출시준비 / **Bond**

공유

Bond

소유자: 이준우 •••
4월 27, 2025에 마지막 업데이트• 이 페이지를 본 사용자의 수 보기

QuantLib 제공 커버리지 분석

인터페이스 정의서

FixedRateBond

I/O	번호	변수명	Type	Size	설명	Sample Data
In	1	evaluationDate	long	1	평가기준일	20240728
	2	settlementDays	long	1	결제일 영업일 수	2
	3	issueDate	long	1	발행일(정보성)	20201210
	4	maturityDate	long	1	만기일(정보성)	20301210
	5	notional	double	1	명목금액	6000000000
	6	couponRate	double	1	쿠폰 이자율	0.015
	7	couponDayCounter	long	1	쿠폰 이자율의 날짜 계산 관행	1
	8	numberOfCoupons	long	1	지급 쿠폰 개수	1
	9	paymentDates	long[]	numberOfCoupons	쿠폰 지급일	20241210, 20250610,
	10	realStartDates	long[]	numberOfCoupons	쿠폰 계산 시작일	20241210, 20250610,
	11	realEndDates	long[]	numberOfCoupons	쿠폰 계산 종료일	20241210, 20250610,
	12	numberOfGirrTenors	long	1	Girr 금리커브 테너개수	10

13	girrTenorDays	long[]	numberOf	Girr 금리커브 테너 날짜수	91,183, 365
			GirrTenors		0.03254, ...
15	girrDayCounter	long	1	Girr 제로금리의 날짜 계산 관행	1
16	girrInterpolator	long	1	Girr 제로금리의 Interpolation 방법	1
17	girrCompounding	long	1	Girr 제로금리의 복리계산 관행	1
18	girrFrequency	long	TBD	Girr 제로금리의 복리계산 주기	1
19	spreadOverYield	double	TBD	채권의 고유 Credit Spread	0.0015324
20	spreadOverYieldCompounding	long	1	채권 Spread의 복리계산 관행	1
21	spreadOverYieldDayCounter	long	1	채권 Spread의 날짜계산 관행	1
22	numberOfCsrTenors	long	1	csr 스프레드 커브의 테너개수	5
23	csrTenorDays	long[]	1	csr 스프레드 커브의 테너 날짜수	183, 365, ...
24	csrSpreads	double	1	csr 스프레드 커브의 spread	0.0001, 0.0005, ...

인터페이스 이슈

- 스케쥴 커버리지: 직전 지급일과 다음 지급일 사이를 이자구간으로 가정
    - 지급일 배열 + 직전 지급일 수신
    - (or) 외부에서 스케쥴 배열을 만들어서 수신
- 현금흐름 생성
    - Quantlib의 Act/Act 계산방법 상이(Ex. **기간**: 2024-12-15 ~ 2025-03-15
        - **Algo**

- 2024-12-15 ~ 2024-12-31: 16일
- 2025-01-01 ~ 2025-03-15: 74일

- Quantlib
  - 2024-12-15 ~ 2025-01-01: 17일
  - 2025-01-02 ~ 2025-03-15: 73일
  - 17/366+73/365 ≈ 0.046448+0.2=0.246448
- 2024-12-31~2025-01-01 구간 1day에 적용되는 기간을 1/366 or 1/365 중 결정 이슈

- 명목금액 상환: 만기 일시상환만 가능
- 금리커브 생성 관련: RiskWatch는 Today + 날짜수 배열이 아닌, Today + Period 형태로 계산되는 것 같음
  - CurveTenor 인터페이스 표준: 날짜(20250315), 시간(1.1), 기간(365일), 테너(1Y), ...
- 공통 적용 인터페이스 관련
  - Enumeration 코드화(Glossary)
  - 날짜 입력 기준: 엑셀 숫자 입력, Char 타입(“20251002”), Integer(20251002)

평가방법 이슈

- Spread Interpolation 시, 첫 테너 bumping 방법

평가로직 예시 Script

```

#include <iostream>
#include "ql/termstructures/yield/piecewisezerospreadedtermstructure.hpp"

#include "bondTest.hpp"
#include "ql/termstructures/yield/zerocurve.hpp"
#include "ql/quotes/simplequote.hpp"
#include "ql/pricingengines/bond/discountingbondengine.hpp"
#include "ql/instruments/bonds/zerocouponbond.hpp"
#include "ql/time/calendars/southkorea.hpp"
#include "ql/instruments/bonds/fixedratebond.hpp"
#include "ql/time/schedule.hpp"
#include "ql/time/daycounters/actualactual.hpp"

using namespace QuantLib;

void ZeroBondTest() {

    Date asOfDate = Date(21, Apr, 2025);
    Settings::instance().evaluationDate() = asOfDate;
```

/ **Bond**

```
    girrDates.emplace_back(asOfDate);
    girrDates.emplace_back(asOfDate + Period(3, Months));
    girrDates.emplace_back(asOfDate + Period(6, Months));
    girrDates.emplace_back(asOfDate + Period(1, Years));
    girrDates.emplace_back(asOfDate + Period(2, Years));
    girrDates.emplace_back(asOfDate + Period(3, Years));
    girrDates.emplace_back(asOfDate + Period(5, Years));
    girrDates.emplace_back(asOfDate + Period(10, Years));
    girrDates.emplace_back(asOfDate + Period(15, Years));
    girrDates.emplace_back(asOfDate + Period(20, Years));
    girrDates.emplace_back(asOfDate + Period(30, Years));

    std::vector<Rate> girrRates;
    girrRates.emplace_back(0.01);
    girrRates.emplace_back(0.01);
    girrRates.emplace_back(0.01);
    girrRates.emplace_back(0.02);
    girrRates.emplace_back(0.01);
    girrRates.emplace_back(0.01);
    girrRates.emplace_back(0.01);
    girrRates.emplace_back(0.01);
    girrRates.emplace_back(0.01);
    girrRates.emplace_back(0.01);
    girrRates.emplace_back(0.01);

    DayCounter girrDayCounter = Actual365Fixed();
    Linear girrInterpolator = Linear();
    Compounding girrCompounding = Compounding::Continuous;
    Frequency girrFrequency = Frequency::Annual;

    ext::shared_ptr<YieldTermStructure> girrTermstructure = ext::make_shared<ZeroCurve>(g
                                                                                  girrInterpolator, girrCo
    RelinkableHandle<YieldTermStructure> girrCurve;
    girrCurve.linkTo(girrTermstructure);


    std::vector<Date> csrDates;
    csrDates.emplace_back(asOfDate);
    csrDates.emplace_back(asOfDate + Period(6, Months));
    csrDates.emplace_back(asOfDate + Period(1, Years));
    csrDates.emplace_back(asOfDate + Period(3, Years));
    csrDates.emplace_back(asOfDate + Period(5, Years));
    csrDates.emplace_back(asOfDate + Period(10, Years));

    std::vector<Handle<Quote>> csrSpreads;
    csrSpreads.emplace_back(ext::make_shared<SimpleQuote>(0.002));
    csrSpreads.emplace_back(ext::make_shared<SimpleQuote>(0.002));
```

```
    csrSpreads.emplace_back(ext::make_shared<SimpleQuote>(0.003));
    csrSpreads.emplace_back(ext::make_shared<SimpleQuote>(0.003));

    ext::shared_ptr<ZeroYieldStructure> discountingTermStructure =
            ext::make_shared<PiecewiseZeroSpreadedTermStructure>(girrCurve, csrSpreads, c
    RelinkableHandle<YieldTermStructure> discountingCurve;
    discountingCurve.linkTo(discountingTermStructure);

//    Date startDate = asOfDate;
//    Date endDate = asOfDate + Period(10, Years);
    auto bondEngine = ext::make_shared<DiscountingBondEngine>(discountingCurve);

    // Zero coupon bond
    Size settlementDays = 2;
    Real faceAmount = 10000;
//    ZeroCouponBond zeroCouponBond(
//            settlementDays,
//            SouthKorea(),
//            faceAmount,
//            Date(15,August,2033),
//            Following,
//            Real(116.92),
//            Date(15,August,2023));
//
//    zeroCouponBond.setPricingEngine(bondEngine);
//    Real npv = zeroCouponBond.NPV();
//    std::cout << "NPV: " << npv << std::endl;

    // Fixed 4.5% bond
    Schedule fixedBondSchedule(
            Date(15, May, 2017), Date(15,May,2027), Period(Annual),
            SouthKorea(), Unadjusted, Unadjusted, DateGeneration::Backward, false);

    FixedRateBond fixedRateBond(
            settlementDays,
            faceAmount,
            fixedBondSchedule,
            std::vector<Rate>(1, 0.045),
            ActualActual(ActualActual::Bond),
            ModifiedFollowing,
            100.0, Date(15, May, 2007));

    fixedRateBond.setPricingEngine(bondEngine);
    Real npv = fixedRateBond.NPV();
    std::cout << "NPV: " << npv << std::endl;

}
```

/ **Bond**

```

void callZeroBondTest() {
    // QuantLib 라이브러리 사용 예제
    const long evaluationDate = 45657;          // 2024-12-31
    const long settlementDays = 0;

    const long issueDate = 44175;
    const long maturityDate = 47827;

    const double notional = 6000000000.0;

    const double couponRate = 0.015;
    const int couponDayCounter = 5; //Actual/Actual(Bond)

    const long numberOfCpnSch = 12;
    const long paymentDates[] = {45818, 46001, 46183, 46366, 46548, 46731, 46916, 47098, 4
    const long realStartDates[] = {45636, 45818, 46001, 46183, 46366, 46548, 46731, 46916,
    const long realEndDates[] = { 45818, 46001, 46183, 46366, 46548, 46731, 46916, 47098,

    const long numberOfGirrTenors = 10;
    const long girrDates[] = {91, 183, 365, 730, 1095, 1825, 3650, 5475, 7300, 10950};
    const double girrRates[] = {0.0337, 0.0317, 0.0285, 0.0272, 0.0269, 0.0271, 0.0278, 0.
    const long girrDayCounter = 1; // Actual/365
    const long girrInterpolator = 1; // Linear
    const long girrCompounding = 1; // Continuous
    const long girrFrequency = 1; // Annual

    const double spreadOverYield = 0.001389;
    const int spreadOverYieldCompounding = 1; // Continuous
    const int spreadOverYieldDayCounter = 1; // Actual/365
    const long numberOfCsrTenors = 5;
    const long csrDates[] = {183, 365, 1095, 1825, 3650};
    const double csrRates[] = {0.0, 0.0, 0.0, 0.0005, 0.001};

    //printSettlementDate(date, settlementDays);
    ZeroBondTest(
            evaluationDate,
            settlementDays,
            issueDate,
            maturityDate,
            notional,
            couponRate,
            couponDayCounter,
            numberOfCpnSch,
            paymentDates,
```

```
            numberOfGirrTenors,
            girrDates,
            girrRates,
            girrDayCounter,
            girrInterpolator,
            girrCompounding,
            girrFrequency,
            spreadOverYield,
            spreadOverYieldCompounding,
            spreadOverYieldDayCounter,
            numberOfCsrTenors,
            csrDates,
            csrRates
    );
};
```

```
//
// Created by junwo on 2025-04-21.
//

#include <iostream>
#include "ql/termstructures/yield/piecewisezerospreadedtermstructure.hpp"

#include "bondTest.hpp"
#include "ql/termstructures/yield/zerocurve.hpp"
#include "ql/quotes/simplequote.hpp"
#include "ql/pricingengines/bond/discountingbondengine.hpp"
#include "ql/instruments/bonds/zerocouponbond.hpp"
#include "ql/time/calendars/southkorea.hpp"
#include "ql/instruments/bonds/fixedratebond.hpp"
#include "ql/time/schedule.hpp"
#include "ql/time/daycounters/actualactual.hpp"

using namespace QuantLib;

double ZeroBondTest(
        long evaluationDate,            // 평가일 (serial number, 예: 46164)
        long settlementDays,            // 결제일 offset (보통 2일)

        long issueDate,                 // 발행일
        long maturityDate,              // 만기일
        double notional,                // 채권 원금

        double couponRate,              // 쿠폰 이율
```

```
        int numberOfCoupons,            // 쿠폰 개수
        const long* paymentDates,       // 지급일 배열
        const long* realStartDates,     // 각 구간 시작일
        const long* realEndDates,       // 각 구간 종료일

        int numberOfGirrTenors,         // GIRR 만기 수
        const long* girrTenorDays,      // GIRR 만기 (startDate로부터의 일수)
        const double* girrRates,        // GIRR 금리
        int girrDayCounter,             // GIRR DayCounter (예: 1 = Actual/365)
        int girrInterpolator,           // 보간법 (예: 1 = Linear)
        int girrCompounding,            // 이자 계산 방식 (예: 1 = Continuous)
        int girrFrequency,              // 이자 빈도 (예: 1 = Annual)

        double spreadOverYield,         // 채권의 종목 Credit Spread
        int spreadOverYieldCompounding, // Continuous
        int spreadOverYieldDayCounter,  // Actual/365

        int numberOfCsrTenors,          // CSR 만기 수
        const long* csrTenorDays,       // CSR 만기 (startDate로부터의 일수)
        const double* csrSpreads        // CSR 스프레드 (금리 차이)

)
{
    std::cout.precision(15);
    Date asOfDate_ = Date(evaluationDate);
    Settings::instance().evaluationDate() = asOfDate_;
    Size settlementDays_ = settlementDays;
    Real notional_ = notional;
    std::vector<Rate> couponRate_ = std::vector<Rate>(1, couponRate);
    DayCounter couponDayCounter_ = ActualActual(ActualActual::ISDA); // TODO 변환 함수 적

    std::vector<Date> girrDates_;
    std::vector<Real> girrRates_;
    std::vector<Period> girrPeriod = {Period(3, Months), Period(6, Months), Period(1, Yea
                                      Period(3, Years), Period(5, Years), Period(10, Year
                                      Period(20, Years), Period(30, Years)};
    girrDates_.emplace_back(asOfDate_);
    girrRates_.emplace_back(girrRates[0]);
    for (Size dateNum = 0; dateNum < numberOfGirrTenors; ++dateNum) {
//        girrDates_.emplace_back(asOfDate_ + girrTenorDays[dateNum]);
        girrDates_.emplace_back(asOfDate_ + girrPeriod[dateNum]);
        girrRates_.emplace_back(girrRates[dateNum]+spreadOverYield);
    }

    // TODO 변환 함수 적용
    DayCounter girrDayCounter_ = Actual365Fixed();
    Linear girrInterpolator_ = Linear();
```

/ **Bond**

```

    ext::shared_ptr<YieldTermStructure> girrTermstructure = ext::make_shared<ZeroCurve>(g
                                                                                       g
    RelinkableHandle<YieldTermStructure> girrCurve;
    girrCurve.linkTo(girrTermstructure);

    double tmpSpreadOverYield = spreadOverYield;
    Compounding spreadOverYieldCompounding_ = Compounding::Continuous;
    DayCounter spreadOverYieldDayCounter_ = Actual365Fixed();  // Actual/365
    InterestRate tempRate(tmpSpreadOverYield, spreadOverYieldDayCounter_, spreadOverYield

    std::vector<Date> csrDates_;
    std::vector<Period> csrPeriod = {Period(6, Months), Period(1, Years), Period(3, Years
    csrDates_.emplace_back(asOfDate_);
    std::vector<Handle<Quote>> csrSpreads_;
    double spreadOverYield_ = tempRate.equivalentRate(girrCompounding_, girrFrequency_, g
    csrSpreads_.emplace_back(ext::make_shared<SimpleQuote>(csrSpreads[0]));
//    csrSpreads_.emplace_back(ext::make_shared<SimpleQuote>(csrSpreads[0]+spreadOverYiel
    for (Size dateNum = 0; dateNum < numberOfCsrTenors; ++dateNum) {
//        csrDates_.emplace_back(asOfDate_ + csrTenorDays[dateNum]);
        csrDates_.emplace_back(asOfDate_ + csrPeriod[dateNum]);
        spreadOverYield_ = tempRate.equivalentRate(girrCompounding_, girrFrequency_, girr
        csrSpreads_.emplace_back(ext::make_shared<SimpleQuote>(csrSpreads[dateNum]));
//        csrSpreads_.emplace_back(ext::make_shared<SimpleQuote>(csrSpreads[dateNum] + sp
    }
    ext::shared_ptr<ZeroYieldStructure> discountingTermStructure =
            ext::make_shared<PiecewiseZeroSpreadedTermStructure>(girrCurve, csrSpreads_,
    RelinkableHandle<YieldTermStructure> discountingCurve;
    discountingCurve.linkTo(discountingTermStructure);

    auto bondEngine = ext::make_shared<DiscountingBondEngine>(discountingCurve);

    std::vector<Date> couponSch_;
    couponSch_.emplace_back(realStartDates[0]);
    for (Size schNum = 0; schNum < numberOfCoupons; ++schNum) {
        couponSch_.emplace_back(realEndDates[schNum]);
    }
    Schedule fixedBondSchedule_(couponSch_);

    FixedRateBond fixedRateBond(
            settlementDays_,
            notional_,
            fixedBondSchedule_,
            couponRate_,
            couponDayCounter_,
            ModifiedFollowing,
            100.0);
```

/ **Bond**

```
//    //디버깅용 배열
//    const Leg& tmpLeg = fixedRateBond.cashflows();
//    std::vector<Real> tmpCf;
//    std::vector<DiscountFactor> tmpDf;
//    for (const auto& cf : tmpLeg) {
//        tmpCf.emplace_back(cf->amount());
//        tmpDf.emplace_back(discountingCurve->discount(cf->date()));
//    }

    Real npv = fixedRateBond.NPV();
    std::cout << "NPV: " << npv << std::endl;

    Real girrBump = 0.0001;
    std::vector<Real> disCountingGirr;
    for (Size bumpNum = 1; bumpNum < girrRates_.size(); ++bumpNum) {
        std::vector<Rate> bumpGirrRates = girrRates_;
        bumpGirrRates[bumpNum] += girrBump;
        ext::shared_ptr<YieldTermStructure> bumpGirrTermstructure = ext::make_shared<Zero

        RelinkableHandle<YieldTermStructure> bumpGirrCurve;
        bumpGirrCurve.linkTo(bumpGirrTermstructure);

        ext::shared_ptr<ZeroYieldStructure> bumpDiscountingTermStructure =
                ext::make_shared<PiecewiseZeroSpreadedTermStructure>(bumpGirrCurve, csrSp
        RelinkableHandle<YieldTermStructure> bumpDiscountingCurve;
        bumpDiscountingCurve.linkTo(bumpDiscountingTermStructure);
        bumpDiscountingCurve->enableExtrapolation();

        auto bumpBondEngine = ext::make_shared<DiscountingBondEngine>(bumpDiscountingCurv
        fixedRateBond.setPricingEngine(bumpBondEngine);
        Real tmpGirr = fixedRateBond.NPV() - npv;
        disCountingGirr.emplace_back(tmpGirr);
        std::cout << "Girr[" << bumpNum << "]:" << tmpGirr << std::endl;
    }


    Real csrBump = 0.0001;
    std::vector<Real> disCountingCsr;
    for (Size bumpNum = 1; bumpNum < csrSpreads_.size(); ++bumpNum) {
//        std::vector<Handle<Quote>> bumpCsrSpreads_ = csrSpreads_;
//        bumpCsrSpreads_[bumpNum] = Handle<Quote>(ext::make_shared<SimpleQuote>(csrSprea
        std::vector<Handle<Quote>> bumpCsrSpreads_;
        for (Size i = 0; i < csrSpreads_.size(); ++i) {
            Real bump = (i == bumpNum) ? csrBump : 0.0;
```

```
      bumpCsrSpreads_.emplace_back(ext::make_shared<SimpleQuote>(csrSpreads_[i]->va
//       std::cout << "bumpNum, i: " << bumpNum << ", " << bumpCsrSpreads_[i]->value
```

```
    ext::shared_ptr<ZeroYieldStructure> bumpDiscountingTermStructure =
        ext::make_shared<PiecewiseZeroSpreadedTermStructure>(girrCurve, bumpCsrSp
    RelinkableHandle<YieldTermStructure> bumpDiscountingCurve;
    bumpDiscountingCurve.linkTo(bumpDiscountingTermStructure);
    bumpDiscountingCurve->enableExtrapolation();

    auto bumpBondEngine = ext::make_shared<DiscountingBondEngine>(bumpDiscountingCurv
    fixedRateBond.setPricingEngine(bumpBondEngine);
    Real tmpCsr = fixedRateBond.NPV() - npv;
    disCountingCsr.emplace_back(tmpCsr);
    std::cout << "Csr[" << bumpNum << "]:" << tmpCsr << std::endl;
  }
  Real cleanPrice = fixedRateBond.cleanPrice() / 100.0 * notional;
  Real dirtyPrice = fixedRateBond.dirtyPrice() / 100.0 * notional;
  Real accruedInterest = fixedRateBond.accruedAmount() / 100.0 * notional;

  return npv;
}

std::vector<Real> calcGirrBondEngine(ext::shared_ptr<Bond> bond, Handle<YieldTermStructur

}
```

+ 레이블 추가

관련 콘텐츠

Structured Coupon
출시준비
다음과 유사하게

Quantlib Enum
출시준비
다음과 유사하게

Models
출시준비
다음과 유사하게

CallableVanillaSwap
출시준비
다음과 유사하게

Fx Swap

IR Swap

/ **Bond**

👍 👏 🎉 첫 번째로 반응을 추가해 보세요