 used to determine the lifting factor for the fixed coupons. If omitted it will take the observed CPI fixing on startDate - observationLag.

  Allowable values: Any positive real number.

- StartDate [Optional]: The start date needs to be provided in case the schedule comprises only a single date. If the schedule has at least two dates and a start date is given at the same time, the first schedule date is taken as the start date and the supplied `StartDate` is ignored.

  Allowable values: See `Date` in Table 20.

- ObservationLag [Optional]: The observation lag to be applied. It's the amount of time from the fixing at the start or end of the period, moving backward in time, to the

inflation index observation date (the inflation fixing). Fallback to the index observation lag as specified in the inflation swap conventions of the underlying index, if not specified.

Allowable values: An integer followed by *D*, *W*, *M* or *Y*. Interpolation lags are typically expressed in a positive number of *M*, months. Note that negative values are allowed, but mean that the inflation is observed forward in time from the period start/end date, which is unusual.

- Interpolation [Optional]: The type of interpolation that is applied to inflation fixings. *Linear* interpolation means that the inflation fixing for a given date is interpolated linearly between the surrounding - usually monthly - actual fixings, whereas with *Flat* interpoltion the inflation fixings are constant for each day at the value of the previous/latest actual fixing (flat forward interpolation). Fallback to the Interpolation as specified in the inflation swap conventions of the underlying index, if not specified.

  Allowable values: *Linear, Flat*

- SubtractInflationNotional [Optional]: A flag indicating whether the non-inflation adjusted notional amount should be subtracted from the final inflation-adjusted notional exchange at maturity. Note that the final coupon payment is not affected by this flag.
  Final notional payment if *true*: $N\,(I(T)/I_0 - 1)$.
  Final notional payment if *false*: $N\,I(T)/I_0$

  Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36.
  Defaults to *false* if left blank or omitted.

- SubtractInflationNotionalAllCoupons [Optional]: A flag indicating whether the non-inflation adjusted notional amount should be subtracted from all coupons. Note that the final redemption payment is not affected by this flag.
  Coupon payment if *true*: $N\,(I(T)/I_0 - 1)$.
  Coupon payment if *false*: $N\,I(T)/I_0$

  Allowable values: Boolean node, allowing *Y, N, 1, 0, true, false* etc. The full set of allowable values is given in Table 36.
  Defaults to *false* if left blank or omitted.

- Caps [Optional]: This node contains child elements of type `Cap` indicating that the inflation indexed payment is capped; the cap is applied to the inflation index and expressed as an inflation rate, see CPI Cap/Floor in the Product Description.
  If the cap is constant over the life of the cpi leg, only one cap value should be entered. If two or more coupons have different caps, multiple cap values are required, each represented by a `Cap` child element. The first cap value corresponds to the first coupon, the second cap value corresponds to the second coupon, etc. If the number of coupons exceeds the number of cap values, the cap will be kept at at the value of last entered spread for the remaining coupons. The number of entered cap values cannot exceed the number of coupons. Notice that the caps defined under this node only apply to the cpi coupons, but not a final notional flow (if present). A cap for the final notional flow can be defined under the FinalFlowCap node.

  Allowable values: Each child element can take any real number. The cap is expressed in decimal form, e.g. 0.03 is a cap of 3%.

- Floors [Optional]: This node contains child elements of type `Floor` indicating that the inflation indexed payment is floored; the floor is applied to the inflation index and expressed as an inflation rate. The mode of specification is analogous to caps, see above. Notice that the floors defined under this node only apply to the cpi coupons, but not a final notional flow (if present). A floor for the final notional flow can be defined under the FinalFlowFloor node.

  Allowable values: Each child element can take any real number. The floor is expressed in decimal form, e.g. 0.01 is a cap of 1%.

- FinalFlowCap [Optional]: The cap to be applied to the final notional flow of the cpi leg. If not given, no cap is applied.

  Note that final and non-final inflation cap/floor strikes are quoted as a number K and converted to a price via:

  $(1 + K)^t$

  where

  K = the cap/floor rate

  t = time to expiry.

  So inflation caps/floors are caps/floors on the inflation rate and not the inflation index ratio. For example, to cap the final flow at the initial notional it should be K=0, i.e. FinalFlowCap should be 0.

  Allowable values: A real number. The FinalFlowCap is expressed in decimal form, e.g. 0.01 is a cap on the final flow at 1% of the inflation rate over the life of the trade.

- FinalFlowFloor [Optional]: The floor to be applied to the final notional flow of the cpi leg. If not given, no floor is applied.

  Allowable values: A real number. The FinalFlowFloor is expressed in decimal form, e.g. 0.01 is a floor on the final flow at 1% of the inflation rate over the life of the trade.

- NakedOption [Optional]: Optional node, if *true* the leg represents only the embedded floor, cap or collar. By convention these embedded options are considered long if the leg is a receiver leg, otherwise short.

  Allowable values: *true*, *false*. Defaults to *false* if left blank or omitted.

Whether the leg cotains a final redemption flow at all or not depends on the notional exchange setting, see section 19.3.3 and listing 207.

Listing 228: CPI leg data with capped annual coupons

```
<LegData>
  <LegType>CPI</LegType>
  <Payer>false</Payer>
  <Currency>GBP</Currency>
  <Notionals>
    <Notional>10000000</Notional>
    <Exchanges>
      <NotionalInitialExchange>false</NotionalInitialExchange>
      <NotionalFinalExchange>true</NotionalFinalExchange>
    </Exchanges>
  </Notionals>
  <DayCounter>ACT/ACT</DayCounter>
  <PaymentConvention>Following</PaymentConvention>
  <ScheduleData>
    <Rules>
      <StartDate>2016-07-18</StartDate>
      <EndDate>2021-07-18</EndDate>
      <Tenor>1Y</Tenor>
      <Calendar>UK</Calendar>
      <Convention>ModifiedFollowing</Convention>
      <TermConvention>ModifiedFollowing</TermConvention>
      <Rule>Forward</Rule>
      <EndOfMonth/>
      <FirstDate/>
      <LastDate/>
    </Rules>
  </ScheduleData>
  <CPILegData>
    <Index>UKRPI</Index>
    <Rates>
      <Rate>0.02</Rate>
    </Rates>
    <BaseCPI>210</BaseCPI>
    <StartDate>2016-07-18</StartDate>
    <ObservationLag>2M</ObservationLag>
    <Interpolation>Linear</Interpolation>
    <Caps>
       <Cap>0.03</Cap>
    </Caps>
    <Floors>
      <Floor>0.0</Floor>
    <Floors>
    <FinalFlowCap>0.03</FinalFlowCap>
    <FinalFlowFloor>0.0</FinalFlowFloor>
    <NakedOption>false</NakedOption>
    <SubtractInflationNotionalAllCoupons>false</SubtractInflationNotionalAllCoupons>
  </CPILegData>
</LegData>
```

Listing 229: CPI leg data with just the final redemption

```
<LegData>
  <Payer>false</Payer>
  <LegType>CPI</LegType>
  <Currency>GBP</Currency>
  <PaymentConvention>ModifiedFollowing</PaymentConvention>
  <DayCounter>ActActISDA</DayCounter>
  <Notionals>
    <Notional>25000000.0</Notional>
    <Exchanges>
      <NotionalInitialExchange>false</NotionalInitialExchange>
      <NotionalFinalExchange>true</NotionalFinalExchange>
    </Exchanges>
  </Notionals>
  <ScheduleData>
    <Dates>
      <Calendar>GBP</Calendar>
      <Dates>
        <Date>2020-08-17</Date>
      </Dates>
    </Dates>
  </ScheduleData>
  <CPILegData>
    <Index>UKRPI</Index>
    <Rates>
      <Rate>1.0</Rate>
    </Rates>
    <BaseCPI>280.64</BaseCPI>
    <StartDate>2018-08-19</StartDate>
    <ObservationLag>2M</ObservationLag>
    <Interpolation>Linear</Interpolation>
    <SubtractInflationNotional>true</SubtractInflationNotional>
    <SubtractInflationNotionalAllCoupons>false</SubtractInflationNotionalAllCoupons>
  </CPILegData>
</LegData>
```

19.3.18 YY Leg Data

Listing 230 shows an example for a leg of type YY. The YYLegData block contains the following elements:

- Index: The underlying zero inflation index.

  Allowable values: Any string (provided it is the ID of an inflation index in the market configuration).

- FixingDays: The number of fixing days.

  Allowable values: An integer followed by *D*,

- ObservationLag [Optional]: The observation lag to be applied. Fallback to the index observation lag as specified in the inflation swap conventions of the underlying index, if not specified.

  Allowable values: An integer followed by *D*, *W*, *M* or *Y*. Interpolation lags are typically expressed in *M*, months.

- Spreads [Optional]: The spreads applied to index fixings. As usual, this can be a single value, a vector of values or a dated vector of values.
- Gearings [Optional]: This node contains child elements of type `Gearing` indicating that the coupon rate is multiplied by the given factors. The mode of specification is analogous to spreads, see above.
- Caps [Optional]: This node contains child elements of type `Cap` indicating that the coupon rate is capped at the given rate (after applying gearing and spread, if any).
- Floors [Optional]: This node contains child elements of type `Floor` indicating that the coupon rate is floored at the given rate (after applying gearing and spread, if any).
- NakedOption [Optional]: Optional node (defaults to N), if Y the leg represents only the embedded floor, cap or collar. By convention these embedded options are considered long if the leg is a receiver leg, otherwise short.
- AddInflationNotional [Optional]: If true, the payoff will include the notional of the coupon $N \tau \frac{I_t}{I_{t-1Y}}$.
- IrregularYoY [Optional]: If true, instead of using a YoY inflation rate the coupon is based on the inflation rate during the actual coupon period, e.g. for a 6M coupon the inflation rate will be computed as $\frac{I_t}{I_{t-6m}} - 1$.

Listing 230: YY leg data

```
<LegData>
  <LegType>YY</LegType>
  <Payer>false</Payer>
  <Currency>EUR</Currency>
  <Notionals>
    <Notional>10000000</Notional>
  </Notionals>
  <DayCounter>ACT/ACT</DayCounter>
  <PaymentConvention>Following</PaymentConvention>
  <ScheduleData>
    <Rules>
      <StartDate>2016-07-18</StartDate>
      <EndDate>2021-07-18</EndDate>
      <Tenor>1Y</Tenor>
      <Calendar>UK</Calendar>
      <Convention>ModifiedFollowing</Convention>
      <TermConvention>ModifiedFollowing</TermConvention>
      <Rule>Forward</Rule>
      <EndOfMonth/>
      <FirstDate/>
      <LastDate/>
    </Rules>
  </ScheduleData>
  <YYLegData>
    <Index>EUHICPXT</Index>
    <FixingDays>2</FixingDays>
    <ObservationLag>2M</ObservationLag>
    <Interpolated>true</Interpolated>
    <Spreads>
      <Spread>0.0010</Spread>
    </Spreads>
    <Gearings>
      <Gearing>2.0</Gearing>
    </Gearings>
    <Caps>
      <Cap>0.05</Cap>
    </Caps>
    <Floors>
      <Floor>0.01</Floor>
    </Floors>
    <NakedOption>N</NakedOption>
    <AddInflationNotional>false</AddInflationNotional>
                                  <IrregularYoY>false</IrregularYoY>
  </YYLegData>
</LegData>
```

19.3.19 ZeroCouponFixed Leg Data

A Zero Coupon Fixed leg contains a series of Zero Coupon payments, i.e $(1+r)^t\ N$ for a compounded coupon. The uninflated notional N can be subtracted from the payment , i.e $((1+r)^t - 1)\ N$, see `SubtractNotional` below.

Listing 231 shows an example for a leg of type Zero Coupon Fixed.

To create a leg with only one payment, the schedule must only contain the start and end date.

Note that this can be achieved by setting the *Tenor* to *0D* or *1T* in a rules based Schedule.

Note that the DayCounter in a Zero Coupon Fixed leg is used to compute t in $(1+r)^t$ so that the series of zero coupon payments are calculated as $(1+r)^t N$. Also note that the "1/1" DayCounter is treated as "Year" DayCounter on Zero Coupon Fixed legs, (otherwise the t would always be 1).

For leg types other than Zero Coupon Fixed, the DayCounter is used to compute the "Accrual" (i.e. the accrual time period of a coupon) in a coupon payment calculated as: $N * Accrual * r$. However, the "Accrual" in the coupon formula is defaulted to 1 for the Zero Coupon Fixed leg type.

- Rates: The fixed real rate(s) of the leg. While this can be a single value, a vector of values or a dated vector of values.

  Allowable values: Each rate element can take any real number. The rate is expressed in decimal form, e.g. *0.05* is a rate of 5%.

- Compounding [Optional]: The method of compounding applied to the rate.

  Allowable values: *Simple* i.e. $(1+r*t)$, or *Compounded* i.e. $(1+r)^t$. Defaults to *Compounded* if left blank or omitted.

- SubtractNotional [Optional]: Decides whether the notional is subtracted from the compounding factor, i.e. $(1+r*t)-1$ respectively $(1+r)^t-1$, or not, i.e. $(1+r*t)$ respectively $(1+r)^t$

  Note that if NotionalFinalExchange is set to *true* an additional final uninflated notional flow N is added. So if NotionalFinalExchange is set to *true*, and SubtractNotional is set to *false*, there will be two final notional flows. It is recommended to omit NotionalFinalExchange causing it to default to *false*, and solely use SubtractNotional to determine the final notional flow.

  Allowable values: *true*, *Y* or *false*, *N*, defaults to *true* if left blank or omitted.

Listing 231: ZeroCouponFixed leg data

```
<LegData>
  <LegType>ZeroCouponFixed</LegType>
  <Payer>false</Payer>
  <Currency>EUR</Currency>
  <Notionals>
    <Notional>10000000</Notional>
  </Notionals>
  <DayCounter>Year</DayCounter>
  <PaymentConvention>Following</PaymentConvention>
  <ScheduleData>
    <Rules>
      <StartDate>2016-07-18</StartDate>
      <EndDate>2021-07-18</EndDate>
      <Tenor>0D</Tenor>
      <Calendar>UK</Calendar>
      <Convention>ModifiedFollowing</Convention>
      <TermConvention>ModifiedFollowing</TermConvention>
      <Rule>Forward</Rule>
      <EndOfMonth/>
      <FirstDate/>
      <LastDate/>
    </Rules>
  </ScheduleData>
  <ZeroCouponFixedLegData>
      <Rates>
          <Rate>0.02</Rate>
      </Rates>
      <Compounding>Simple</Compounding>
      <SubtractNotional>false</SubtractNotional>
  </ZeroCouponFixedLegData>
</LegData>
```

19.3.20 Commodity Fixed Leg

A commodity fixed leg is specified in a `LegData` node with `LegType` set to `CommodityFixed`. It is used to define a sequence of cashflows that are linked to a fixed price in a commodity derivative contract. Each cashflow has an associated *Calculation Period*. The outline of a commodity fixed leg is given in listing 232. It has the usual `LegData` elements described in section 19.3.3 and a `CommodityFixedLegData` node that is described in section 19.3.21 below. The section 19.3.23 describes some aspects of the `ScheduleData` node in the context of commodity derivatives.

19.3.21 Commodity Fixed Leg Data

The `CommodityFixedLegData` node outline is shown in listing 233. The meaning and allowable values for each node are as follows:

- `Quantities` [Optional]: this node is used to specify a constant quantity or a quantity that varies over the calculation periods. The usage of this node is analagous to the usage of the `Notionals` node as outlined in section 19.3.3. For convenience, this node can be omitted if the quantities are identical to those on a commodity floating leg, outlined in Section 19.3.22, on the same trade. In this case, the quantities from the

floating leg are used. If there is only a single commodity floating leg, as is the case in a standard swap, the quantities are taken from that leg. If there are multiple commodity floating legs on the trade, a specific commodity floating leg can be picked using the `Tag` node specified below. In other words, a `Tag` can be specified on the fixed leg and the same `Tag` specified on the floating leg from which the quantities should be taken.

- `Prices`: this node is used to specify a constant price or a price that varies over the calculation periods. The usage of this node is analagous to the usage of the `Notionals` node as outlined in section 19.3.3.
- `CommodityPayRelativeTo` [Optional]: the allowable values for this node are `CalculationPeriodStartDate`, `CalculationPeriodEndDate`, `TerminationDate`, `FutureExpiryDate`. They specify whether payment is relative to the calculation period start date, calculation period end date, leg maturity date or the future expiry date (of the corresponding cashflow on the floating leg with the same Tag as the fixed leg) respectively. The default is `CalculationPeriodEndDate`. The payment date is then further adjusted by the payment conventions outlined in section 19.3.3 i.e. `PaymentConvention` and `PaymentLag`. If explicit payment dates are given via the `PaymentDates` node described in section 19.3.3, then those explicit payment dates are used instead and adjusted by the `PaymentCalendar` and `PaymentConvention`.
- `Tag` [Optional]: The use of this node is explained in the `Quantities` resp. `CommodityPayRelativeTo` piece above.

19.3.22 Commodity Floating Leg

A commodity floating leg is specified in a `LegData` node with `LegType` set to `CommodityFloating`. It is used to define a sequence of cashflows that are linked to the price of a given commodity. Each cashflow has an associated *Calculation Period*. The price that is being referenced may be a commodity spot price or a commodity future contract settlement price. The cashflow may depend on the price observed on a single *Pricing Date* in the *Calculation Period* or it may depend on the arithmetic average of the prices over some or all of the business days in the *Calculation Period*.

The outline of a commodity floating leg is given in listing 234. It has the usual `LegData` elements described in section 19.3.3 and a `CommodityFloatingLegData` node that is described in section 19.3.24 below. Before describing the `CommodityFloatingLegData` node, we devote section 19.3.23 to the `ScheduleData` node in the context of commodity derivatives.

19.3.23 Commodity Schedules

The *Calculation Period* in a commodity derivative contract is in general specified as a period from and including a given *Start Date* to and including a given *End Date*. A commodity trade leg consists of a sequence of these *Calculation Period*s. It is important to set up the `ScheduleData` in the trade XML such that these periods are correctly represented in the VRE instrument. The `ScheduleData` allows for the creation of a list of dates that define the boundaries of the periods from the trade *Effective Date* to the trade *Termination Date*. When the `ScheduleData` is used on a commodity leg in the VRE trade XML, the `StartDate` is included in the first period and the `EndDate` is included in the final period. Each intervening

date generated by the ScheduleData is understood to be the included end date of a period with the subsequent period beginning on the day after the intervening date. The following two examples illustrate the set up of the ScheduleData.

A common commodity derivative schedule is one that has monthly periods running from and including the first calendar day in the month to and including the last calendar day in the month. For example, the contract periods may be specified as shown in table 93. The corresponding ScheduleData node that should be used to represent this in VRE XML is shown in listing 235. Note that Convention and TermConvention are set to Unadjusted and EndOfMonth is set to true to place all dates at the end of the month when generating the dates Backward from 30 Apr 2020. In general, these values should be used when generating monthly periods for commodity derivatives.

Start Date	End Date
2020-01-01	2020-01-31
2020-02-01	2020-02-29
2020-03-01	2020-03-31
2020-04-01	2020-04-30

Table 93: Commodity derivative monthly schedule.

Note that for fixed and floating commodity legs, the AdjustEndDateToPreviousMonthEnd field can be added to automatically adjust the end date to the end of the previous month:

AdjustEndDateToPreviousMonthEnd [Optional]: Only relevant for commodity legs. Allows for the EndDate to be on a date other than the end of the month. If set to *true* the given EndDate is restated to the end date to the end of previous month.

Allowable values: *true* or *false*. Defaults to false if left blank or omitted.

In certain cases, a sequence of periods may be provided which do not fit within the Rules provided by ScheduleData. In this case, one may use the Dates node provided by ScheduleData. As an example of such a case, consider table 94 which shows the periods for a commodity swap leg on the arithmetic average of the nearby month NYMEX WTI future contract settlement price. In this example, the *Calculation Period* runs from the day after the previous future contract expiry to and including the nearby month's contract expiry. In this case, we need to use explicit dates as shown in listing 236.

Start Date	End Date
2019-11-21	2019-12-19
2019-12-20	2020-01-21
2020-01-22	2020-02-20
2020-02-21	2020-03-20

Table 94: Commodity derivative explicit schedule.

19.3.24 Commodity Floating Leg Data

The CommodityFloatingLegData node outline is shown in listing 237. The meaning and allowable values for each node are as follows:

- `Name`: An identifier specifying the commodity being referenced in the leg. Table 32 lists the allowable values for `Name` and gives a description.
- `PriceType`: It is *Spot* if the leg is referencing a commodity spot price. It is *FutureSettlement* if the leg is referencing a commodity future contract settlement price.

  Allowable values: *Spot*, *FutureSettlement*
- `Quantities`: This node is used to specify a constant quantity or a quantity that varies over the calculation periods. The usage of this node is analogous to the usage of the `Notionals` node as outlined in section 19.3.3.

  Each `Quantity` is the number of units of the underlying commodity covered by the transaction or calculation period. The unit type is defined in the underlying contract specs for the commodity name in question. For avoidance of doubt, the `Quantity` is the number of units of the underlying commodity, not the number of contracts.
- `CommodityQuantityFrequency` [Optional]: In some cases, the quantity in a commodity derivatives contract is given as a quantity per time period. This quantity is then multiplied by the number of such time periods in each calculation period to give the quantity relevant for that full calculation period. The `CommodityQuantityFrequency` can be set to
  - *PerCalculationPeriod*: This indicates that quantitie(s) as given are for the full calculation period and that no multiplication or alteration is required. This is the default setting if this node is omitted.
  - *PerPricingDay*: This indicates that the quantitie(s) are to be considered per pricing date. In general, this can be seen on averaging contracts where the quantity provided must be multiplied by the number of pricing dates in the averaging period to give the quantity applicable for the full calculation period i.e. the quantity to which the average price over the period is applied.
  - *PerHour*: This indicates that quantitie(s) are to be considered per hour. This is common in the electricity markets. The quantity then must be multiplied by the hours per day to give the quantity for a given pricing date. Also, if the contract is averaging, the resulting daily amount is multiplied by the number of pricing dates in the period to give the quantity for the full calculation period. Note that the hours per day may be specified in the `HoursPerDay` node directly. If it is omitted, it is looked up in the conventions associated with the commodity. If it is not found there and *PerHour* is used, an exception is thrown during trade building.
  - *PerCalendarDay*: This indicates that quantitie(s) are to be considered per calendar day in the period. In other words, the quantity provided is multiplied by the number of calendar days in the period to give the quantity applicable for the full calculation period.
  - *PerHourAndCalendarDay*: This indicates that quantitie(s) are to be considered per hour and per calendar day in the period. In other words, the quantity provided is multiplied by the number of calendar days and number of hours per day in the period to give the quantity applicable for the full calculation period. The number of hours per period is corrected by daylight saving hours as specified in the conventions of the commodity.

Allowable values: *PerCalculationPeriod*, *PerPricingDay*, *PerHour*, *PerCalendarDay*, *PerHourAndCalendarDay*. Defaults to *PerCalculationPeriod* if omitted.

- `CommodityPayRelativeTo` [Optional]: The allowable values for this node are `CalculationPeriodStartDate`, `CalculationPeriodEndDate`, `TerminationDate`, `FutureExpiryDate`. They specify whether payment is relative to the calculation period start date, calculation period end date, leg maturity date or the future expiry date (not allowed for averaging legs) respectively. The default is `CalculationPeriodEndDate`. The payment date is then further adjusted by the payment conventions outlined in section 19.3.3 i.e. `PaymentConvention` and `PaymentLag`. If explicit payment dates are given via the `PaymentDates` node described in section 19.3.3, then those explicit payment dates are used instead and adjusted by the `PaymentCalendar` and `PaymentConvention`.

  Allowable values: *CalculationPeriodStartDate*, *CalculationPeriodEndDate*, *TerminationDate*. Defaults to *CalculationPeriodEndDate* if omitted.

- `Spreads` [Optional]: This node allows for the addition of an optional spread to the referenced commodity price in each calculation period. The usage of this node is exactly as described in section 19.3.6, except that for a Commodity leg, the Spread is not a percentage but an amount in the currency the commodity is quoted in.

  Allowable values: Each child `Spread` element can take any real number. Defaults to zero spread in each calculation period if the `Spreads` node is omitted.

- `Gearings` [Optional]: This node allows for the multiplication of the referenced commodity price in each calculation period by an optional gearing factor. The usage of this node is exactly as described in section 19.3.6. If the `Gearings` node is omitted, the gearing is one in each calculation period. Note that any spread is added to the referenced price before the gearing is applied.

- `PricingDateRule` [Optional]: The allowable values are *FutureExpiryDate* and *None*. This setting is ignored when `IsAveraged` is *true* or when `PriceType` is *Spot*. In particular, when there is no averaging and the leg is referencing a commodity future contract price, setting `PricingDateRule` to *FutureExpiryDate* ensures that the future contract price is observed on its expiry date i.e. that the *Pricing Date* is the future contract expiry date. The particular future contract being referenced is determined by the `IsInArrears` node and the `FutureMonthOffset` node. If `IsInArrears` is *true*, a base date is set as the calculation period end date. If `IsInArrears` is *false* a base date is set as the calculation period start date. The base date's month and year is then possibly moved forward by an integral number of months using the `FutureMonthOffset` node value. If this node value is zero, the base date's month and year are unchanged. The *Pricing Date* is then the expiry date of the future contract with base date month and base date year. Setting `PricingDateRule` to *None* allows the *Pricing Date* to be determined using the `PricingCalendar` and `PricingLag` below.

  Allowable values: *FutureExpiryDate*, *None*. Defaults to *FutureExpiryDate* if omitted.

- `PricingCalendar` [Optional]: This is the business day calendar used to determine pricing date(s) and in the application of the `PricingLag` if provided. If it is omitted, the calendar that has been set up for the reference commodity future contract or referenced commodity spot price will be used.

- `PricingLag` [Optional]: Any non-negative integer is allowed here. This node indicates that the *Pricing Date* is this number of business days before a given base date. The base date is the period start date if `IsInArrears` is *true* and it is the period end date if `IsInArrears` is *false*. This setting is not used when `IsAveraged` is *true*.

  Allowable values: Any non-negative integer. Defaults to zero if omitted.

- `PricingDates` [Optional]: This node is not used when `IsAveraged` is *true*. When `IsAveraged` is *false*, this node allows the *Pricing Date* in each period to be given an explicit value. If this node is included, it must contain the same number of `PricingDate` nodes as calculation periods. In general, this node is omitted but is used when the other options do not give the desired *Pricing Date* as specified in the trade's contractual terms.

- `IsAveraged` [Optional]: This node is set to *true* if the *Floating Price* is the arithmetic average of the commodity reference price over each business day in the calculation period. This node is set to *false* if there is no averaging of the underlying commodity price. Note that `IsAveraged` must be set to *true* if the `Name` given references a future contract that is averaging itself. There is more on this below.

  Allowable values: *true*, *false*. Defaults to *false* if omitted.

- `IsInArrears` [Optional]: This node is not used when `IsAveraged` is *true*. Although, if the observed underlying is averaging itself, having `IsAveraged` set to *true* would be ignored with regards this node. As noted above, this setting determines a base date from which the *Pricing Date* is determined. The base date is the period end date if `IsInArrears` is *true* and it is the period start date if `IsInArrears` is `false`. How the *Pricing Date* is then determined from this base date is determined by the `PricingDateRule` node or the `PricingCalendar` and `PricingLag` nodes.

  Allowable values: *true*, *false*. Defaults to *true* if omitted.

- `FutureMonthOffset` [Optional]: This node allows any non-negative integer value. If this node is omitted, it is set to zero. The node has a different usage depending on whether `IsAveraged` is *true* or *false*:

  - If `IsAveraged` is *true*, this node indicates which future contract is being referenced on each *Pricing Date* in the calculation period by acting as an offset from the next available expiry date. If `FutureMonthOffset` is zero, the settlement price of the next available monthly contract that has not expired with respect to the *Pricing Date* is used as the price on that *Pricing Date*. If `FutureMonthOffset` is one, the settlement price of the second available monthly contract that has not expired with respect to the *Pricing Date* is used as the price on that *Pricing Date*. Similarly for other positive values of `FutureMonthOffset`.

  - If `IsAveraged` is *false*, this node acts as an offset for the contract month and is used in conjunction with the `IsInArrears` setting to determine the future contract being referenced. If `IsInArrears` is `true`, a base date is set as the calculation period end date. If `IsInArrears` is *false*, a base date is set as the calculation period start date. If `FutureMonthOffset` is zero, the future contract month and year is taken as the base date's month and year. If `FutureMonthOffset` is one, the future contract month and year is taken as the month following the base date's month and year and so on for all positive values

of `FutureMonthOffset`.

- `DeliveryRollDays` [Optional]: This node allows any non-negative integer value and is only applicable when `IsAveraged` is *true*. When averaging a commodity future contract price during a calculation period, where the calculation period includes the contract expiry date, this node's value indicates when we should begin using the next future contract prices in the averaging. If the value is zero, we should include the contract prices up to and including the contract expiry. If the value is one, we should include the contract prices up to and including the day that is one business day before the contract expiry and then switch to using the next contract prices thereafter. Similarly for other non-negative integer values.

  Allowable values: Any non-negative integer. Defaults to zero if omitted.

- `IncludePeriodEnd` [Optional]: If this node is set to *true*, the period end date is included in the calculation period. If it is set to *false*, the period end date is excluded from the calculation period. There is more about this in the section 19.3.23. If this node is omitted, it is set to *true*. In general, this node should be omitted and allowed to take its default value.

- `ExcludePeriodStart` [Optional]: If this node is set to *true*, the period start date is excluded from the calculation period. If it is set to *false*, the period start date is included from the calculation period. There is more about this in the section 19.3.23. If this node is omitted, it is set to *true*. In general, this node should be omitted and allowed to take its default value.

- `HoursPerDay` [Optional]: This node is used if `CommodityQuantityFrequency` is set to *PerHour* or *PerHourAndCalendarDay*. It is described above under `CommodityQuantityFrequency`.

  Allowable values: A number between 0 and 24. If omitted it defaults to the value of the `HoursPerDay` node in the conventions for the referenced commodity.

- `UseBusinessDays` [Optional]: A boolean flag that defaults to *true* if omitted. It is not applicable if `IsAveraged` is *false*. When set to *true*, the pricing dates in the averaging period are the set of `PricingCalendar` good business days. When set to `false`, the pricing dates in the averaging period are the complement of the set of `PricingCalendar` good business days. This may be useful in certain situations. For example, the contract ICE PW2 with specifications here averages the PJM Western Hub locational marginal prices over each day in the averaging period that is a Saturday, Sunday or NERC holiday. So, in this case, `UseBusinessDays` would be *false* and `PricingCalendar` would be `US-NERC` to generate the correct pricing dates in the averaging period.

  Allowable values: *true*, *false*. Defaults to *true* if omitted.

- `UnrealisedQuantity` [Optional]: A boolean flag that defaults to *false* if omitted. This is a rarely used flag. When set to *true*, it allows the user, on a given valuation date, to enter the current period quantity as an amount remaining in the current period after the valuation date i.e. the unrealised portion of the current period's quantity. This unrealised quantity is then scaled up internally to give the quantity over the full period.

  Allowable values: *true*, *false*. Defaults to *false* if omitted.

- `LastNDays` [Optional]: This node allows a positive integer value less than or equal to 31 and is currently only supported when `PriceType` is `FutureSettlement`. When included, instead of the commodity future price being observed on the single *Pricing Date* in the period, it is observed on the `LastNDays` *Pricing Date*s, up to and including the original *Pricing Date*, for which future settlement prices are available.
- `Tag` [Optional]: This node takes any string and can be used to link the floating leg with a fixed leg that has not explicitly provided its own quantities. This can be useful in situations where the quantities on the floating leg are specified with a `CommodityQuantityFrequency` that is not simply `PerCalculationPeriod`. The fixed leg does not have the `CommodityQuantityFrequency` field. In these cases, the fixed leg can omit its `Quantities` node and take the quantities from the floating leg. This `Tag` node allows the fixed leg to link to a specific floating leg if there is more than one floating leg on the trade i.e. the fixed leg must just have the same `Tag`. The link is also used to set the payment dates of the fixed leg if CommodityPayRelativeTo is set to FutureExpiryDate.
- `DailyExpiryOffset` [Optional]: This node allows any non-negative integer value. It only has effect the underlying commodity `Name` is not being averaged and has a daily contract frequency.

  If this node is omitted, it defaults to zero. This node indicates which future contract is being referenced on each *Pricing Date* by acting as a business day offset, using the commodity `Name`'s expiry calendar, from the *Pricing Date*. It is useful e.g. in the base metals market where a future contract on each *Pricing Date* is the cash contract on that *Pricing Date* i.e. the contract with expiry date two business days after the *Pricing Date*. In this case, the `DailyExpiryOffset` would be set to `2`.
- `FXIndex` [Optional]: If `IsAveraged` is *true* this node allows the fx conversion to be applied daily in the computation of averaged cash flows. It cannot be used with the `Indexing` node.

  Allowable values: See Table 28 for supported fx indices.
- `AvgPricePrecision` [Optional]: This is only applicable when averaging is enabled. Allowed values: non-negative integer. Specifies the number of decimal places to which the average price should be rounded.

We note above that `IsAveraged` must be set to *true* if the `Name` given references a future contract that is averaging itself. For the avoidance of doubt, this does not lead to the prices of the averaging future contract being averaged in each calculation period. Instead, a check is performed in the code if the contract defined by `Name` is averaging, and if the leg itself is averaging we switch to observing the averaging future contract price on the single *Pricing Date* determined by the `PricingDateRule` node or the `PricingCalendar` and `PricingLag` nodes or the `PricingDates` node. This is best illustrated using an example. Suppose that we have a commodity swap with the schedule shown in table 95. Suppose that the *Floating Price* for the swap is specified as *For each Calculation Period, the arithmetic average of the Commodity Reference Price, for each Commodity Business Day in the Calculation Period* and that the *Commodity Reference Price* is specified as *OIL-WTI-NYMEX* with *Delivery Date* of *First Nearby Month*. There are two approaches to setting up the XML for this commodity floating leg:

1. The first approach is shown in listing 238. Note that the `Name` is `NYMEX:CL` to indicate the NYMEX WTI future contract, `IsAveraged` is `true` and `FutureMonthOffset` is `0` to indicate that we are using the nearby month contract price in the averaging. This approach is clear.
2. The second approach is to use the `CommodityFloatingLegData` shown in listing 239. Note that we have changed the `Name` to `NYMEX:CSX` to reference the NYMEX WTI Financial Futures contract. This future contract settlement price at expiry is the exact payoff of the swap leg in that it is the arithmetic average of the nearby month NYMEX WTI future contract settlement prices over the calendar month. The contract details are given here. We keep `IsAveraged` set to `true`. If we set `IsAveraged` to `false`, an error will be thrown. When `IsAveraged` is set to `true` and the `Name` references a future contract that is averaging, it is understood that the commodity leg is to use the same averaging as the future contract. In this case, we switch to a non-averaging cashflow in the code and read the averaged price directly off the price curve that we have set up using the averaging future contract prices.

In some cases, we will only have an averaging future contract available as an allowable `Name` value. For example, `NYMEX:A7Q` is one such instance. The contract details are given here. This future contract's price at the end of each contract month is the *arithmetic average of the OPIS Mt. Belvieu Natural Gasoline (non-LDH) price for each business day during the contract month*. The corresponding commodity floating leg would be set up with `Name` set to `NYMEX:A7Q` and `IsAveraged` set to `true`. Again, for the avoidance of doubt, we are not averaging the averaging future contract price. Instead, we switch to a non-averaging cashflow in the code and read the averaged price directly off the price curve that we have built out of `NYMEX:A7Q` future contract prices. We are pricing a leg that has the same payoff as the future contract.

If we have an averaging coupon and the valuation date is during the coupon period, the choice between the first and second approach above will have an effect on the sensitivities that are generated for that one single coupon. It should not affect the NPV of the coupon. The effect becomes more pronounced as the number of days remaining in the coupon period reduce. In the first approach, the coupon is priced by reading the expected future prices on future *Pricing Date*s off the non-averaging future price curve and fetching past fixed settlement prices on past *Pricing Date*s. All of these prices are then averaged. It is clear that as the valuation date approaches the final date in the coupon period, the sensitivity decreases because any bump in the curve used for pricing is only affecting the values on the remaining future *Pricing Date*s. In the second approach, the average price relevant for the full coupon period is read directly off the averaging future price curve. Any bump to the averaging future price curve affects the full coupon regardless of the position of the valuation date in the coupon period. The sensitivity will therefore be larger than using the first approach and the difference will become more noticeable as the valuation date moves towards the end of the coupon period. This subtlety can lead to differences that are larger than expected on basis swaps with averaging coupons and short maturities. If one commodity floating leg references a non-averaging price curve and the other leg references an averaging price curve, the differing effects of the bump outlined above on each leg can lead to a larger than expected net sensitivity.

Start Date	End Date	Quantity Per Period
2019-09-01	2019-09-30	5,000
2019-10-01	2019-10-31	5,000

Table 95: Example commodity swap schedule.

19.3.25 Equity Margin Leg

An equity margin leg is specified in a `LegData` node with `LegType` set to `EquityMargin`. It is used to define a sequence of cashflows that are linked to an equity price and it's associated margin factor. Each cashflow has an associated *Calculation Period*. This leg is typically used to represent a part of a Total Return Swap (TRS) on an Equity Index Future. The full TRS on the Equity Index Future uses TradeType *Swap*, and one leg of type *Equity*, and the other leg of type *EquityMargin*. Note that the equity identifier on both legs (the Name field) should be for the Equity Index, and not the Future.

The outline of a equity margin leg is given in listing 240. It has the usual `LegData` elements described in section 19.3.3 and a `EquityMarginLegData` node that is described in section 19.3.26 below.

19.3.26 Equity Margin Leg Data

The `EquityMarginLegData` node outline is shown in listing 240. The meaning and allowable values for each node are as follows:

- `Rates`: The fixed real rate(s) of the leg. While this can be a single value, a vector of values or a dated vector of values. Allowable values: Each rate element can take any real number. The rate is expressed in decimal form, e.g. *0.05* is a rate of 5%..
- `InitialMarginFactor`: this node is used to specify the equity margin factor for the first period of the trade. It's a percentage that reflecting the current applicable official Exchange initial margin requirement. It is expressed in decimal form, e.g. *0.05* is a rate of 5%..
- `EquityLegData`: this node is used to specify the underlying equity details. It's values are as outlined in section 19.3.16.
- `Multiplier` [Optional]: in some cases, the cashflow amounts are multiplied by a fixed amount. Defaults to 1.

19.3.27 CDS Reference Information

This trade component can be used to define the reference entity, tier, currency and documentation clause in credit derivative trades. For example, it can be used in the `CreditDefaultSwapData` section in a CDS trade and in the `BasketData` section in credit derivatives involving more than one underlying reference entity. The value for each of these fields is generally agreed and specified in the credit derivative contract and they determine the credit curve that is used in pricing the trade.

Listing 242: CDS reference information node

```
<ReferenceInformation>
  <ReferenceEntityId>...</ReferenceEntityId>
  <Tier>...</Tier>
  <Currency>...</Currency>
  <DocClause>...</DocClause>
</ReferenceInformation>
```

The meanings and allowable values of the elements in the `ReferenceInformation` node are as follows:

- `ReferenceEntityId`: This is typically a six digit Markit RED code specifying the underlying reference entity with the prefix `RED:` e.g. `RED:008CA0`.
- `Tier`: The debt tier that is applicable for the specified reference entity in the credit derivative. Table 33 provides the allowable values.
- `Currency`: The currency that is applicable for the specified reference entity in the credit derivative. Table 22 provides the allowable values.
- `DocClause`: The documentation clause that is applicable for the specified reference entity in the credit derivative. This defines what constitutes a credit event for the contract as well as any limitations on the deliverable debt in the event of a credit event. Table 34 provides the allowable values.

19.3.28 Basket Data

This trade component node is used in credit derivative trades referencing more than one reference entity e.g. in the `IndexCreditDefaultSwapData` node of an index CDS trade. It contains `Name` sub-nodes with the details of each constituent reference entities (names) of the basket. An example structure of the `BasketData` trade component node is shown in Listing 243.

Listing 243: Basket data

```
<BasketData>
  <Name>
    <IssuerId>CPTY_1</IssuerId>
    <CreditCurveId>RED:...</CreditCurveId>
    <Notional>100000.0</Notional>
    <Currency>USD</Currency>
  </Name>
  <Name>
    <IssuerId>CPTY_2</IssuerId>
    <CreditCurveId>RED:...</CreditCurveId>
    <Notional>100000.0</Notional>
    <Currency>USD</Currency>
  </Name>
  <Name>
    <IssuerId>CPTY_3</IssuerId>
    <CreditCurveId>RED:...</CreditCurveId>
    <Notional>100000.0</Notional>
    <Currency>USD</Currency>
  </Name>
  ...
</BasketData>
```

The meanings and allowable values of the elements in each `Name` sub-node of the `BasketData` node follow below.

- IssuerId: A unique identifier for the index component reference entity. For informational purposes and not used for pricing.

  Allowable values: Any alphanumeric string.

- CreditCurveId: The unique identifier of the index component defining one of the default curves used for pricing. The pricing can be set up to either use the curve identifiers of the index components, or one single index curve id defined in the trade specific data. A `ReferenceInformation` node may be used in place of this `CreditCurveId` node.

  Allowable values: See `CreditCurveId` for credit trades - single name in Table 30. Duplicate CreditCurveId:s are not allowed.

- `ReferenceInformation`: This node may be used as an alternative to the `CreditCurveId` node to specify the reference entity, tier, currency and documentation clause for the basket constituent. This in turn defines the credit curve used for this basket constituent in the pricing. The `ReferenceInformation` node is described in further detail in Section 19.3.27.

- Notional: The notional of the index component reference entity. Note that the sum of index component notionals (all names) must match the fixed premium leg notional. Allowable values: Any positive real number.

- Weight: Can be used, instead of Notional, to specify the weight of the index component reference entity. Note that the sum of index component notionals (all names) must match 1. Allowable values: Any positive real number.

- Currency: Defines the currency of the component, only mandatory together with a given notional.

19.3.29 Underlying

This trade component can be used to define the underlying entity for an Equity, Commodity or FX trade, but it can also define an underlying interest rate, inflation index, credit name or an underlying bond. It can be used for a single underlying, or within a basket with associated weight. For an equity underlying a string representation is used to match `Underlying` node to required configuration and reference data. The string representation is of the form IdentifierType:Name:Currency:Exchange, with all entries optional except for Name.

Listing 244: Underlying node

```
<Underlying>
  <Type>...</Type>
  <Name>...</Name>
  <Weight>...</Weight>
  <Currency>...</Currency>
  <IdentifierType>...</IdentifierType>
  <Exchange>...</Exchange>
  <PriceType>...</PriceType>
  <FutureMonthOffset>...</FutureMonthOffset>
  <DeliveryRollDays>...</DeliveryRollDays>
  <DeliveryRollCalendar>...</DeliveryRollCalendar>
</Underlying>
```

Example structures of the `Underlying` trade component node are shown in Listings 245 and 246 for an equity underlying, in Listing 249 for an fx underlying, in Listing 250 for a commodity underlying, in Listing 251 for an underlying interest rate index, in Listing 252 for an underlying inflation index, in Listing 253 for an underlying credit name, in listing 254 for an underlying bond.

Listing 245: Equity Underlying - RIC

```
        <Underlying>
            <Type>Equity</Type>
            <Name>.SPX</Name>
            <Weight>1.0</Weight>
            <IdentifierType>RIC</IdentifierType>
        </Underlying>
```

Listing 246: Equity Underlying - ISIN

```
        <Underlying>
            <Type>Equity</Type>
            <Name>NL0000852580</Name>
            <Weight>1.0</Weight>
            <IdentifierType>ISIN</IdentifierType>
            <Currency>EUR</Currency>
            <Exchange>XAMS</Exchange>
        </Underlying>
```

Listing 247: Equity Underlying - FIGI

```
<Underlying>
    <Type>Equity</Type>
    <Name>BBG000BLNNV0</Name>
    <IdentifierType>FIGI</IdentifierType>
</Underlying>
```

Listing 248: Equity Underlying - Bloomberg Identifier (Parsekey)

```
<Underlying>
    <Type>Equity</Type>
    <Name>BARC LN Equity</Name>
    <IdentifierType>BBG</IdentifierType>
</Underlying>
```

Listing 249: FX Underlying

```
<Underlying>
  <Type>FX</Type>
  <Name>ECB-EUR-USD</Name>
  <Weight>1.0</Weight>
</Underlying>
```

Listing 250: Commodity Underlying

```
<Underlying>
  <Type>Commodity</Type>
  <Name>NYMEX:CL</Name>
  <Weight>1.0</Weight>
  <PriceType>FutureSettlement</PriceType>
  <FutureMonthOffset>0</FutureMonthOffset>
  <DeliveryRollDays>0</DeliveryRollDays>
  <DeliveryRollCalendar>TARGET</DeliveryRollCalendar>
  <FutureContractMonth>Nov2023</FutureContractMonth>
</Underlying>
```

Listing 251: InterestRate Underlying

```
<Underlying>
  <Type>InterestRate</Type>
  <Name>USD-CMS-10Y</Name>
  <Weight>1.0</Weight>
</Underlying>
```

Listing 252: Inflation Index Underlying

```
        <Underlying>
          <Type>Inflation</Type>
          <Name>USCPI</Name>
          <Weight>1.0</Weight>
          <!-- optional -->
          <Interpolation>Linear</Interpolation>
</Underlying>
```

Listing 253: Credit Underlying

```
        <Underlying>
          <Type>Credit</Type>
          <Name>ISSUER_A</Name>
          <Weight>1.0</Weight>
        </Underlying>
```

Listing 254: Bond Underlying

```
      <Underlying>
        <Type>Bond</Type>
        <Name>US69007TAB08</Name>
        <IdentifierType>ISIN</IdentifierType>
        <Weight>0.5</Weight>
        <BidAskAdjustment>-0.0025</BidAskAdjustment>
      </Underlying>
```

The meanings and allowable values of the elements in the `Underlying` node are as follows:

- `Type`: The type of the Underlying asset.

  Allowable values: *Equity*, *FX*, *Commodity*, *InterestRate*, *Inflation*, *Credit*, *Bond*

- `Name`: The name of the Underlying asset.

  Allowable values:

  Equity: See `Name` for equity trades in Table 31

  FX: A string on the form SOURCE-CCY1-CCY2, where SOURCE is the FX fixing source, and the fixing is expressed as amount in CCY2 per one unit of CCY1. See Table 28, and note that the FX- prefix is not included in `Name` as it is already included in `Type`.

  InterestRate: Any valid interest rate index name, see Table 26

  Inflation: Any valid zero coupon inflation index (CPI) name, See Table 29

  Credit: Any valid credit name with a configured default curve, see Table 30

  Bond: Any valid bond identifier, the bond must be set up in the reference data.

  Commodity: An identifier specifying the commodity being referenced in the leg. Table 32 lists the allowable values for `Name` and gives a description.

- `Weight` [Optional]: The relative weight of the underlying if part of a basket. For a single underlying this can be omitted or set to 1.

  Allowable values: A real number. Defaults to 1 if left blank or omitted. A value of zero means that the underlying is excluded from the basket.

  Notes on negative weights in the *TotalReturnSwap* trade type:

  Negative weights for EquityOptionPositions are allowed, but not recommended. A negative weight for an EquityOptionPosition is equivalent to inverting the LongShort flag in the respective OptionData node.

  For EquityPositions a negative weight means that flows are in the opposite direction of the Payer flag on the return leg. A use case for negative weights is for a basket of EquityPositions that include both long and short positions.

- `IdentifierType` [Optional]: Only valid when `Type` is *Equity* or *Bond*. The type of the identifier being used.

  Allowable values: *RIC*, *ISIN*, *FIGI*, *BBG*. Defaults to *RIC*, if left blank or omitted, and `Type`: is *Equity*.

- `Currency` [Mandatory when `IdentifierType` is *ISIN*]: Only valid when `Type` is *Equity*. The currency the underlying equity is quoted in. Used when `IdentifierType` is *ISIN*, to - together with the `Exchange` convert a given ISIN to a RIC code.

  Allowable values: See Table 22 `Currency`. Mandatory when `IdentifierType` is *ISIN*, and should not be used for other `IdentifierType`:s When `Type` is *Equity*, Minor Currencies in Table 22 are also allowable.

- `Exchange` [Mandatory when `IdentifierType` is *ISIN*]: Only valid when `Type` is *Equity*. A string code representing the exchange the equity is traded on. Used when `IdentifierType` is *ISIN*, to - together with the `Currency` convert a given ISIN to a RIC code.

  Allowable values: The MIC code of the exchange, see Table 35. Mandatory when `IdentifierType` is *ISIN*, and should not be used for other `IdentifierType`:s.

- `PriceType` [Optional]: Only valid when `Type` is *Commodity*. Whether the Spot or Future price is referenced.

  Allowable values: *Spot*, *FutureSettlement*. Mandatory when `Type` is *Commodity* .

- `FutureMonthOffset` [Optional]: Only valid when `Type` is *Commodity*. Only relevant for the *FutureSettlement* price type, in which case the $N + 1$th future with expiry greater than ObservationDate for the given commodity underlying will be referenced.

  Allowable values: An integer. Mandatory for when `Type` is *Commodity* and `PriceType` is *FutureSettlement*.

- `DeliveryRollDays` [Optional]: Only valid when `Type` is *Commodity*. The number of days the observation date is rolled forward before the next future expiry is looked up.

  Allowable values: An integer. Defaults to 0 if left blank or omitted, and `Type`: is *Commodity*.

- `DeliveryRollCalendar` [Optional]: Only valid when `Type` is *Commodity*. The calendar used to roll forward the observation date.

  Allowable values: See Table 24. Defaults to the null calendar if left blank or omitted, and `Type`: is *Commodity*.

- `FutureContractMonth` [Optional]: Only valid when `Type` is *Commodity*, `PriceType` is FutureSettlement and there is no `FutureExpiryDate` node. It specifies the underlying future contract month in the format *MonYYYY*, for example Nov2023.

- `FutureExpiryDate` [Optional]: Only valid when `Type` is *Commodity*, `PriceType` is FutureSettlement and there is no `FutureContractMonth` node. This gives the expiration date of the underlying commodity future contract.

  If the field `FutureExpiryDate` and `FutureContractMonth` are omitted, the expiration date of the underlying commodity future contract is set to the prompt future, adjusted for any `FutureMonthOffset`.

- `Interpolation` [Optional]: Only valid when `Type` is *Inflation*. The index observation interpolation between fixings.

  Allowable values: Flat, Linear

- `BidAskAdjustment` [Optional]: Only valid when `Type` is *Bond*. A correction applied to the price found in the market data (usually mid), if the bond basket price is defined on the bid or ask side rather than mid.

  Allowable values: Any real number.

19.3.30 StrikeData

This trade component that can be used to define the strike entity for commodity, equity and bond options. It can be used to define either a Price or Yield strike, with examples below in 255 and 256 respectively.

Listing 255: Strike Price

```
<StrikeData>
        <StrikePrice>
                <Value>1</Value>
                <Currency>EUR</Currency>
        </StrikePrice>
</StrikeData>
```

The meanings and allowable values of the elements in the `StrikePrice` node are as follows:

- `Value`: The strike price.

  Allowable values: Any positive real number.

- `Currency` [Mandatory for Quanto/Compo, Optional otherwise]: The currency of the amount given in `Value`, i.e. the strike currency.

  Note:

Quanto: The payment/leg currency and the currency the underlying asset is quoted in differ. The strike currency is in the currency the asset is quoted in.
Compo (Composite): The payment/leg currency and the currency the underlying asset is quoted in differ. The strike currency is in the payment/leg currency.

Allowable values: See Table 22 `Currency`. Minor Currencies in Table 22 are also allowable. In non-quanto/compo cases, if left blank or omitted, it defaults to the currency of the leg for equity and commodity options, and to the currency the underlying bond is quoted in for BondOptions using reference data.

Listing 256: Strike Yield

```
<StrikeData>
        <StrikeYield>
                <Yield>0.055</Yield>
                <Compounding>SimpleThenCompounded</Compounding>
        </StrikeYield>
</StrikeData>
```

The meanings and allowable values of the elements in the `StrikeYield` node are as follows:

- `Yield`: A Yield quoted in decimal form, e.g. 10% should be entered as 0.1.

  Allowable values: Any real number.

- `Compounding` [Optional]: The compounding or the yield given in `Yield`.

  Allowable values: *Simple, Compounded, Continuous, SimpleThenCompounded*. Defaults to *SimpleThenCompounded* if left blank or omitted.

19.3.31 Barrier Data

This trade component node is used within the trade data containers listed in table 96. Note that not every trade type allows for all barrier styles, the allowable combinations are listed in in table 96.

The barrier data element is specified as in listing 257

Listing 257: Barrier data

```
<BarrierData>
    <Type>UpAndIn</Type>
    <Style>American</Style>
    <Levels>
        <Level>1.2</Level>
    </Levels>
    <Rebate>100000</Rebate>
    <RebateCurrency>USD</RebateCurrency>
    <RebatePayTime>atExpiry</RebatePayTime>
    <OverrideTriggered>true</OverrideTriggered>
</BarrierData>
```

The meanings and allowable values of the elements in the `BarrierData` node follow below.

Trade Data Container	Supported Barrier Styles
FxBarrierOptionData	*American*
FxDigitalBarrierOptionData	*American*
FxEuropeanBarrierOptionData	*European*
FxTouchOptionData	*American*
FxDoubleTouchOptionData	*American*
FxDoubleBarrierOptionData	*American*
FxKIKOBarrierOptionData	*American*
FxTaRFData	*European*
FxAccumulatorData	*European, American*
EquityTaRFData	*European*
EquityAccumulatorData	*European, American*
CommodityAccumulatorData	*European, American*
FxGenericBarrierOption	*American*
EquityGenericBarrierOption	*American*
CommodityGenericBarrierOption	*American*

Table 96: Supported barrier styles per trade data container

- Type: Specifies barrier type. The allowable values are given in Table 97.

Type	**Description**
UpAndOut	The underlying price starts below the barrier level and has to move up for the option to be knocked out.
DownAndOut	The underlying price starts above the barrier level and has to move down for the option to become knocked out.
UpAndIn	The underlying price starts below the barrier level and has to move up for the option to become activated.
DownAndIn	The underlying price starts above the barrier level and has to move down for the option to become activated.
KnockOut	For double level only. The underlying price starts between the barrier levels and has to move up or down for the option to be knocked out.
KnockIn	For double level only. The underlying price starts between the barrier levels and has to move up or down for the option to become activated.
CumulatedProfitCap	For TaRFs only. The instrument terminates once the generated profit reaches the CumulatedProfitCap.
CumulatedProfitCapPoints	For TaRFs only. The instrument terminates once the generated profit divided by fixing amount and absolute value of leverage reaches the CumulatedProfitCapPoints.
FixingCap	For TaRFs only. The instrument terminates once the number of observations where a profit is generated reaches the FixingCap.
FixingFloor	For Accumulators only. The first *n* fixings are guaranteed regardless of whether the trade has been knocked out already.

Table 97: Allowable Type Values.

- Style[Optional]: Specifies the monitoring style of the barrier. Optional, if not given, defaults to the supported barrier style (see table 96 and if both *American* and *European*

barriers are supported, defaults to *American*.
Allowable values: *American, European*.

- Level: The barrier level, defined as the amount in sold (domestic) currency per unit bought (foreign) currency. Double barrier instruments can have two `Level` elements, and these must be in ascending order.
  Allowable values: Any positive real number.
- Rebate[Optional]: The barrier rebate is a fixed amount, expressed in domestic / sold currency paid out to the option holder if a barrier option expires inactive, i.e. it is not knocked in/out. Note that `Rebate` is supported for
  - FxBarrierOptionData
  - FxDigitalBarrierOptionData
  - FxDoubleBarrierOptionData
  - FxEuropeanBarrierOptionData
  - FxGenericBarrierOptionData
  - EquityGenericBarrierOptionData
  - CommodityGenericBarrierOptionData

  only. If defined for several "in" barriers, the amounts must be identical across all barrier definitions (because the rebate amount is paid if none of the "in" barrier is touched and can therefore not depend on the particular barrier). Also, the RebatePayTime must be *atExpiry* for "in" barriers obviously.
  Allowable values: Any positive real number. Defaults to zero if omitted. Cannot be left blank.
- RebateCurrency [Optional]: The currency in which the rebate amount is paid. Defaults to the natural pay currency of the trade. Deviating currencies are supported by the following trade types only:
  - FxGenericBarrierOptionData
  - EquityGenericBarrierOptionData
  - CommodityGenericBarrierOptionData

  Allowable Values: See Table 22 `Currency`.
- RebatePayTime [Optional]: For "in" barriers only atExpiry is allowed. For "out" barriers, both atExpiry and atHit is possible. If not given, defaults to "atExpiry". This field is only supported by the following trade types:
  - FxGenericBarrierOptionData
  - EquityGenericBarrierOptionData
  - CommodityGenericBarrierOptionData

  Allowable Values: *atExpiry*, *atHit*
- StrictComparison [Optional]: Determines how the barrier is checked, as per:

0: the barrier checks use <=, >= to check In-barriers and <, > to check Out-barriers.

1: the barrier checks use strict comparison < and > for both In- and Out-barriers.

2: the barrier checks use strict or equal comparison <= and >= for both In- and Out-barriers.

Note that the StrictComparison element only has an effect for TradeType *GenericBarrierOption*. All other TradeTypes are treated as having StrictComparison set to *0*.

Allowable Values: *0*, *1*, or *2*. Defaults to *0* if omitted.

- OverrideTriggered [Optional]: Specifies whether a barrier was triggered before the valuation date. If given, this overrides the automatic check using fixing data.

  Allowable Values: *true*, *false*

19.3.32 RangeBound

This trade component node is used within the following trade data containers

- FxTaRFfData, EquityTaRFData, CommodityTaRFData
- FxAccumulatorData, EquityAccumulatorData, CommodityAccumulatorData

An example structure of the RangeBound trade component node is shown in Listing 258.

Listing 258: RangeBound

```
<RangeBound>
  <RangeFrom>0</RangeFrom>
  <RangeTo>155.00</RangeTo>
  <Leverage>2</Leverage>
  <Strike>150.54</Strike>
</RangeBound>
```

The meanings and allowable values of the elements in the RangeBound node follow below.

- RangeFrom [Optional]: The lower bound of the range.

  Allowable values: Any real number. If omitted, no lower bound applies. Cannot be left blank.

- RangeTo [Optional]: The upper bound of the range.

  Allowable values: Any real number. If omitted, no lower bound applies. Cannot be left blank.

- Leverage [Optional]: The leverage that applies to the range. For TaRFs, negative leverage can be mixed with positive leverage to reflect a TaRF with switching buyer/seller. However, for Accumulators all given Leverage parameters within the same instrument (in multiple RangeBound nodes) must have the same sign.

  Allowable values: Any real number. Defaults to 1 if omitted. Cannot be left blank.

- Strike [Optional]: The strike specific to the range. If given overwrites a strike given on the trade level.

  Allowable values: Any real number. Defaults to the trade level strike if omitted. Cannot be left blank.

- StrikeAdjustment [Optional]: A strike adjustment relative to the strike given on the trade level. If given the strike for the defined range is computed as $K + A$ where K is the strike on the trade level and A is the strike adjustment. Notice that Strike and StrikeAdjustment can not be given both at the same time.

  Allowable values: Any real number.

19.3.33 Bond Basket Data for Cashflow CDO

This trade component node is used in a Cashflow CDO trade as explained in 19.2.50. An example structure of the `BondBasketData` trade component node is shown in Listing 259.

Listing 259: Bond Basket Data for Cashflow CDO

```
<BondBasketData>
    <Trade id="Bond_1">
      <TradeType>Bond</TradeType>
      <Envelope>
        ...
      </Envelope>
      <BondData>
        ...
      </BondData>
    </Trade>
    <Trade id="Bond_2">
      <TradeType>Bond</TradeType>
      <Envelope>
        ...
      </Envelope>
      <BondData>
        ...
      </BondData>
    </Trade>
</BondBasketData>
```

The usage of the BondBasketData is akin to a portfolio of bond trades, but is embraced by the keyword `BondBasketData` as opposed to `Portfolio`. Compare the vanilla bond section 19.2.40 for usage and allowable values.

19.3.34 CBO Tranches

This trade component node is used in a CBO trade as explained in 19.2.50. An example structure of the `CBOTranches` trade component node is shown in Listing 260.

Listing 260: CBO Tranches

```
<CBOTranches>
  <Tranche>
    <Name>JuniorNote</Name>
    <ICRatio>0.0</ICRatio>
    <OCRatio>0.0</OCRatio>
    <Notional>4000000.00</Notional>
    <FixedLegData>
      <Rates>
        <Rate>0.03</Rate>
      </Rates>
    </FixedLegData>
  </Tranche>
  ...
</CBOTranches>
```

The meanings of the elements of the `CBO tranches` node follow below:

- Tranche: Multiple tranches are allowed and are indicated by the tranche node within the embracing CBOTranches node.
- Name: This string is the name of the tranche, possibly reflecting the position in the capital structure.
- ICRatio: The interest coverage ratio is a number, defined as BasketInterest over TrancheInterest (incl. all senior tranches).
- OCRatio: The overcollateralisation ratio is a number, defined as BasketNotional over TrancheNotional (incl. all senior tranches).
- Notional: The face amount of the tranche.

Depending on the tranche, one can specify a floating or fixed return via the nodes:

- FixedLegData, which is outlined in section 19.3.5.
- FloatingLegData, which is outlined in section 19.3.6.

19.3.35 Formula Based Leg Data

The formula based leg data allows to use complex formulas to describe coupon payoffs. Its `LegType` is `FormulaBased`, and it has the data section `FormulaBasedLegData`. It supports IBOR and CMS based payoffs with quanto and digital features. The following example shows the definition of a coupon paying a capped / floored cross currency EUR-GBP CMS Spread contingent on a USD CMS barrier.

The `Index` field supports operations of the following kind:

- indices like IBOR and CMS indices, and constants as factors, spreads and/or cap/floor values;
- basic operations: $+$, $-$, $\cdot$, $/$;
- operators gtZero() (greater than zero) and geqZero() (greater than or equal zero) yielding 1 if the argument is > 0 (resp. ≥ 0) and zero otherwise

- functions: abs(), exp(), log(), min(), max(), pow()

In listing 261, we present a `FormulaBasedLegData` example.

This leg data type can be used in Swap and Bond trades.

20 Scripted Trade

The scripted trade provides a generic interface for representing exotic products.

The trade's payoff is defined in terms of a *payoff script* using a *scripting langage* described in appendix 20.5. The payoff script can depend on single variables or sets of variables in a single risk class or across several risk classes. The payoff script currently supports EQ, FX (including Precious Metals), Commodity, Inflation and IR variables, i.e. Equity spot prices, FX spot rates, Commodity spot or future prices, IBOR index or CMS index fixings.

20.1 General Structure

A scripted trade comprises

- external data that parametrises the payoff script
- the payoff script

In the following example, a plain vanilla equity call or put option is described by providing the expiry and settlement date, the strike, a put / call indicator and the underlying index as external data in the `Data` node, and the payoff script is referenced by the `ScriptName` node:

```
<Trade id="VanillaOption">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope/>
  <ScriptedTradeData>
    <ScriptName>EuropeanOption</ScriptName>
    <Data>
      <Event>
        <Name>Expiry</Name>
        <Value>2020-02-09</Value>
      </Event>
      <Event>
        <Name>Settlement</Name>
        <Value>2020-02-15</Value>
      </Event>
      <Number>
        <Name>Strike</Name>
        <Value>1200</Value>
      </Number>
      <Number>
        <Name>PutCall</Name>
        <Value>1</Value>
      </Number>
      <Index>
        <Name>Underlying</Name>
        <Value>EQ-RIC:.SPX</Value>
      </Index>
      <Currency>
```

```
        <Name>PayCcy</Name>
        <Value>USD</Value>
      </Currency>
    </Data>
  </ScriptedTradeData>
</Trade>
```

The script itself is defined in a script library, which is part of the static data setup of the application like bond or credit index static data, as follows

```
<ScriptLibrary>
  <Script>
    <Name>EuropeanOption</Name>
    <ProductTag>SingleAssetOption({AssetClass})</ProductTag>
    <Script>
      <Code><![CDATA[
          Option = PAY(max( PutCall * (Underlying(Expiry) - Strike), 0 ),
                                                  Expiry, Settlement, PayCcy);
        ]]></Code>
      <NPV>Option</NPV>
      <Results>
        <Result>ExerciseProbability</Result>
      </Results>
    </Script>
  </Script>
  <Script>
  ...
  </Script>
  ...
</ScriptLibrary>
```

Many scripted trades can thus share the same script, and the payoff scripts can be managed and maintained centrally. Each scipt defines an exotic product type. And adding a product type to the library is a configuration rather than a code release change.

20.2 Data Types

The `Data` node contains definitions for external variables that can be used in the script. These variables can either be scalars or one-dimensional arrays with fixed size. The supported data types are

- Event: a date
- Number: a real number
- Counter: an integer
- Index: an EQ, FX or COMM index
- Currency: a currency
- Daycounter: a day count convention

20.2.1 Event

An event is defined either as a scalar

```
<Event>
  <Name>Expiry</Name>
  <Value>2020-02-09</Value>
</Event>
```

or as an array of dates using VRE's ScheduleData node, e.g.

```
<Event>
  <Name>ExerciseDates</Name>
  <ScheduleData>
    <Rules>
     <StartDate>2016-02-06</StartDate>
     <EndDate>2016-05-06</EndDate>
     <Tenor>1D</Tenor>
     <Calendar>TARGET,US</Calendar>
     <Convention>F</Convention>
     <Rule>Forward</Rule>
   </Rules>
  </ScheduleData>
</Event>
```

using a rule based schedule or

```
<Event>
  <Name>ValuationDates</Name>
  <ScheduleData>
    <Dates>
      <Dates>
        <Date>2018-03-10</Date>
        <Date>2019-03-10</Date>
        <Date>2020-03-10</Date>
        <Date>2021-03-10</Date>
        <Date>2022-03-10</Date>
        <Date>2023-03-10</Date>
        <Date>2024-03-11</Date>
      </Dates>
    </Dates>
  </ScheduleData>
</Event>
```

using a list of dates. An array of dates can also be deduced from another, previously defined array by specifying a shift rule which is useful e.g. to generate fixing or payment schedules from accrual schedules, or notification and settlement schedules from exercise date schedules. Example:

```
<Event>
  <Name>FixingDates</Name>
  <DerivedSchedule>
    <BaseSchedule>AccrualDates</BaseSchedule>
    <Shift>-2D</Shift>
    <Calendar>TARGET</Calendar>
    <Convention>MF</Convention>
  </DerivedSchedule>
```

```
</Event>
```

20.2.2 Number

A scalar number is defined as

```
<Number>
  <Name>Strike</Name>
  <Value>2147.56</Value>
</Number>
```

and likewise an array of numbers as

```
<Number>
  <Name>EquityNotionalAmount</Name>
  <Values>
    <Value>100000000</Value>
    <Value>90000000</Value>
    <Value>80000000</Value>
  </Values>
</Number>
```

20.2.3 Index

Indices are defined as

```
<Index>
  <Name>Underlying</Name>
  <Value>EQ-RIC:.SPX</Value>
</Index>
```

in case of a scalar and

```
<Index>
  <Name>Underlyings</Name>
  <Values>
    <Value>EQ-UND1</Value>
    <Value>EQ-UND2</Value>
    <Value>EQ-UND3</Value>
    <Value>EQ-UND4</Value>
    <Value>EQ-UND5</Value>
  </Values>
</Index>
```

in case of a vector. Currently, Equity, FX, Commodity, Interest Rate and Inflation indices as well as generic indices are supported, the naming convention follows the standard VRE conventions, i.e.

- Equity: These are declared based on identifier type
  - `EQ-ISIN:{Name}:{Currency}:{Exchange}` for ISIN equities, e.g. `EQ-ISIN:NL0000852580:EUR:XAMS`,

  - EQ-RIC:{Name} for RIC equities, e.g. EQ-RIC:.SPX,
  - EQ-FIGI:{Name} for FIGI equities, e.g. EQ-FIGI:BBG000BLNNV0,
  - EQ-BBG:{Name} for BBG equities, e.g. EQ-BBG:BARC LN Equity.
- FX: FX-SOURCE-CCY1-CCY2 with foreign currency CCY1 and domestic currency CCY2, as in Table 28. The order of the currencies here is important as they will have different meanings. For a natural payoff, CCY2 must correspond to the instrument's payment currency. Otherwise, we have a quanto payoff. See section 20.4 for more information.
- Commodity: These are declared as COMM-{Name}, with possible extensions as follows (see below also for more information on the meanings of the extensions to the commodity indices):
  - COMM-NYMEX:CL (spot),
  - COMM-NYMEX:CL-2020-09 (future with expiry 01 Sep 2020),
  - COMM-NYMEX:CL-2020-09-15 (future with expiry 15 Sep 2020).
- Interest Rate: These are declared as CCY-INDEX-TENOR, e.g. EUR-EURIBOR-6M, EUR-CMS-10Y. See Table 26 for a list of allowable IR indices.
- Inflation: These are declared as {Name}, with possible extensions as follows(see below also for more information on the meanings of the extensions to the inflation indices):
  - EUHICPXT (constant/non-interpolated),
  - EUHICPXT#F (flat interpolation),
  - EUHICPXT#L (linear interpolation).
- Generic: These are declared as GENERIC-{Name}.

Notice that generic indices only provide historical fixings.

For additional information on the equity name conventions see 30 and 19.3.29. In general the equity name can be represented using the string representation IdentifierType:Name:Currency:Exchange for the underlying.
For additional information on the fx index format see 28.
For additional information on the commodity names see 32.

Within the scripting framework there are additional ways to reference commodity indices in a more dynamical fashion: When evaluating a commodity index using the index evaluation operator (see 20.5) as in

```
CommodityUnderlying(ObservationDate)
```

for

- a variable of the form COMM-{Name}#N, the index will be resolved to the $N+1$th future with expiry greater than ObservationDate for the given commodity underlying, $N \geq 0$. The parameter N corresponds to the field FutureMonthOffset commonly used in commodity trade xml schemas

- the above form can take one or two additional parameters `COMM-{Name}#N#D` or `COMM-{Name}#N#D#{Cal}` where *D* corresponds to `DeliveryRollDays` and `Cal` is the calendar used to roll the observation date forward before the next future expiry is looked up. If `Cal` is not given, it defaults to the null calendar (no holidays).
- a variable of the form `COMM-{Name}!N`, the index will be resolved to the *N*th future relative to the month and year of the ObservationDate.

In general, if a commodity future is referenced, the observation date should be be less or equal to the expiry date of the future, since no historical fixing will in general be available after the future expiry date. In the future-looking simulation the observation of an expired future is possible, in this case the (model) value of the future is kept constant at its value at the expiry date for observation dates after the expiry date.

When evaluating an inflation index as in

```
CPIIndex(FixingDate)
```

the variable `CPIIndex` can be given in the following ways:

- a variable of the form `EUHICPXT` will evaluate the standard VRE inflation index, which is non-interpolated. This means the result will be constant for all FixingDates in an inflation period (i.e. usually constant for a calendar month)
- a variable of the form `EUHICPXT#F`, i.e. an VRE inflation index name followed by `#F` indicating a flat interpolation of the index, this form is equivalent to the previous form making the interpolation mode explicit as "flat"
- a variable of the form `EUHICPXT#L` where the suffix `#L` indicates linear interpolation of the index, i.e. the result will be the interpolated fixing between the usually monthly inflation index fixings

The use of the interpolation extensions is deprecated. Interpolation of fixings should be handled in the payoff script. The CPI indices are forecasting a flat fixing for given the inflation period. The use of interpolation is still supported but will be eventually removed in a later release.

20.2.4 Currency

A scalar currency variable is defined as

```
<Currency>
  <Name>PayCcy</Name>
  <Value>USD</Value>
</Currency>
```

and an array of currencies as

```
<Currency>
  <Name>BasketCurrencies</Name>
  <Values>
    <Value>USD</Value>
    <Value>GBP</Value>
```

```
      <Value>JPY</Value>
    </Value>
  </Currency>
```

20.2.5 Daycounter

A scalar day count convention variable is defined as

```
  <Daycounter>
    <Name>AccrualDayCounter</Name>
    <Value>Actual/360</Value>
  </Daycounter>
```

and an array of day counters as

```
  <Daycounter>
    <Name>LegAccrualDayCounters</Name>
    <Values>
      <Value>30/360</Value>
      <Value>Actual/360</Value>
    </Value>
  </Currency>
```

20.3 Compact Trade XML

In addition to the trade xml described in 20.1, we support an alternative, more compact, format where the variable names are derived from the node names and the type is given by an attribute. The script is referenced via a name derived from the root node of the trade data. Consider the following example of a one touch option in the original format:

```
<Trade id="SCRIPTED_FX_ONE-TOUCH_OPTION">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <ScriptedTradeData>
    <ScriptName>OneTouchOption</ScriptName>
    <Data>
      <Event>
        <Name>Settlement</Name>
        <Value>2020-08-01</Value>
      </Event>
      <Event>
        <Name>ObservationDates</Name>
        <ScheduleData>
          <Rules>
            <StartDate>2018-12-28</StartDate>
            <EndDate>2020-08-01</EndDate>
            <Tenor>1D</Tenor>
            <Calendar>US</Calendar>
            <Convention>U</Convention>
```

```
            <TermConvention>U</TermConvention>
            <Rule>Forward</Rule>
          </Rules>
        </ScheduleData>
        <ApplyCoarsening>true</ApplyCoarsening>
      </Event>
      <Number>
        <Name>BarrierLevel</Name>
        <Value>0.009</Value>
      </Number>
      <Number>
        <Name>Type</Name>
        <Value>-1</Value>
      </Number>
      <Number>
        <Name>LongShort</Name>
        <Value>1</Value>
      </Number>
      <Number>
        <Name>Amount</Name>
        <Value>10000000</Value>
      </Number>
      <Currency>
        <Name>PayCcy</Name>
        <Value>USD</Value>
      </Currency>
      <Index>
        <Name>Underlying</Name>
        <Value>FX-TR20H-USD-JPY</Value>
      </Index>
    </Data>
  </ScriptedTradeData>
</Trade>
```

In the compact format the same trade looks like this:

```
<Trade id="SCRIPTED_FX_ONE-TOUCH_OPTION">
  <TradeType>ScriptedTrade</TradeType>
  <Envelope>
    <CounterParty>CPTY_A</CounterParty>
    <NettingSetId>CPTY_A</NettingSetId>
    <AdditionalFields/>
  </Envelope>
  <OneTouchOptionData>
    <Settlement type="event">2020-08-01</Settlement>
    <ObservationDates type="event">
      <ScheduleData>
        <Rules>
          <StartDate>2018-12-28</StartDate>
          <EndDate>2020-08-01</EndDate>
          <Tenor>1D</Tenor>
          <Calendar>US</Calendar>
          <Convention>U</Convention>
          <TermConvention>U</TermConvention>
          <Rule>Forward</Rule>
        </Rules>
      </ScheduleData>
```

```
      <ApplyCoarsening>true</ApplyCoarsening>
    </ObservationDates>
    <BarrierLevel type="number">0.009</BarrierLevel>
    <BarrierType type="barrierType">DownIn</BarrierType>
    <LongShort type="longShort">Long</LongShort>
    <Amount type="number">10000000</Amount>
    <PayCcy type="currency">USD</PayCcy>
    <Underlying type="index">FX-TR20H-USD-JPY</Underlying>
  </OneTouchOptionData>
</Trade>
```

The supported types that must be specified in the `type` attribute are `number`, `event`, `currency`, `dayCounter` and `index`. In addition we support some custom types that are mapped to numbers internally and allow for a more natural representation of the trade:

- `bool` with a mapping `true` $\mapsto$ 1, `false` $\mapsto$ -1
- `optionType` with a mapping `Call` $\mapsto$ 1, `Put` $\mapsto$ -1, `Cap` $\mapsto$ 1, `Floor` $\mapsto$ -1
- `longShort` with a mapping `Long` $\mapsto$ 1, `Short` $\mapsto$ -1
- `barrierType` with a mapping `DownIn` $\mapsto$ 1, `UpIn` $\mapsto$ 2, `DownOut` $\mapsto$ 3, `UpOut` $\mapsto$ 4

Arrays of events are specified as in the example above (ObservationDates), for the other types the values are listed in value tags, e.g. an array of numbers is declared as

```
  <MyNumberArray type="number">
    <Value>100.0</Value>
    <Value>200.0</Value>
    <Value>200.0</Value>
  </MyNumberArray>
```

20.4 A comment on the Payment Currency in Scripted Trades

The payoffs described in the Trade Specific Data for scripted trades usually involve a payment currency. Unless stated otherwise, this currency represents the *natural* payoff currency, i.e.

- for equity and commodity underlyings this should be the currency in which the underlying price is quoted
- for FX underlyings `FX-SOURCE-CCY1-CCY2` this should be the domestic (target, numeraire) currency `CCY2`.

If the payment currency is set to a different currency on the other hand, the resulting payoff is a true quanto payoff, i.e. the amount of the payoff is determined on the natural currency, but paid in a different currency *without* converting the amount to this latter currency using the fair FX Spot rate on the settlement date.

We are aware that a conversion from the natural payoff currency to a different settlement currency using the fair FX Spot rate is sometimes part of the terms and conditions of a trade. This conversion has no or small impact on the valuation and risk profile of a trade though and is therefore usually not part of the payoff modeling.

As an example consider a Forward Volatility Agreement on the FX pair GBP-EUR. The `PayCcy` should be set to EUR in this case, even if the forward premium is settled in GBP as it

is usual market practice for this pair.

Notice that the above does not apply to fixed premiums, where the premium is given as a fixed number in the trade xml together with a premium currency in which this amount should be paid.

20.5 Payoff Scripting Language

Whitespace

Whitespace (space, tab, return, newline) is ignored during the parsing. All variable identifiers and keywords are case sensitive.

Keywords

The language uses keywords and predefined function names as listed in table 98 which may may not be used as variable identifiers.

In addition the following variable identifiers are automatically populated with special values when running the script engine on a trade:

- `TODAY`: the current evaluation date

Variables

Variables that can be used in the script are either

- externally defined variables, defined in the data node of the trade xml representation
- variables local to the script, declared within the script

Externally defined variables are protected from being modified by the script. All variables used within the script must be either externally defined or declared at the top of the script using

```
NUMBER continuationValue, exerciseValue, x[10];
```

which declares two scalars `continuationValue` and `exerciseValue` and an array `x` of size 10 (see 20.5 for more details on arrays). The only exemption to this rule is the variable declared in the `NPV` node of the script, which is definied implicitly as a scalar number.

Notice that within the script only variables of type Number can be declared.[7] All variables are initialised with 0. The scope of a variable declaration is always global to the script, multiple declarations of the same variable name are forbidden.

Variable identifiers are subject to the following restrictions

- must start with a character, then characters or numbers or underscores may follow
- no other special characters, no keywords or predefined functions allowed (see 20.5)
- e.g. `x` or `x_23`, `aValue` are valid identifiers

[7] this restriction allows the static analysis of the script

Keyword	Context
IF	Control Flow
THEN	
ELSE	
END	
FOR	
IN	
DO	
NUMBER	Type Identifiers
OR	Logical Operators
AND	
abs	Functions
exp	
ln	
sqrt	
normalCdf	
normalPdf	
max	
min	
pow	
black	
dcf	
days	
PAY	Model dependent functions
LOGPAY	
NPV	
NPVMEM	
HISTFIXING	
DISCOUNT	
FWDCOMP	
FWDAVG	
ABOVEPROB	
BELOWPORB	
SIZE	Other Statements
DATEINDEX	
REQUIRE	
SORT	
PERMUTE	

Table 98: Reserved keywords.

- identifiers starting with an underscore are technically allowed as well, but reserved for special use cases (e.g `_AMC_SimDates` and `_AMC_NPV` for AMC exposure generation)

Arrays, SIZE operator

Arrays are declared by specifying the size of the array in sqaure brackets, e.g.

```
NUMBER x[10], y[SIZE(ObservationDates)], z[5+3*v];
```

declares arrays

- `x` of size 10
- `y` with the same size as the array `ObservationDates`
- `z` with size $5 + 3v$ where v is a number variable

Once an array is delcared its size can not be changed. The ith element of an array a is accessed by `a[i]`, where i is an expression evaluating to a number. Here $i = 1, 2, 3, \ldots, n$, where n is the fixed size of the array, i.e. the subscripts start at 1 (as opposed to 0 as in some other languages).

The size of an array `a` can be evaluated by `SIZE(a)`. Only one dimensional arrays are supported. The array subscript must be *deterministic*, e.g.

```
IF Underlying > Strike THEN
  i = 1;
ELSE
  i = 2;
END;
Payoff = y[i];
```

is illegal since i in general will be path-dependent, but

```
IF Underlying > Strike THEN
  Payoff = y[1];
ELSE
  Payoff = y[2];
END;
```

is valid.[8]

Sorting Arrays: SORT and PERMUTE instructions

Given an array `x` of number type the statement

```
SORT (x);
```

will sort the array (pathwise) in ascending order. The statement

[8] the background is the simplicity and performance of the engine implementation

```
SORT (x, y);
```

will write a sorted version of `x` to `y` and leave `x` unchanged. The array `y` must be of number type and have the same size as `x`. The array `y` can also be equal to `x`, the statement `SORT(x,x);` is equivalent to `SORT(x);`. Finally the statement

```
SORT (x, y, p);
```

will write a sorted version of `x` to `y` and populate another array `p` with indices $1, \ldots,$ `SIZE`(x) such that $x[p[1]], \ldots, x[p[n]]$ is sorted. Here `p` must be an array with the same size as `x` and of number type.

A permutation `p` generated as above (or set up otherwise) can be used to sort an unrelated array `z` using

```
PERMUTE (z, p);
```

which will reorder the values of `z` as $z[1] \to z[p[1]]$, $z[2] \to z[p[2]]$ ... etc. The statement

```
PERMUTE (z, w, p);
```

will do the same, but write the result to `w` and leave `z` untouched.

Function `DATEINDEX`

Given an array `a` and a single date `d`, the expression

```
DATEINDEX(d, a, EQ)
```

returns 0 if the date d is not found in the array a and otherwise the (first) index i for which `a[i]` equals `d`. The variable d is required to be of type event. The variable a is only required to be an array, if the type of its elements are not event, the return value will always be zero indicating that d was not found in a. Similarly,

```
DATEINDEX(d, a, GEQ)
```

returns the index of the earliest date in a that is greater or equal than d, and

```
DATEINDEX(d, a, GT)
```

returns the index of the earliest date in a that is greater than d. If no such dates exists for `GEQ` or `GT`, the size of a plus 1 will be returned.

Instructions

A typical script comprises a sequence of instructions, each one terminated by `;`.

Index evaluation

Given an variable `index` of type Index its historical or projected fixing at a date d is evaluated using the expression `index(d)`. This is applicable to all index types. For example

```
Underlying(ObservationDate)
```

evaluates the index assigned to the variable Underlying at the date assigned to the variable ObservationDate. For FX, EQ, IR and COMM Spot indices this corresponds to a a fixing at the observation date in the usual sense. For COMM Future indices it is the observed future price at the observation date.

For INF indices the argument is the actual fixing date, which due to availability lags is observed at a later simulation time in models with dynamical inflation simulation. For example in the GaussianCam model, this lag is defined as the number of calendar days from the zero inflation term structure base date to its reference date (adjusted to the first date of the inflation period to be consistent with the same adjustment applied to the base date). This means that when observing an inflation index at a fixing date d, the model state at d + lag is used to make this observation.

The extended syntax

```
Underlying(ObservationDate, ForwardDate)
```

evaluates the projected fixing for ForwardDate as seen from ObservationDate.

This is applicable to FX, EQ, IR, INF and COMM Spot indices, but not to COMM Future indices, since for the latter the two concepts coincide (for ForwardDate < FutureExpiry). If a forward date is given for the observation of a COMM future index, no error is thrown, but it will be ignored.

For inflation indices, the ForwardDate will be the actual fixing date again and the ObservationDate will be using a lagged state as explained above.

The ForwardDate must be greater or equal than the ObservationDate. If the ForwardDate is strictly greater than the ObservationDate the ObservationDate must not be a past date (for inflation indices it must not lie before the inflation term structure's base date), since the computation of projected fixings for past dates would involve the knowledge of past curves, i.e. past market data.

Notice also the further specifics of commodity and inflation indices in 20.2.3.

Comparisons == and !=

Compares two values, e.g. `x==y` or `x!=y`. This is applicable to all types. For a number the interpretation is "numerically equal".

Comparisons <, <=, >, >=

Compares two values `x<y`, `x<=y`, `x>y`, `x>=y`. Applicable to numbers and events, but not to currencies or indices. For numbers the interpretation is "less than, but not numerically equal", "less than or numerically equal", etc.

Operations +, -, *, /

Arithmetic operations `x+y`, `x-y`, `x*y`, `x/y`, applicable to numbers only.

Assignment =

Assignment `x = y`, only allowed for numbers within the script.

Logical Operators AND, OR, NOT

Connects results of comparisons or other logical expressions:

- `x<y AND z!=0`
- `x<y OR z!=0`
- `NOT(x==y)`
- AND has higher precedence than OR, e.g.
- `x<y AND y<z OR z!=0` same as `{x<y AND y<z} OR z!=0`, but
- `x<y AND {y<z OR z!=0}` requires parenthesis
- better always use parenthesis when mixing `AND` / `OR`

Conditionals: IF ... THEN ... ELSE ...

Conditional execution can be written as

```
IF condition THEN
    ... if-body ...
ELSE
    ... else-body ...
END
```

Examples:

```
IF x == y THEN
  z = PAY(X,d,p,ccy);
  w = 1;
END;
```

```
IF x == y THEN
  z = PAY(X,d,p,ccy);
ELSE
  z = 0;
  w = 0;
END;
```

where the `ELSE` part is optional. The body can comprise one vre more instructions, each of which must be terminated by `;`.

Loops: FOR ... IN ... DO

Loops are written as

```
FOR i IN (a,b,s) DO
... body ...
END
```

where `i` is a number variable identifier, and `a`, `b`, `s` are expressions that yield a result of type Number. The variable `i` must have been declared in the script before it can be used as a loop variable. The code in the body is executed for the values $i = a, a + s, \ldots$ until $a + ks > b$ if $s > 0$ or $a + ks < b$ if $s < 0$ for some integer $k > 0$. All values a, b, s must be integers and $s \neq 0$.

Example:

```
NUMBER i,x;
FOR i IN (1,100.1) DO x = x + i; END;
```

Here `a, b` must be deterministic, `i` must not be modified in the loop body. If a or b are modified in the loop body, still the initial values read at the start of the loop are used. The loop body can comprise one or more instructions, each of which must be terminated by `;`.

Special variable: TODAY

A constant event variable, set to the current evaluation date. This can e.g. be used to restrict exercise decisions to future dates, see 20.5 for an example.

Checks: REQUIRE

If the condition `C` is not true, a runtime error is thrown. Examples:

- `REQUIRE SIZE(ExerciseDates) == SIZE(SettlementDates);`
- `REQUIRE SIZE(Underlyings) == 2;`
- `REQUIRE Strike >= 0;`

Functions `min`, `max`, `pow`

Binary functions `min(x,y)`, `max(x,y)`, `pow(x,y)`, applicable to numbers only.

Functions `-`, abs, exp, ln, sqrt

Unary functions `-x`, `abs(x)`, `exp(x)`, `ln(x)`, `sqrt(x)`, applicable to numbers only.

Functions `normalPdf`, `normalCdf`

Returns the standard normal pdf $\phi(x)$ resp. cdf $\Phi(x)$, applicable to numbers only.

Function `black`

Implements the black formula `black(omega, obs, expiry, k, f, sigma)` with

$$\begin{aligned} \text{black} &= \omega \cdot (f\Phi(\omega d_1) - k\Phi(\omega d_2)) \\ d_{1,2} &= \frac{\ln(f/k) \pm \frac{1}{2}\sigma^2 t}{\sigma\sqrt{t}} \end{aligned}$$

where t is the (model's) year fraction between obs and expiry date, i.e.:

- `omega` is 1 (call) or -1 (put)
- `obs, expiry` are the observation / expiry dates
- `k, f` are the strike and the forward
- `sigma` is the implied volatility
- notice that no discounting is applied

Function `dcf`

The expression `dcf(dc, d1, d2)` returns the day count fraction for a day count convention `dc` and a period defined by dates `d1` and `d2`.

Function `days`

The expression `days(dc, d1, d2)` returns the number of days between `d1` and `d2` for a day count convention `dc`.

Function `PAY`

The expression `PAY(X, d, p, ccy)` calculates a discounted payoff for an amount X observed at a date d, paid at a date p in currency ccy, i.e.

$$\frac{XP_{ccy}(d,p)FX_{\text{ccy,base}}(d)}{N(d)} \tag{1}$$

where

- here P_{ccy} is the discount factor in currency ccy, FX is the FX spot from ccy to base and N is the model numeraire
- $d \leq p$ must hold
- if p lies on or before the evaluation date, the result is zero; X is not evaluated in this case. Note that X is evaluated in the LOGPAY function if past cashflows are included, see 20.5.
- avoids reading non-relevant past fixings from the index history
- if d lies before (but p after) the evaluation date, it is set to the evaluation date, i.e. the result is computed as of the evaluation date

Function LOGPAY

The expression `LOGPAY(X, d, p, ccy)` has the same meaning as `PAY(X, d, p, ccy)` (see 20.5) but as a side effect populates an internal cashflow log that is used to generate expected flows. The generated flow is

$$\frac{N(0)E\left(\frac{XP_{ccy}(d,p)FX_{\text{ccy,base}}(d)}{N(d)}\right)}{FX_{\text{ccy,base}}(0)P_{ccy}(0,p)} \tag{2}$$

which ensures that the flows discounted on T0 curves and converted with T0 FX Spots reproduce the NPV generated from `LOGPAY` expressions.

There is a second form `LOGPAY(X, d, p, ccy, legNo, type)` taking in addition

- a leg number `legNo`, which must evaluate to a determinisitc number
- a cashflow type `type`, which is an arbitrary string meeting the conventions for variable names

This additional information is used to populate the VRE cashflow report. If not given, `legNo` is set to 0 and type is set to `Unspecified`. Notice that cashflows will equal pay dates, pay currencies, leg numbers and types are aggregated to one number in the cashflow report.

A third form `LOGPAY(X, d, p, ccy, legNo, type, slot)` takes an additional parameter `slot` which must evaluate to a whole positive and deterministic number $1, 2, 3, \ldots$. If several cashflows are logged into the same slot, previous results are overwritten. This is useful for scripts where tentative cashflows are generated that are later on superseded by other cashflows (e.g. for an American option).

Examples for the three forms are given below:

```
Payoff1 = LOGPAY( Notional * fixedRate, PayDate, PayDate, PayCcy);
Payoff2 = LOGPAY( Notional * fixedRate, PayDate, PayDate, PayCcy, 2, Interest);
Payoff3 = LOGPAY( Notional * fixedRate, PayDate, PayDate, PayCcy, 2, Interest, 3);
```

Here, Payoff1 will appear under leg number 0 and flow type “Unspecified” in the cashflow report. Payoff2 will appear under leg number 2 and flow Type “Interest”. The same holds for Payoff3, but if any amounts were booked using the slot parameter 3 previously they will be overwritten with the current amount.

Note: If IncludePastCashflows in the pricing engine config is set to true then even if p lies on or before the evaluation date, a cashflow entry will be generated.

Function NPV, NPVMEM

The expression `NPV(X, d, [C], [R1], [R2])` calculates a conditional NPV of an amount X conditional on a date d, i.e.

$$E(X\,|\mathcal{F}_d \cap \mathcal{F}_C) \tag{3}$$

where $\mathcal{F}_d$ is the sigma algebra representing the information generated by the model up to d and $\mathcal{F}_C$ represents the additional condition C (if given). In an MC model 3 is computed using

a regression against the model state at d. C can be used to filter the training paths, e.g. on ITM paths only. d must not lie before the evaluation date, but for convenience the scipt engines will treat d as if it were equal to the evaluation date in this case for the purpose of the NPV function evaluation.

The regressor can be enriched by (at most 2) additional variables R_i. A typical usage is the accumulated coupon in a target redemption feature which heavily influences the future conditional NPV but is not captured in the model state.

A typical usage of the NPV function is to decide on early exercises in the Longstaff-Schwartz algorithm:

```
NUMBER Payoff, d;
FOR d IN (SIZE(ExerciseDates), 1) DO
    IF ExerciseDates[d] > TODAY THEN
      Payoff = PAY( PutCall * (Underlying(ExerciseDates[d]) - Strike),
                    ExerciseDates[d], ExerciseDates[d], PayCcy);
      IF Payoff > 0 AND Payoff > NPV( Option, ExerciseDates[d], Payoff > 0) THEN
        Option = Payoff;
      END;
    END;
END;
Option = LongShort * Quantity * Option;
```

Here `TODAY` represents the evaluation date to ensure that only future exercise dates are evaluated, see 20.5.

Note: It is the users responsibility to use NPV() correctly to a certain extend: An example would be that X is composed from both past and future fixings w.r.t. the observation time t. In that case only the future fixings should be included in the argument of NPV(), whereas the past fixings are known and should just be added to the result of NPV().

The variant `NPVMEM(X, d, s, [C], [R1], [R2])` works exactly like `NPV(X, d, [C], [R1], [R2])` except that it takes an additional parameter `s` that must be an integer. If `NPVMEM()` is called more than once for the same parameter `s` a regression model representing the conditional npv will only be trained once and after that the trained model will be reused. The usual use case is for scripts used in combination with the AMC module where a regression model will be trained on a relative large number of paths (specified in the pricing engine configuration) and then reused in the global exposure simulation on a relatively small number of paths (specified in the xva simulation setup).

Function `HISTFIXING`

The expression `HISTFIXING(Underlying, d)` returns 1 if d lies on or before the reference date *and* the underlying has a historical fixing as of the date d and 0 otherwise.

Function `DISCOUNT`

The expression `DISCOUNT(d, p, ccy)` calculates a discount factor $P_{ccy}(d, p)$ as of d for p in currency ccy. Here d must not be a past date and $d \leq p$ must hold.

Functions `FWDCOMP` and `FWDAVG`

The `FWDCOMP()` and `FWDAVG()` functions are used to calculate a daily compounded or averaged rate over a certain period based on an overnight index such as USD-SOFR, GBP-SONIA, EUR-ESTER etc..

The rate is estimated as seen from an observation date looking *forward* from that date, even if fixings relevant for the rate lie in the past w.r.t. the observation date. In the latter case, an approximation to the true rate which is then dependent on the path leading from TODAY to the current model state at the observation date is calculated. This approximation is model-dependent. The only exception to this mechanics are historical fixings that are *known* as of TODAY. Such fixings are always taken into consideration with their true value.

More specifically, the `FWDCOMP()` and `FWDAVG()` functions take the following parameters. The parameters must be given in that order, and all parameters must be given in sequence up to the parameter "end" (last mandatory parameter) or the end of an optional parameter group (i.e. an optional parameter group must be given as a whole). Furthermore, all parameters must be deterministic.

- index [mandatory]: an overnight index `index`, e.g. EUR-EONIA, USD-SOFR, ...
- obs [mandatory]: an observation date obs $\leq$ start; if obs is $<$ TODAY it is set to TODAY, i.e. the result is as of TODAY in this case
- start [mandatory]: the value start date, this might be modified by a non-zero lookback
- end [mandatory]: the value end date, this might be modified by a non-zero lookback
- spread [optional group 1]: a spread, defaults to 0 if not given
- gearing [optional group 1]: a gearing, defaults to 1 if not given
- lookback [optional group 2]: a lookback period given as number of days, defaults to 0 if not given. This argument must be given as either a constant number or a plain variable, i.e. not as a more complex expression than either of these.
- rateCutoff [optional group 2]: a rate cutoff given as number of days, defaults to 0 if not given
- fixingDays [optional group 2]: the fixing lag given as number of days, defaults to 0 if not given. This argument must be given as either a constant number or a plain variable, i.e. not as a more complex expression than either of these.
- includeSpread [optional group 2]: a flag indicating whether to include the spread in the compounding, a value equal to 1 indicates 'true', -1 false, defaults to 'false' if not given
- cap [optional group 3]: a cap value, defaults to 999999 (no cap) if not given
- floor [optional group 3]: a floor value, defaults to -999999 (no floor) if not given
- nakedOption [optional group 3]: a flag indicating whether the embedded cap / floor should be estimated, a value equal to -1 indicates 'false' (capped / floored coupon rate is estimated), 1 'true' (embedded cap / floor rate is estimated), defaults to 'false' if not given
- localCapFloor [optional group3]: a flag indicating whether the cap / floor is local, a value equal to -1 indicates 'false', 1 'true', defaults to 'false' if not given.

Based on these parameters a rate corresponding to that computed for a vanilla floating leg is estimated, see the description in section "Floating Leg Data, Spreads, Gearings, Caps and Floors" for more details on this.

Functions ABOVEPROB, BELOWPROB

These functions are only available in Monte Carlo engines. The expression

```
ABOVEPROB(underlying, d1, d2, U)
```

returns the pathwise probability that the value of an index `underlying` lies at or above a number U for at least one time t between dates $d1$ and $d2$ conditional on the underlying taking the simulated path values at $d1$ and $d2$. The probability is by definition computed assuming a continuous monitoring. Similarly,

```
BELOWPROB(underlying, d1, d2, D)
```

returns the probability that the value of the underlying lies at or below D. Notice that $d1$ and $d2$ should be adjacent simulation dates to ensure that the results computed in the script are meaningful. This means the script should not evaluate the underlying at a date d with $d1 < d < d2$.

We note that U and D are not required to be deterministic quantities, although the common use case will probably be to have path-independent inputs.

Finally, if $d1 > d2$ both functions return 0.

Listing 160: Payoff script for a Worst Of Basket Swap.

```
REQUIRE SIZE(Underlyings) == SIZE(InitialPrices);
REQUIRE SIZE(SettlementDates) == SIZE(DeterminationDates);
REQUIRE SIZE(KnockOutLevels) == SIZE(DeterminationDates) - 1;
REQUIRE SIZE(CouponTriggerLevels) == SIZE(DeterminationDates) - 1;

NUMBER alive, couponAccumulation, numOfKnockedAssets, fixing, n, accrualFraction, indexInitial;
NUMBER numOfTriggeredAssets, indexFinal, performance, worstPerformance, d, payoff, u;

Option = Option + LOGPAY(LongShort * Quantity * InitialFixedRate,
                         SettlementDates[1], SettlementDates[1], PayCcy, 0, InitialFixedAmount);

alive = 1;
couponAccumulation = 1;
n = SIZE(DeterminationDates);

FOR d IN (2, n, 1) DO
  fixing = FloatingIndex(FixingSchedule[d-1]) + FloatingSpread;
  accrualFraction = dcf(FloatingDayCountFraction, DeterminationDates[d-1], DeterminationDates[d]);
  Option = Option - LOGPAY(LongShort * Quantity * alive * fixing * accrualFraction,
                           FixingSchedule[d-1], SettlementDates[d], PayCcy, 1, FloatingLeg);

  numOfTriggeredAssets = 0;
  FOR u IN (1, SIZE(Underlyings), 1) DO
    IF Underlyings[u](DeterminationDates[d]) >= CouponTriggerLevels[d-1] * InitialPrices[u] THEN
      numOfTriggeredAssets = numOfTriggeredAssets + 1;
    END;
  END;
  IF numOfTriggeredAssets == SIZE(Underlyings) THEN
    Option = Option + LOGPAY(LongShort * Quantity * alive * CouponRate * couponAccumulation,
                             SettlementDates[d], SettlementDates[d], PayCcy, 2, FixedCouponLeg);
    couponAccumulation = 1;
  ELSE
    IF AccumulatingCoupons == 1 THEN
      couponAccumulation = couponAccumulation + 1;
    END;
  END;

  IF d == n THEN
    FOR u IN (1, SIZE(Underlyings), 1) DO
      indexInitial = InitialPrices[u];
      indexFinal = Underlyings[u](DeterminationDates[n]);
      performance = indexFinal / indexInitial;

      IF {u == 1} OR {performance < worstPerformance} THEN
        worstPerformance = performance;
      END;
    END;

    IF worstPerformance < min(Strike, KnockInLevel) THEN
      payoff = worstPerformance - Strike;
      Option = Option + LOGPAY(LongShort * Quantity * alive * payoff, DeterminationDates[n],
                               SettlementDates[n], PayCcy, 3, EquityAmountPayoff);
    END;
  END;

  IF d != n THEN
    numOfKnockedAssets = 0;
    FOR u IN (1, SIZE(Underlyings), 1) DO
      IF Underlyings[u](DeterminationDates[d]) >= KnockOutLevels[d-1] * InitialPrices[u] THEN
        numOfKnockedAssets = numOfKnockedAssets + 1;
      END;
    END;
    IF numOfKnockedAssets == SIZE(Underlyings) THEN
      alive = 0;
    END;
  END;
END;
```

Listing 167: Payoff script for a TaRF.

```
REQUIRE FixingAmount > 0;
REQUIRE LongShort == 1 OR LongShort == -1;
REQUIRE SIZE(RangeUpperBounds) == SIZE(RangeLowerBounds);
REQUIRE SIZE(RangeLowerBounds) == SIZE(RangeLeverages);
REQUIRE SIZE(RangeLowerBounds) == SIZE(RangeStrikes);
REQUIRE TargetType == -1 OR TargetType == 0 OR TargetType == 1;
REQUIRE SIZE(FixingDates) == SIZE(SettlementDates);

NUMBER Payoff, d, r, PnL, wasTriggered, AccProfit, Hits, currentNotional;
NUMBER Fixing[SIZE(FixingDates)], Triggered[SIZE(FixingDates)];

FOR r IN (1, SIZE(RangeUpperBounds), 1) DO
  REQUIRE RangeLowerBounds[r] <= RangeUpperBounds[r];
  REQUIRE RangeStrikes[r] >= 0;
END;

FOR d IN (1, SIZE(FixingDates), 1) DO
  Fixing[d] = Underlying(FixingDates[d]);
  IF wasTriggered != 1 THEN
    PnL = 0;
    FOR r IN (1, SIZE(RangeUpperBounds), 1) DO
      IF Fixing[d] > RangeLowerBounds[r] AND Fixing[d] <= RangeUpperBounds[r] THEN
        PnL = PnL + RangeLeverages[r] * FixingAmount * (Fixing[d] - RangeStrikes[r]);
      END;
    END;

    IF PnL >= 0 THEN
      AccProfit = AccProfit + PnL;
      Hits = Hits + 1;
    END;

    IF {KnockOutProfitEvents > 0 AND Hits >= KnockOutProfitEvents} OR
       {KnockOutProfitAmount > 0 AND AccProfit >= KnockOutProfitAmount} THEN
      wasTriggered = 1;
      Triggered[d] = 1;
      IF TargetType == 0 THEN
        Payoff = Payoff + LOGPAY(TargetAmount - (AccProfit - PnL), FixingDates[d], SettlementDates[d
      END;
      IF TargetType == 1 THEN
        Payoff = Payoff + LOGPAY(PnL, FixingDates[d], SettlementDates[d], PayCcy, 0, Cashflow);
      END;
    ELSE
        Payoff = Payoff + LOGPAY(PnL, FixingDates[d], SettlementDates[d], PayCcy, 0, Cashflow);
    END;
  END;
END;
value = LongShort * Payoff;
currentNotional = FixingAmount * RangeStrikes[1];
```

Listing 168: Knock Out Swap

```
<Trade id="194837232">
  <TradeType>KnockOutSwap</TradeType>
  <Envelope>...</Envelope>
  <KnockOutSwapData>
    <!-- BarrierData and BarrierStartDate specify the knock out terms -->
    <BarrierData>
      <Type>UpAndOut</Type>
      <Levels>
        <Level>0.05</Level>
      </Levels>
    </BarrierData>
    <BarrierStartDate>2024-10-01</BarrierStartDate>
    <!-- we require exactly one Floating and one Fixed Leg -->
    <LegData>
      <LegType>Floating</LegType>
      ...
    </LegData>
    <LegData>
      <LegType>Fixed</LegType>
      ...
    </LegData>
  </KnockOutSwapData>
</Trade>
```

Listing 169: Payoff script for a BestOfAssetOrCashRainbowOption.

```
REQUIRE SIZE(Underlyings) == SIZE(Weights);
NUMBER u, thisPrice, bestPrice, Payoff, currentNotional;
bestPrice = Strike;
FOR u IN (1, SIZE(Underlyings)) DO
    thisPrice = Underlyings[u](Expiry) * Weights[u];
    IF thisPrice > bestPrice THEN
        bestPrice = thisPrice;
    END;
END;
Option = LongShort * Notional * PAY(bestPrice, Expiry, Settlement, PayCcy);
currentNotional = Notional * Strike;
```

Listing 170: Payoff script for a WorstOfAssetOrCashRainbowOption.

```
REQUIRE SIZE(Underlyings) == SIZE(Weights);
NUMBER u, thisPrice, worstPrice, Payoff, currentNotional;
worstPrice = Strike;
FOR u IN (1, SIZE(Underlyings)) DO
    thisPrice = Underlyings[u](Expiry) * Weights[u];
    IF thisPrice < worstPrice THEN
        worstPrice = thisPrice;
    END;
END;
Option = LongShort * Notional * PAY(worstPrice, Expiry, Settlement, PayCcy);
currentNotional = Notional * Strike;
```

Listing 171: Payoff script for a MaxRainbowOption.

```
REQUIRE SIZE(Underlyings) == SIZE(Weights);

NUMBER u, thisPrice, maxPrice, Payoff, ExerciseProbability, currentNotional;
maxPrice = 0;
FOR u IN (1, SIZE(Underlyings)) DO
    thisPrice = Underlyings[u](Expiry) * Weights[u];
    IF thisPrice > maxPrice THEN
        maxPrice = thisPrice;
    END;
END;

Payoff = max(PutCall * (maxPrice - Strike), 0);

Option = LongShort * Notional * PAY(Payoff, Expiry, Settlement, PayCcy);

IF Payoff > 0 THEN
    ExerciseProbability = 1;
END;
currentNotional = Notional * Strike;
```

Listing 172: Payoff script for a MinRainbowOption.

```
REQUIRE SIZE(Underlyings) == SIZE(Weights);
REQUIRE SIZE(Underlyings) > 0;

NUMBER u, thisPrice, minPrice, Payoff, ExerciseProbability, currentNotional;
minPrice = Underlyings[1](Expiry) * Weights[1];
FOR u IN (1, SIZE(Underlyings)) DO
    thisPrice = Underlyings[u](Expiry) * Weights[u];
    IF thisPrice < minPrice THEN
        minPrice = thisPrice;
    END;
END;

Payoff = max(PutCall * (minPrice - Strike), 0);

Option = LongShort * Notional * PAY(Payoff, Expiry, Settlement, PayCcy);

IF Payoff > 0 THEN
    ExerciseProbability = 1;
END;
currentNotional = Notional * Strike;
```

Listing 173: Payoff script for a EuropeanRainbowCallSpreadOption.

```
REQUIRE SIZE(Underlyings) == SIZE(Weights);
NUMBER perf[SIZE(Underlyings)], return, u;
FOR u IN (1, SIZE(Underlyings)) DO
  perf[u] = Underlyings[u](Expiry) / InitialStrikes[u];
END;
SORT (perf);
FOR u IN (1, SIZE(Underlyings)) DO
  return = return + Weights[u] * perf[SIZE(Underlyings) + 1 - u];
END;
Option = PAY( Notional * min( max( Floor, return - 1 ), Cap ), Expiry,
                                                Settlement, PayCcy );
```

Listing 174: Payoff script for a RainbowCallSpreadBarrierOption.

```
REQUIRE SIZE(Underlyings) == SIZE(Weights);
REQUIRE Floor <= Cap;
NUMBER performance, perf[SIZE(Underlyings)], return, u, d, payoff, knockedIn;

FOR u IN (1, SIZE(Underlyings), 1) DO
  perf[u] = Underlyings[u](Expiry) / InitialPrices[u];
END;
SORT (perf);

FOR u IN (1, SIZE(Underlyings), 1) DO
  return = return + Weights[u] * perf[SIZE(Underlyings) + 1 - u];
END;

IF BermudanBarrier == 1 THEN
  FOR d IN (1, SIZE(BarrierSchedule), 1) DO
    IF knockedIn == 0 THEN
      FOR u IN (1, SIZE(Underlyings), 1) DO
        performance = Underlyings[u](BarrierSchedule[d]) / InitialPrices[u];
        IF performance <= BarrierLevel THEN
          knockedIn = 1;
        END;
      END;
    END;
  END;
ELSE
  FOR u IN (1, SIZE(perf), 1) DO
    IF perf[u] <= BarrierLevel THEN
      knockedIn = 1;
    END;
  END;
END;

payoff = min( max( Floor, return - Strike ), Cap );
Option = LongShort * PAY(Notional * Gearing * payoff * knockedIn,
                         Expiry, Settlement, PayCcy);
```

Listing 175: Payoff script for a AsianRainbowCallSpreadOption.

```
REQUIRE SIZE(Underlyings) == SIZE(Weights);
NUMBER perf[SIZE(Underlyings)], return, d, u;
FOR u IN (1, SIZE(Underlyings), 1) DO
  FOR d IN (1, SIZE(AveragingDates), 1) DO
    perf[u] = perf[u] + Underlyings[u](AveragingDates[d]);
  END;
  perf[u] = perf[u] / SIZE(AveragingDates);
END;
SORT (perf);
FOR u IN (1, SIZE(Underlyings), 1) DO
  return = return + Weights[u] * perf[SIZE(Underlyings) + 1 - u];
END;
Option = LongShort * PAY( Notional * min( max( Floor, return - 1 ), Cap ), Expiry,
                         Settlement, PayCcy );
```

Listing 176: Payoff script for a WorstPerformanceRainbowOption01.

```
REQUIRE SIZE(Underlyings) == SIZE(InitialPrices);
REQUIRE ObservationDate <= SettlementDate;

NUMBER u, indexInitial, indexFinal, performance;
NUMBER worstPerformance, payoff, premium;

FOR u IN (1, SIZE(Underlyings), 1) DO
  indexInitial = InitialPrices[u];
  indexFinal = Underlyings[u](ObservationDate);
  performance = indexFinal / indexInitial;

  IF {u == 1} OR {performance < worstPerformance} THEN
    worstPerformance = performance;
  END;
END;

payoff = LOGPAY(Quantity * (worstPerformance - 1), ObservationDate,
                SettlementDate, PayCcy, 1, Payoff);

IF worstPerformance < 1 THEN
  payoff = payoff * PayoffMultiplier;
END;

premium = LOGPAY(Premium, PremiumDate, PremiumDate, PayCcy, 0, Premium);

Option = LongShort * (payoff - premium);
```

Listing 177: Payoff script for a WorstPerformanceRainbowOption02.

```
REQUIRE SIZE(Underlyings) == SIZE(InitialPrices);
REQUIRE ObservationDate <= SettlementDate;
REQUIRE Floor <= 0;

NUMBER u, initialPrice, finalPrice, performance;
NUMBER worstPerformance, payoff, premium;

FOR u IN (1, SIZE(Underlyings), 1) DO
  initialPrice = InitialPrices[u];
  finalPrice = Underlyings[u](ObservationDate);
  performance = finalPrice / initialPrice;

  IF {u == 1} OR {performance < worstPerformance} THEN
    worstPerformance = performance;
  END;
END;

IF worstPerformance > 1 THEN
  payoff = PayoffMultiplier * (worstPerformance - 1);
ELSE
  IF worstPerformance < 1 THEN
    payoff = max(Floor, worstPerformance - 1);
  ELSE
    payoff = 0;
  END;
END;

payoff = Quantity * LOGPAY(payoff, ObservationDate, SettlementDate,
                           PayCcy, 1, Payoff);
premium = LOGPAY(Premium, PremiumDate, PremiumDate, PayCcy,
                 0, Premium);

Option = LongShort * (payoff - premium);
```

Listing 178: Payoff script for a WorstPerformanceRainbowOption03.

```
REQUIRE SIZE(Underlyings) == SIZE(InitialPrices);
REQUIRE ObservationDate <= SettlementDate;
REQUIRE Floor <= Cap;

NUMBER indexInitial, indexFinal, performance, d;
NUMBER worstPerformance, payoff, premium, knockedIn, u;

FOR u IN (1, SIZE(Underlyings), 1) DO
  indexInitial = InitialPrices[u];
  indexFinal = Underlyings[u](ObservationDate);
  performance = indexFinal / indexInitial;

  IF {u == 1} OR {performance < worstPerformance} THEN
    worstPerformance = performance;
  END;
END;

IF BermudanBarrier == 1 THEN
  FOR d IN (1, SIZE(BarrierSchedule), 1) DO
    IF knockedIn == 0 THEN
      FOR u IN (1, SIZE(Underlyings), 1) DO
        indexInitial = InitialPrices[u];
        indexFinal = Underlyings[u](BarrierSchedule[d]);
        performance = indexFinal / indexInitial;

        IF performance <= BarrierLevel THEN
          knockedIn = 1;
        END;
      END;
    END;
  END;
ELSE
  IF worstPerformance <= BarrierLevel THEN
    knockedIn = 1;
  END;
END;

payoff = min(Cap, max(Floor, worstPerformance - Strike));
payoff = LOGPAY(Quantity * payoff * knockedIn, ObservationDate,
                SettlementDate, PayCcy, 1, Payoff);

IF worstPerformance < 1 THEN
  payoff = payoff * PayoffMultiplier;
END;

premium = LOGPAY(Premium, PremiumDate, PremiumDate,
                 PayCcy, 0, Premium);

Option = LongShort * (payoff - premium);
```

Listing 179: Payoff script for a WorstPerformanceRainbowOption04.

```
REQUIRE SIZE(Underlyings) == SIZE(InitialPrices);
REQUIRE ObservationDate <= SettlementDate;
REQUIRE Floor <= Cap;

NUMBER indexInitial, indexFinal, performance, d;
NUMBER worstPerformance, payoff, premium, knockedIn, u;

FOR u IN (1, SIZE(Underlyings), 1) DO
  indexInitial = InitialPrices[u];
  indexFinal = Underlyings[u](ObservationDate);
  performance = indexFinal / indexInitial;

  IF {u == 1} OR {performance < worstPerformance} THEN
    worstPerformance = performance;
  END;
END;

IF BermudanBarrier == 1 THEN
  FOR d IN (1, SIZE(BarrierSchedule), 1) DO
    IF knockedIn == 0 THEN
      FOR u IN (1, SIZE(Underlyings), 1) DO
        indexInitial = InitialPrices[u];
        indexFinal = Underlyings[u](BarrierSchedule[d]);
        performance = indexFinal / indexInitial;

        IF performance <= BarrierLevel THEN
          knockedIn = 1;
        END;
      END;
    END;
  END;
ELSE
  IF worstPerformance <= BarrierLevel THEN
    knockedIn = 1;
  END;
END;

payoff = worstPerformance - Strike;
IF knockedIn == 0 THEN
  payoff = min(Cap, max(Floor, PayoffMultiplier * payoff));
END;

payoff = LOGPAY(Quantity * payoff, ObservationDate,
                SettlementDate, PayCcy, 1, Payoff);

premium = LOGPAY(Premium, PremiumDate, PremiumDate,
                 PayCcy, 0, Premium);

Option = LongShort * (payoff - premium);
```

Listing 180: Payoff script for a WorstPerformanceRainbowOption05.

```
REQUIRE SIZE(Underlyings) == SIZE(InitialPrices);
REQUIRE ObservationDate <= SettlementDate;
REQUIRE BarrierType == 1 OR BarrierType == 2;

NUMBER indexInitial, indexFinal, performance;
NUMBER worstPerformance, payoff, premium, knockedIn, u;

FOR u IN (1, SIZE(Underlyings), 1) DO
  indexInitial = InitialPrices[u];
  indexFinal = Underlyings[u](ObservationDate);
  performance = indexFinal / indexInitial;

  IF {u == 1} OR {performance < worstPerformance} THEN
    worstPerformance = performance;
  END;
END;

IF {{BarrierType == 1 OR BarrierType == 4}
      AND worstPerformance <= BarrierLevel}
OR {{BarrierType == 2 OR BarrierType == 3}
      AND worstPerformance >= BarrierLevel} THEN
  knockedIn = 1;
END;

IF knockedIn == 0 THEN
  payoff = 0;
ELSE
  payoff = max(0, PutCall * (worstPerformance - Strike));
END;

payoff = LOGPAY(Quantity * payoff, ObservationDate,
                SettlementDate, PayCcy, 1, Payoff);

premium = LOGPAY(Premium, PremiumDate, PremiumDate,
                 PayCcy, 0, Premium);

Option = LongShort * (payoff - premium);
```

Listing 181: Payoff script for a VarianceOption.

```
REQUIRE {Notional >= 0} AND {Strike >= 0};

NUMBER expectedN, realisedVariance, currPrice, currentNotional;
NUMBER prevPrice, payoff, realisedVariation, strike, premium, d;

FOR d IN (2, SIZE(ValuationSchedule), 1) DO
  currPrice = Underlying(ValuationSchedule[d]);
  prevPrice = Underlying(ValuationSchedule[d-1]);
  realisedVariance = realisedVariance + pow(ln(currPrice/prevPrice), 2);
END;

expectedN = SIZE(ValuationSchedule) - 1;
realisedVariance = (252/expectedN) * realisedVariance;

IF SquaredPayoff == 1 THEN
  realisedVariation = realisedVariance;
  currentNotional = pow(100, 2) * Notional / (2 * 100 * VarianceReference);
  strike = pow(Strike, 2);
ELSE
  realisedVariation = sqrt(realisedVariance);
  currentNotional = 100 * Notional;
  strike = Strike;
END;

payoff = currentNotional * max(PutCall * (realisedVariation - strike), 0);

NUMBER ExerciseProbability;
IF payoff > 0 THEN
  ExerciseProbability = 1;
END;

premium = PAY(PremiumAmount, PremiumDate, SettlementDate, PayCcy);
payoff = PAY(payoff, ValuationSchedule[SIZE(ValuationSchedule)],
             SettlementDate, PayCcy);
Option = LongShort * (payoff - premium);
```

Listing 182: Payoff script for a KIKOVarianceSwap.

```
REQUIRE {Notional >= 0} AND {Strike >= 0};
REQUIRE {Cap >= 0} AND {Floor >= 0};

alive = 1;
FOR d IN (2, SIZE(ValuationSchedule), 1) DO
  IF alive == 1 THEN
    currPrice = Underlying(ValuationSchedule[d]);
    prevPrice = Underlying(ValuationSchedule[d-1]);
    realisedVariance = realisedVariance + pow(ln(currPrice/prevPrice), 2);

    IF BarrierType == 3 OR BarrierType == 4 THEN
      daysBeforeKO = daysBeforeKO + 1;
    END;

    IF {BarrierType == 3 AND currPrice <= BarrierLevel} OR
       {BarrierType == 4 AND currPrice >= BarrierLevel} THEN
      alive = 0;
    END;

    IF knockedIn == 0 THEN
      IF {BarrierType == 1 AND currPrice <= BarrierLevel} OR
         {BarrierType == 2 AND currPrice >= BarrierLevel} THEN
        knockedIn = 1;
      END;
    END;
  END;
END;

expectedN = SIZE(ValuationSchedule) - 1;
realisedVariance = (252/expectedN) * realisedVariance;

IF SquaredPayoff == 1 THEN
  realisedVariation = realisedVariance;
  currentNotional = pow(100, 2) * Notional / (2 * 100 * Strike);
  strike = pow(Strike, 2);
ELSE
  realisedVariation = sqrt(realisedVariance);
  currentNotional = 100 * Notional;
  strike = Strike;
END;

IF Floor > 0 THEN
  IF SquaredPayoff == 1 THEN
    floor = pow(Floor, 2);
  ELSE
    floor = Floor;
  END;
  realisedVariation = max(floor * strike, realisedVariation);
END;
IF Cap > 0 THEN
  IF SquaredPayoff == 1 THEN
    cap = pow(Cap, 2);
  ELSE
    cap = Cap;
  END;
  realisedVariation = min(cap * strike, realisedVariation);
END;

payoff = LongShort * knockedIn * (daysBeforeKO / expectedN) *
         currentNotional * (realisedVariation - strike);

Swap = PAY(payoff, ValuationSchedule[SIZE(ValuationSchedule)],
           SettlementDate, PayCcy);
```

Listing 183: Payoff script for a CorridorVarianceSwap.

```
REQUIRE {Notional >= 0} AND {Strike >= 0};
REQUIRE UpperBarrierLevel >= LowerBarrierLevel;

FOR d IN (2, SIZE(ValuationSchedule), 1) DO
  currPrice = Underlying(ValuationSchedule[d]);
  prevPrice = Underlying(ValuationSchedule[d-1]);

  IF {CountBothObservations == 1 AND
      currPrice >= LowerBarrierLevel AND currPrice <= UpperBarrierLevel AND
      prevPrice >= LowerBarrierLevel AND prevPrice <= UpperBarrierLevel} OR
     {CountBothObservations == -1 AND
      prevPrice >= LowerBarrierLevel AND prevPrice <= UpperBarrierLevel} THEN
        realisedVariance = realisedVariance + pow(ln(currPrice/prevPrice), 2);
        accruedDays = accruedDays + 1;
  END;
END;

expectedN = SIZE(ValuationSchedule) - 1;
realisedVariance = (252/expectedN) * realisedVariance;

IF AccrualAdjustment == -1 THEN
  accruedDays = expectedN;
END;

IF SquaredPayoff == 1 THEN
  realisedVariation = realisedVariance;
  currentNotional = pow(100, 2) * Notional / (2 * 100 * Strike);
  strike = pow(Strike, 2);
  adjustedStrike = (accruedDays / expectedN) * strike;
ELSE
  realisedVariation = sqrt(realisedVariance);
  currentNotional = 100 * Notional;
  strike = Strike;
  adjustedStrike = sqrt(accruedDays/expectedN) * strike;
END;

IF Floor > 0 THEN
  IF SquaredPayoff == 1 THEN
    floor = pow(Floor, 2);
  ELSE
    floor = Floor;
  END;
  realisedVariation = max(floor * adjustedStrike, realisedVariation);
END;
IF Cap > 0 THEN
  IF SquaredPayoff == 1 THEN
    cap = pow(Cap, 2);
  ELSE
    cap = Cap;
  END;
  realisedVariation = min(cap * adjustedStrike, realisedVariation);
END;

payoff = LongShort * currentNotional * (realisedVariation - adjustedStrike);

Swap = PAY(payoff, ValuationSchedule[SIZE(ValuationSchedule)],
           SettlementDate, PayCcy);
```

Listing 184: Payoff script for a KIKOCorridorVarianceSwap.

```
REQUIRE {Notional >= 0} AND {Strike >= 0} AND {KIKOBarrierLevel > 0};
REQUIRE CorridorUpperBarrierLevel >= CorridorLowerBarrierLevel;

n = SIZE(ValuationSchedule);

alive = 1;
FOR d IN (2, n, 1) DO
  currPrice = Underlying(ValuationSchedule[d]);
  prevPrice = Underlying(ValuationSchedule[d-1]);

  IF alive == 1 THEN
    IF {CountBothObservations == 1 AND
        currPrice >= CorridorLowerBarrierLevel AND currPrice <= CorridorUpperBarrierLevel AND
        prevPrice >= CorridorLowerBarrierLevel AND prevPrice <= CorridorUpperBarrierLevel} OR
       {CountBothObservations == -1 AND
        prevPrice >= CorridorLowerBarrierLevel AND prevPrice <= CorridorUpperBarrierLevel} THEN
      realisedVariance = realisedVariance + pow(ln(currPrice/prevPrice), 2);
      accruedDays = accruedDays + 1;
    END;

    IF {KIKOBarrierType == 3 AND currPrice <= KIKOBarrierLevel} OR
       {KIKOBarrierType == 4 AND currPrice >= KIKOBarrierLevel} THEN
      alive = 0;
    END;
  END;

  IF knockedIn == 0 THEN
    IF {KIKOBarrierType == 1 AND currPrice <= KIKOBarrierLevel} OR
       {KIKOBarrierType == 2 AND currPrice >= KIKOBarrierLevel} THEN
      knockedIn = 1;
    END;
  END;

  IF {alive == 0 OR d == n} AND {calculated == 0} THEN
    calculated = 1;
    expectedN = n - 1;
    realisedVariance = (252/expectedN) * realisedVariance;

    IF AccrualAdjustment == -1 THEN
      accruedDays = expectedN;
    END;

    currentNotional = pow(100, 2) * Notional / (2 * 100 * Strike);
    strike = pow(Strike, 2);
    adjustedStrike = (accruedDays / expectedN) * strike;

    IF Floor > 0 THEN
      floor = pow(Floor, 2);
      realisedVariance = max(floor * adjustedStrike, realisedVariance);
    END;
    IF Cap > 0 THEN
      cap = pow(Cap, 2);
      realisedVariance = min(cap * adjustedStrike, realisedVariance);
    END;

    IF {{KIKOBarrierType == 1 OR KIKOBarrierType == 2} AND {knockedIn == 1}} OR
       {{KIKOBarrierType == 3 OR KIKOBarrierType == 4} AND {alive == 0}} THEN
      TriggerProbability = 1;
    END;

    IF KIKOBarrierType == 3 OR KIKOBarrierType == 4 THEN
      knockedIn = 1;
    END;

    payoff = LongShort * knockedIn * currentNotional
             * (realisedVariance - adjustedStrike);

    Swap = LOGPAY(payoff, ValuationSchedule[d], SettlementSchedule[d], PayCcy);
  END;
END;
```

Listing 185: Payoff script for a ConditionalVarianceSwap01.

```
REQUIRE {Notional >= 0} AND {Strike > 0};

FOR d IN (2, SIZE(ValuationSchedule), 1) DO
  currPrice = Underlying(ValuationSchedule[d]);
  prevPrice = Underlying(ValuationSchedule[d-1]);

  IF {CountBothObservations == 1 AND {
      {{BarrierType == 1 OR BarrierType == 4} AND
        currPrice <= BarrierLevel AND prevPrice <= BarrierLevel} OR
      {{BarrierType == 2 OR BarrierType == 3} AND
        currPrice >= BarrierLevel AND prevPrice >= BarrierLevel}}}
  OR {CountBothObservations == -1 AND {
      {{BarrierType == 1 OR BarrierType == 4} AND
        prevPrice <= BarrierLevel} OR
      {{BarrierType == 2 OR BarrierType == 3} AND
        prevPrice <= BarrierLevel} }}
  THEN
    realisedVariance = realisedVariance + pow(ln(currPrice/prevPrice), 2);
    accruedDays = accruedDays + 1;
  END;
END;

expectedN = SIZE(ValuationSchedule) - 1;
realisedVariance = (252/expectedN) * realisedVariance;

IF AccrualAdjustment == -1 THEN
  accruedDays = expectedN;
END;

IF SquaredPayoff == 1 THEN
  realisedVariation = realisedVariance;
  currentNotional = pow(100, 2) * Notional / (2 * 100 * Strike);
  strike = pow(Strike, 2);
  adjustedStrike = (accruedDays / expectedN) * strike;
ELSE
  realisedVariation = sqrt(realisedVariance);
  currentNotional = 100 * Notional;
  strike = Strike;
  adjustedStrike = sqrt(accruedDays/expectedN) * strike;
END;

IF Floor > 0 THEN
  IF SquaredPayoff == 1 THEN
    floor = pow(Floor, 2);
  ELSE
    floor = Floor;
  END;
  realisedVariation = max(floor * adjustedStrike, realisedVariation);
END;
IF Cap > 0 THEN
  IF SquaredPayoff == 1 THEN
    cap = pow(Cap, 2);
  ELSE
    cap = Cap;
  END;
  realisedVariation = min(cap * adjustedStrike, realisedVariation);
END;

payoff = LongShort * currentNotional * (realisedVariation - adjustedStrike);

Swap = PAY(payoff, ValuationSchedule[SIZE(ValuationSchedule)],
           SettlementDate, PayCcy);
```

Listing 186: Payoff script for a ConditionalVarianceSwap02.

```
REQUIRE {Notional >= 0} AND {Strike > 0} AND {VarianceReference > 0};

FOR d IN (2, SIZE(ValuationSchedule), 1) DO
  currPrice = Underlying(ValuationSchedule[d]);
  prevPrice = Underlying(ValuationSchedule[d-1]);

  IF {CountBothObservations == 1 AND {
      {{BarrierType == 1 OR BarrierType == 4} AND
        currPrice <= BarrierLevel AND prevPrice <= BarrierLevel} OR
      {{BarrierType == 2 OR BarrierType == 3} AND
        currPrice >= BarrierLevel AND prevPrice >= BarrierLevel}}}
  OR {CountBothObservations == -1 AND {
      {{BarrierType == 1 OR BarrierType == 4} AND
        prevPrice <= BarrierLevel} OR
      {{BarrierType == 2 OR BarrierType == 3} AND
        prevPrice <= BarrierLevel} }}
  THEN
    realisedVariance = realisedVariance + pow(ln(currPrice/prevPrice), 2);
    accruedDays = accruedDays + 1;
  END;
END;

expectedN = SIZE(ValuationSchedule) - 1;
realisedVariance = (252/expectedN) * realisedVariance;

IF AccrualAdjustment == -1 THEN
  accruedDays = expectedN;
END;

IF SquaredPayoff == 1 THEN
  realisedVariation = realisedVariance;
  currentNotional = pow(100, 2) * Notional / (2 * 100 * VarianceReference);
  strike = pow(Strike, 2);
  adjustedStrike = (accruedDays / expectedN) * strike;
ELSE
  realisedVariation = sqrt(realisedVariance);
  currentNotional = 100 * Notional;
  strike = Strike;
  adjustedStrike = sqrt(accruedDays/expectedN) * strike;
END;

IF Floor > 0 THEN
  IF SquaredPayoff == 1 THEN
    floor = pow(Floor, 2);
  ELSE
    floor = Floor;
  END;
  realisedVariation = max(floor * adjustedStrike, realisedVariation);
END;
IF Cap > 0 THEN
  IF SquaredPayoff == 1 THEN
    cap = pow(Cap, 2);
  ELSE
    cap = Cap;
  END;
  realisedVariation = min(cap * adjustedStrike, realisedVariation);
END;

payoff = LongShort * currentNotional * (realisedVariation - adjustedStrike);

Swap = PAY(payoff, ValuationSchedule[SIZE(ValuationSchedule)],
           SettlementDate, PayCcy);
```

Listing 188: Payoff script for a PairwiseVarianceSwap.

```
REQUIRE {SIZE(Underlyings) == 2} AND {SIZE(UnderlyingStrikes) == 2}
REQUIRE {SIZE(UnderlyingNotionals) == 2} AND {UnderlyingStrikes[1] >= 0};
REQUIRE {UnderlyingStrikes[2] >= 0} AND {BasketStrike >= 0};
REQUIRE {UnderlyingNotionals[1] >= 0} AND {UnderlyingNotionals[2] >= 0};
REQUIRE {BasketNotional >= 0} AND {PayoffLimit > 0};
REQUIRE {SIZE(ValuationSchedule) == SIZE(LaggedValuationSchedule)};

FOR d IN (1, SIZE(ValuationSchedule)-1, 1) DO
  performance1 = ln(Underlyings[1](LaggedValuationSchedule[d]) /
                    Underlyings[1](ValuationSchedule[d]));
  performance2 = ln(Underlyings[2](LaggedValuationSchedule[d]) /
                    Underlyings[2](ValuationSchedule[d]));
  basketPerformance = (performance1 + performance2) / 2;

  realisedVariance1 = realisedVariance1 + pow(performance1, 2);
  realisedVariance2 = realisedVariance2 + pow(performance2, 2);
  realisedVarianceBasket = realisedVarianceBasket + pow(basketPerformance, 2);
END;

expectedN = SIZE(ValuationSchedule) - 1;
realisedVariance1 = 252 / (expectedN * AccrualLag) * realisedVariance1;
realisedVariance2 = 252 / (expectedN * AccrualLag) * realisedVariance2;
realisedVarianceBasket = 252 / (expectedN * AccrualLag) * realisedVarianceBasket;

currentNotional1 = pow(100, 2) * UnderlyingNotionals[1] /
                                 (2 * 100 * UnderlyingStrikes[1]);
currentNotional2 = pow(100, 2) * UnderlyingNotionals[2] /
                                 (2 * 100 * UnderlyingStrikes[2]);
currentNotionalBasket = pow(100, 2) * BasketNotional / (2 * 100 * BasketStrike);
strike1 = pow(UnderlyingStrikes[1], 2);
strike2 = pow(UnderlyingStrikes[2], 2);
strikeBasket = pow(BasketStrike, 2);

IF Floor > 0 THEN
  floor = pow(Floor, 2);
  realisedVariance1 = max(floor * strike1, realisedVariance1);
  realisedVariance2 = max(floor * strike2, realisedVariance2);
  realisedVarianceBasket = max(floor * strikeBasket, realisedVarianceBasket);
END;
IF Cap > 0 THEN
  cap = pow(Cap, 2);
  realisedVariance1 = min(cap * strike1, realisedVariance1);
  realisedVariance2 = min(cap * strike2, realisedVariance2);
  realisedVarianceBasket = min(cap * strikeBasket, realisedVarianceBasket);
END;

equityAmount1 = currentNotional1 * (realisedVariance1 - strike1);
equityAmount2 = currentNotional2 * (realisedVariance2 - strike2);
equityAmountBasket = currentNotionalBasket * (realisedVarianceBasket - strikeBasket);
pairEquityAmount = equityAmount1 + equityAmount2 + equityAmountBasket;

maxPairEquityAmount = PayoffLimit * (abs(UnderlyingNotionals[1]) +
                                     abs(UnderlyingNotionals[2]));
minPairEquityAmount = -maxPairEquityAmount;

pairEquityAmount = max(minPairEquityAmount, pairEquityAmount);
pairEquityAmount = min(maxPairEquityAmount, pairEquityAmount);

Swap = PAY(LongShort * pairEquityAmount, ValuationSchedule[SIZE(ValuationSchedule)],
           SettlementDate, PayCcy);
```

Listing 189: Payoff script for a VarianceDispersionSwap.

```
D = SIZE(ValuationSchedule);
expectedN = D - 1;

FOR u IN (1, n1, 1) DO
  FOR d IN (2, D, 1) DO
    currPrice = Underlyings1[u](ValuationSchedule[d]);
    prevPrice = Underlyings1[u](ValuationSchedule[d-1]);
    realisedVariance = realisedVariance + pow(ln(currPrice/prevPrice), 2);
  END;

  realisedVariance = (252/expectedN) * realisedVariance + Spreads1[u];
  currentNotional = pow(100, 2) * Notionals1[u] / (2 * 100 * Strikes1[u]);
  strike = pow(Strikes1[u], 2);

  IF Floors1[u] > 0 THEN
    floor = pow(Floors1[u], 2);
    realisedVariance = max(floor * strike, realisedVariance);
  END;
  IF Caps1[u] > 0 THEN
    cap = pow(Caps1[u], 2);
    realisedVariance = min(cap * strike, realisedVariance);
  END;
  payoff1 = payoff1 + currentNotional * (realisedVariance - strike) * Weights1[u];
END;

FOR u IN (1, n2, 1) DO
  FOR d IN (2, D, 1) DO
    currPrice = Underlyings2[u](ValuationSchedule[d]);
    prevPrice = Underlyings2[u](ValuationSchedule[d-1]);
    realisedVariance = realisedVariance + pow(ln(currPrice/prevPrice), 2);
  END;

  realisedVariance = (252/expectedN) * realisedVariance + Spreads2[u];
  currentNotional = pow(100, 2) * Notionals2[u] / (2 * 100 * Strikes2[u]);
  strike = pow(Strikes2[u], 2);

  IF Floors2[u] > 0 THEN
    floor = pow(Floors2[u], 2);
    realisedVariance = max(floor * strike, realisedVariance);
  END;
  IF Caps2[u] > 0 THEN
    cap = pow(Caps2[u], 2);
    realisedVariance = min(cap * strike, realisedVariance);
  END;
  payoff2 = payoff2 + currentNotional * (realisedVariance - strike) * Weights2[u];
END;

payoff = LongShort * (payoff1 - payoff2);

Swap = PAY(payoff, ValuationSchedule[D], SettlementDate, PayCcy);

NUMBER currentNotional1, currentNotional2;

FOR u IN (1, n1, 1) DO
  currentNotional1 = currentNotional1 + (pow(100, 2) * Notionals1[u]
                                         / (2 * 100 * Strikes1[u]));
END;
FOR u IN (1, n2, 1) DO
  currentNotional2 = currentNotional2 + (pow(100, 2) * Notionals2[u]
                                         / (2 * 100 * Strikes2[u]));
END;
```

Listing 190: Payoff script for a CorridorVarianceDispersionSwap.

```
N = SIZE(Underlyings1);
D = SIZE(ValuationSchedule);
expectedN = D - 1;

FOR u IN (1, N, 1) DO
  FOR d IN (2, D, 1) DO
    currPrice1 = Underlyings1[u](ValuationSchedule[d]);
    prevPrice1 = Underlyings1[u](ValuationSchedule[d-1]);

    IF {CountBothObservations == 1 AND
        currPrice1 >= LowerBarrierLevels[u] AND currPrice1 <= UpperBarrierLevels[u] AND
        prevPrice1 >= LowerBarrierLevels[u] AND prevPrice1 <= UpperBarrierLevels[u]} OR
       {CountBothObservations == -1 AND
        prevPrice >= LowerBarrierLevels[u] AND prevPrice <= UpperBarrierLevels[u]} THEN
          currPrice2 = Underlyings2[u](ValuationSchedule[d]);
          prevPrice2 = Underlyings2[u](ValuationSchedule[d-1]);
          realisedVariance1 = realisedVariance1 + pow(ln(currPrice1/prevPrice1), 2);
          realisedVariance2 = realisedVariance2 + pow(ln(currPrice2/prevPrice2), 2);
          accruedDays = accruedDays + 1;
    END;
  END;

  IF AccrualAdjustment == -1 THEN
    accruedDays = expectedN;
  END;

  realisedVariance1 = (252/expectedN) * realisedVariance1 + Spreads1[u];
  realisedVariance2 = (252/expectedN) * realisedVariance2 + Spreads2[u];

  currentNotional1 = pow(100, 2) * Notionals1[u] / (2 * 100 * Strikes1[u]);
  currentNotional2 = pow(100, 2) * Notionals2[u] / (2 * 100 * Strikes2[u]);

  strike1 = pow(Strikes1[u], 2);
  strike2 = pow(Strikes2[u], 2);

  adjustedStrike1 = (accruedDays / expectedN) * strike1;
  adjustedStrike2 = (accruedDays / expectedN) * strike2;

  IF Floors1[u] > 0 THEN
    floor1 = pow(Floors1[u], 2);
    realisedVariance1 = max(floor1 * adjustedStrike1, realisedVariance1);
  END;
  IF Floors2[u] > 0 THEN
    floor2 = pow(Floors2[u], 2);
    realisedVariance2 = max(floor2 * adjustedStrike2, realisedVariance2);
  END;
  IF Caps1[u] > 0 THEN
    cap1 = pow(Caps1[u], 2);
    realisedVariance1 = max(cap1 * adjustedStrike1, realisedVariance1);
  END;
  IF Caps2[u] > 0 THEN
    cap2 = pow(Caps2[u], 2);
    realisedVariance2 = max(cap2 * adjustedStrike2, realisedVariance2);
  END;

  payoff1 = currentNotional1 * (realisedVariance1 - adjustedStrike1);
  payoff2 = currentNotional2 * (realisedVariance2 - adjustedStrike2);
  payoff = payoff + LongShort * Weights[u] * (payoff1 - payoff2);
END;

Swap = PAY(payoff, ValuationSchedule[D], SettlementDate, PayCcy);
```

Listing 191: Payoff script for a KOCorridorVarianceDispersionSwap.

```
N = SIZE(Underlyings1);
Dk = SIZE(KnockOutSchedule);
expectedN = D - 1;

FOR d IN (1, Dk, 1) DO
  IF KnockOutSchedule[d] > VarianceAccrualStartDate THEN
    expectedN = expectedN + 1;
  END;
END;

alive = 1;
FOR d IN (1, Dk, 1) DO
  IF alive == 1 THEN
    IF KnockOutSchedule[d] > VarianceAccrualStartDate THEN
      FOR u IN (1, N, 1) DO
        currPrice1 = Underlyings1[u](KnockOutSchedule[d]);
        prevPrice1 = Underlyings1[u](KnockOutSchedule[d-1]);

        IF {CountBothObservations == 1 AND
            currPrice1 >= CorridorLowerBarrierLevels[u] AND currPrice1 <= CorridorUpperBarrierLevels[u] AND
            prevPrice1 >= CorridorLowerBarrierLevels[u] AND prevPrice1 <= CorridorUpperBarrierLevels[u]} OR
           {CountBothObservations == -1 AND
            prevPrice >= CorridorLowerBarrierLevels[u] AND prevPrice <= CorridorUpperBarrierLevels[u]} THEN
          currPrice2 = Underlyings2[u](KnockOutSchedule[d]);
          prevPrice2 = Underlyings2[u](KnockOutSchedule[d-1]);
          realisedVariance1 = realisedVariance1 + pow(ln(currPrice1/prevPrice1), 2);
          realisedVariance2 = realisedVariance2 + pow(ln(currPrice2/prevPrice2), 2);
          accruedDays = accruedDays + 1;
        END;
      END;
    END;

    FOR u IN (1, N, 1) DO
      currPrice1 = Underlyings1[u](KnockOutSchedule[d]);
      IF currPrice1 <= KOLowerBarrierLevels[u] OR currPrice1 >= KOUpperBarrierLevels[u] THEN
        alive = 0;
      END;
    END;
  END;

  IF {alive == 0 OR d == Dk} AND {calculated == 0} THEN
    calculated = 1;

    realisedVariance1 = (252/expectedN) * realisedVariance1 + Spreads1[u];
    realisedVariance2 = (252/expectedN) * realisedVariance2 + Spreads2[u];

    IF AccrualAdjustment == -1 THEN
      accruedDays = expectedN;
    END;

    currentNotional1 = pow(100, 2) * Notionals1[u] / (2 * 100 * Strikes1[u]);
    currentNotional2 = pow(100, 2) * Notionals2[u] / (2 * 100 * Strikes2[u]);

    strike1 = pow(Strikes1[u], 2);
    strike2 = pow(Strikes2[u], 2);

    adjustedStrike1 = (accruedDays / expectedN) * strike1;
    adjustedStrike2 = (accruedDays / expectedN) * strike2;

    IF Floors1[u] > 0 THEN
      floor1 = pow(Floors1[u], 2);
      realisedVariance1 = max(floor1 * adjustedStrike1, realisedVariance1);
    END;
    IF Floors2[u] > 0 THEN
      floor2 = pow(Floors2[u], 2);
      realisedVariance2 = max(floor2 * adjustedStrike2, realisedVariance2);
    END;
    IF Caps1[u] > 0 THEN
      cap1 = pow(Caps1[u], 2);
      realisedVariance1 = max(cap1 * adjustedStrike1, realisedVariance1);
    END;
    IF Caps2[u] > 0 THEN
      cap2 = pow(Caps2[u], 2);
      realisedVariance2 = max(cap2 * adjustedStrike2, realisedVariance2);
    END;

    payoff1 = currentNotional1 * (realisedVariance1 - adjustedStrike1);
    payoff2 = currentNotional2 * (realisedVariance2 - adjustedStrike2);
    payoff = payoff + LongShort * Weights[u] * (payoff1 - payoff2);
```

Listing 192: Payoff script for a GammaSwap.

```
REQUIRE {Notional >= 0} AND {Strike >= 0};

NUMBER d, expectedN, realisedVariance, currPrice, prevPrice, presPrice, currentNotional;
NUMBER payoff, realisedVariation, strike;

presPrice = Underlying(ValuationSchedule[1]);

FOR d IN (2, SIZE(ValuationSchedule), 1) DO
  currPrice = Underlying(ValuationSchedule[d]);
  prevPrice = Underlying(ValuationSchedule[d-1]);
  realisedVariance = realisedVariance + (currPrice/presPrice) * pow(ln(currPrice/prevPrice), 2);
END;

expectedN = SIZE(ValuationSchedule) - 1;
realisedVariance = (252/expectedN) * realisedVariance;

realisedVariation = realisedVariance;
currentNotional = pow(100, 2) * Notional / (2 * 100 * Strike);
strike = pow(Strike, 2);

payoff = LongShort * currentNotional * (realisedVariation -
            strike);

Swap = PAY(payoff, ValuationSchedule[SIZE(ValuationSchedule)],
           SettlementDate, PayCcy);
```

Listing 194: Payoff script for a BasketVarianceSwap.

```
NUMBER sumOfWeights;
FOR i IN (1, n, 1) DO
  sumOfWeights = sumOfWeights + Weights[i];
END;
REQUIRE sumOfWeights == 1;

NUMBER d, expectedN, currPrice[n], prevPrice[n];
NUMBER realisedVariance, basketVariation, realisedVariation;
NUMBER strike, cap, floor, currentNotional, payoff;

FOR d IN (2, SIZE(ValuationSchedule), 1) DO
  basketVariation = 0;
  FOR i IN (1, n, 1) DO
    currPrice[i] = Underlyings[i](ValuationSchedule[d]);
    prevPrice[i] = Underlyings[i](ValuationSchedule[d-1]);
    basketVariation = basketVariation + Weights[i] * ln(currPrice[i]/prevPrice[i]);
  END;
  realisedVariance = realisedVariance + pow(basketVariation, 2);
END;

expectedN = SIZE(ValuationSchedule) - 1;
realisedVariance = (252/expectedN) * realisedVariance;

IF SquaredPayoff == 1 THEN
  realisedVariation = realisedVariance;
  currentNotional = pow(100, 2) * Notional / (2 * 100 * Strike);
  strike = pow(Strike, 2);
ELSE
  realisedVariation = sqrt(realisedVariance);
  currentNotional = 100 * Notional;
  strike = Strike;
END;

IF Floor > 0 THEN
  IF SquaredPayoff == 1 THEN
    floor = pow(Floor, 2);
  ELSE
    floor = Floor;
  END;
  realisedVariation = max(floor * strike, realisedVariation);
END;
IF Cap > 0 THEN
  IF SquaredPayoff == 1 THEN
    cap = pow(Cap, 2);
  ELSE
    cap = Cap;
  END;
  realisedVariation = min(cap * strike, realisedVariation);
END;

payoff = LongShort * currentNotional * (realisedVariation - strike);

Swap = PAY(payoff, ValuationSchedule[SIZE(ValuationSchedule)],
           SettlementDate, PayCcy);
```

Listing 232: Commodity fixed leg outline.

```
<LegData>
  <LegType>CommodityFixed</LegType>
  <Payer>...</Payer>
  <Currency>...</Currency>
  <PaymentConvention>...</PaymentConvention>
  <PaymentLag>...</PaymentLag>
  <PaymentCalendar>...</PaymentCalendar>
  <ScheduleData>
    ...
  </ScheduleData>
  <PaymentDates>
    <PaymentDate>...</PaymentDate>
  </PaymentDates>
  <CommodityFixedLegData>
    ...
  </CommodityFixedLegData>
</LegData>
```

Listing 233: Commodity fixed leg data outline.

```
<CommodityFixedLegData>
  <Quantities>
    <Quantity>...</Quantity>
  </Quantities>
  <Prices>
    <Price>...</Price>
  </Prices>
  <CommodityPayRelativeTo>...</CommodityPayRelativeTo>
  <Tag>...</Tag>
</CommodityFixedLegData>
```

Listing 234: Commodity floating leg outline.

```
<LegData>
  <LegType>CommodityFloating</LegType>
  <Payer>...</Payer>
  <Currency>...</Currency>
  <PaymentConvention>...</PaymentConvention>
  <PaymentLag>...</PaymentLag>
  <PaymentCalendar>...</PaymentCalendar>
  <ScheduleData>
    ...
  </ScheduleData>
  <PaymentDates>
    <PaymentDate>...</PaymentDate>
  </PaymentDates>
  <CommodityFloatingLegData>
    ...
  </CommodityFloatingLegData>
</LegData>
```

Listing 235: `ScheduleData` *node for monthly periods.*

```
<ScheduleData>
  <Rules>
    <StartDate>2020-01-01</StartDate>
    <EndDate>2020-04-30</EndDate>
    <Tenor>1M</Tenor>
    <Calendar>NullCalendar</Calendar>
    <Convention>Unadjusted</Convention>
    <TermConvention>Unadjusted</TermConvention>
    <Rule>Backward</Rule>
    <EndOfMonth>true</EndOfMonth>
    <AdjustEndDateToPreviousMonthEnd>false</AdjustEndDateToPreviousMonthEnd>
  </Rules>
</ScheduleData>
```

Listing 236: `ScheduleData` *node for explicit periods.*

```
<ScheduleData>
  <Dates>
    <Calendar>NullCalendar</Calendar>
    <Convention>Unadjusted</Convention>
    <Dates>
      <Date>2019-11-21</Date>
      <Date>2019-12-19</Date>
      <Date>2020-01-21</Date>
      <Date>2020-02-20</Date>
      <Date>2020-03-20</Date>
    </Dates>
  </Dates>
</ScheduleData>
```

Listing 237: Commodity floating leg data outline.

```
<CommodityFloatingLegData>
  <Name>...</Name>
  <PriceType>...</PriceType>
  <Quantities>
    <Quantity>...</Quantity>
  </Quantities>
  <CommodityQuantityFrequency>...</CommodityQuantityFrequency>
  <CommodityPayRelativeTo>...</CommodityPayRelativeTo>
  <Spreads>
    <Spread>...</Spread>
  </Spreads>
  <Gearings>
    <Gearing>...</Gearing>
  </Gearings>
  <PricingDateRule>...</PricingDateRule>
  <PricingCalendar>...</PricingCalendar>
  <PricingLag>...</PricingLag>
  <PricingDates>
    <PricingDate>...</PricingDate>
  </PricingDates>
  <IsAveraged>...</IsAveraged>
  <IsInArrears>...</IsInArrears>
  <FutureMonthOffset>...</FutureMonthOffset>
  <DeliveryRollDays>...</DeliveryRollDays>
  <IncludePeriodEnd>...</IncludePeriodEnd>
  <ExcludePeriodStart>...</ExcludePeriodStart>
  <HoursPerDay>...</HoursPerDay>
  <UseBusinessDays>...</UseBusinessDays>
  <Tag>...</Tag>
  <DailyExpiryOffset>...</DailyExpiryOffset>
</CommodityFloatingLegData>
```

Listing 238: Example WTI averaging floating leg, first approach.

```
<LegData>
  <LegType>CommodityFloating</LegType>
  <Payer>true</Payer>
  <Currency>USD</Currency>
  <PaymentLag>2</PaymentLag>
  <PaymentConvention>Following</PaymentConvention>
  <PaymentCalendar>US-NYSE</PaymentCalendar>
  <CommodityFloatingLegData>
    <Name>NYMEX:CL</Name>
    <PriceType>FutureSettlement</PriceType>
    <Quantities>
      <Quantity>5000</Quantity>
    </Quantities>
    <IsAveraged>true</IsAveraged>
    <FutureMonthOffset>0</FutureMonthOffset>
  </CommodityFloatingLegData>
  <ScheduleData>
    <Rules>
      <StartDate>2019-09-01</StartDate>
      <EndDate>2019-10-31</EndDate>
      <Tenor>1M</Tenor>
      <Calendar>NullCalendar</Calendar>
      <Convention>Unadjusted</Convention>
      <TermConvention>Unadjusted</TermConvention>
      <Rule>Backward</Rule>
      <EndOfMonth>true</EndOfMonth>
    </Rules>
  </ScheduleData>
</LegData>
```

Listing 239: Example WTI averaging floating leg, second approach.

```
<CommodityFloatingLegData>
<Name>NYMEX:CSX</Name>
<PriceType>FutureSettlement</PriceType>
<Quantities>
  <Quantity>5000</Quantity>
</Quantities>
<IsAveraged>true</IsAveraged>
<FutureMonthOffset>0</FutureMonthOffset>
</CommodityFloatingLegData>
```

Listing 240: Equity Margin leg outline.

```
<LegData>
  <LegType>EquityMargin</LegType>
  <Payer>true</Payer>
  <Currency>EUR</Currency>
  <PaymentConvention>Following</PaymentConvention>
  <PaymentLag>2D</PaymentLag>
  <PaymentCalendar>TARGET</PaymentCalendar>
  <ScheduleData>
       <Dates>
        <Dates>
         <Date>2019-12-31</Date>
         <Date>2020-03-30</Date>
         <Date>2020-06-30</Date>
         <Date>2020-09-30</Date>
         <Date>2020-12-30</Date>
         <Date>2021-03-30</Date>
        </Dates>
     </Dates>
  </ScheduleData>
  <PaymentDates>
    <PaymentDate>...</PaymentDate>
  </PaymentDates>
  <EquityMarginLegData>
    ...
  </EquityMarginLegData>
</LegData>
```

Listing 241: Equity margin leg data outline.

```
<EquityMarginLegData>
  <Rates>
    <Rate>0.003</Rate>
  </Rates>
  <InitialMarginFactor>0.12</InitialMarginFactor>
  <Multiplier>10</Multiplier>
  <EquityLegData>
     <ReturnType>Total</ReturnType>
     <Name>RIC:.STOXX50E</Name>
     <InitialPrice>2946</InitialPrice>
     <NotionalReset>false</NotionalReset>
     <FixingDays>2</FixingDays>
  </EquityLegData>
</EquityMarginLegData>
```

Listing 261: FormulaBasedLegData configuration.

```
<LegData>
  <LegType>FormulaBased</LegType>
  <Payer>true</Payer>
  <Currency>EUR</Currency>
  <PaymentConvention>MF</PaymentConvention>
  <PaymentLag>2</PaymentLag>
  <PaymentCalendar>TARGET</PaymentCalendar>
  <DayCounter>A360</DayCounter>
   ...
  <FormulaBasedLegData>
    <Index>gtZero({USD-CMS-5Y}-0.03)*
              max(min(9.0*({EUR-CMS-10Y}-{GBP-CMS-2Y})+0.02,0.08),0.0)</Index>
    <IsInArrears>false</IsInArrears>
    <FixingDays>2</FixingDays>
  </FormulaBasedLegData>
   ...
</LegData>
```

References

[1] VRE Product Catalogue, available at http://www.vannarho.com

[2] VRE Methodology Guide, available athttp://www.vannarho.com

[3] http://www.opensourcerisk.org

[4] Open Source Risk Engine, available at http://www.opensourcerisk.org

[5] http://quantlib.org

[6] QuantLib, available at http://quantlib.org

[7] https://sourceforge.net/projects/boost/files/boost-binaries

[8] Git download and installation guidance, available at https://git-scm.com/downloads

[9] http://www.boost.org

[10] http://jupyter.org

[11] https://enzyme.mit.edu/

[12] William S. Moses, Sri Hari Krishna Narayanan, Ludger Paehler, Valentin Churavy, Michel Schanen, Jan Hueckelheim, Johannes Doerfert, and Paul Hovland, Scalable Automatic Differentiation of Multiple Parallel Paradigms through Compiler Augmentation, SC '22: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis Article No.: 60, Pages 1 - 1 (2022) https://dl.acm.org/doi/abs/10.5555/3571885.3571964

[13] R. Rebonato and P. Jaeckel, The most general methodology to create a valid correlation matrix for risk management and option pricing purposes, The Journal of Risk, 2(2), Winter 1999/2000, http://www.quarchome.org/correlationmatrix.pdf

[14] Andersen, L., and Piterbarg, V. (2010): Interest Rate Modeling, Volume I-III

[15] Roland Lichters, Roland Stamm, Donal Gallagher, *Modern Derivatives Pricing and Credit Exposure Analysis, Theory and Practice of CSA and XVA Pricing, Exposure Simulation and Backtesting*, Palgrave Macmillan, 2015.

[16] Fabrizio Anfuso, Daniel Aziz, Paul Giltinan, Klearchos Loukopoulos, *A Sound Modelling and Backtesting Framework for Forecasting Initial Margin Requirements*, http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2716279, 2016

[17] Leif B. G. Andersen, Michael Pykhtin, Alexander Sokol, *Rethinking Margin Period of Risk*, http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2719964, 2016

[18] Peter Caspers, Paul Giltinan, Paul; Lichters, Roland; Nowaczyk , Nikolai. *Forecasting Initial Margin Requirements – A Model Evaluation*, Journal of Risk Management in Financial Institutions, Vol. 10 (2017), No. 4, https://ssrn.com/abstract=2911167

[19] ISDA SIMM Methodology, version 2.5A, (based on v2.5a) https://www.isda.org/a/FBLgE/ISDA-SIMM_v2.5A.pdf