DFX Finance

오익준, 이예범

목차

- DFX 주요 파트 변화
- Assimilator 개요
- Assimilator 코드 분석
- Curve 개요

DFX 주요 파트 변화

DFX Protocol V0.5

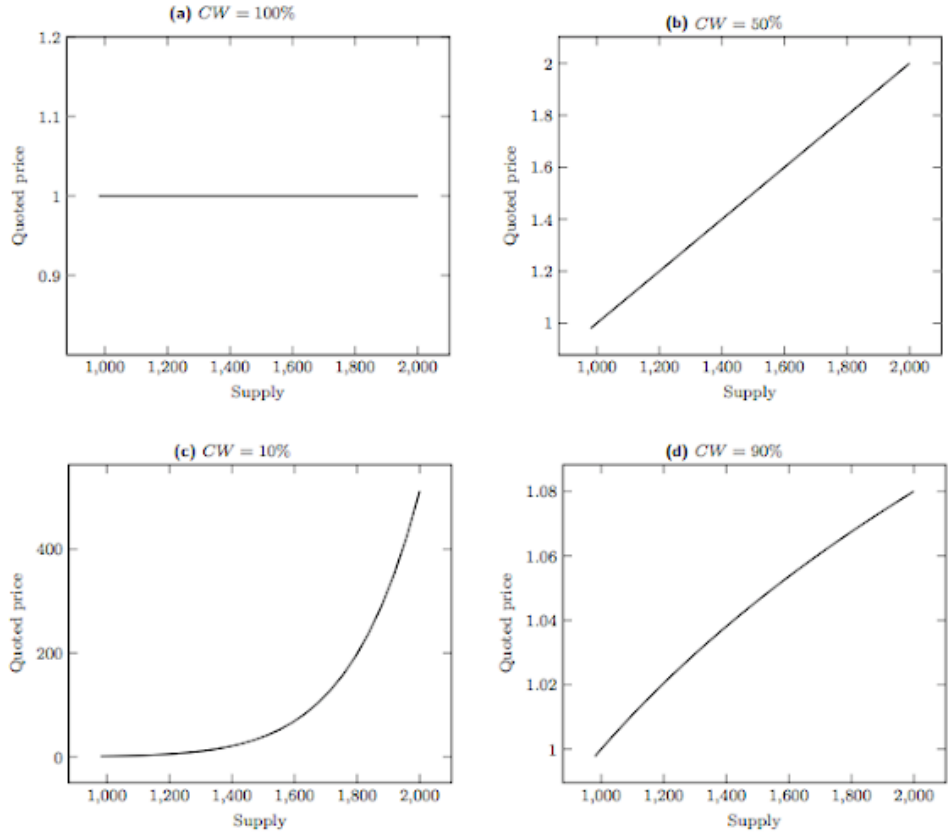
- shellprotocol@48dac1c 포크

- 두 개의 주요 파트
  - Assimilators - AMM(Automated Market Maker)이 서로 다른 가치의 한 쌍을 다룰 수 있게 하고, 각 통화에 대한 오라클 가격을 검색한다.

  - Curves - dynamic fees, halting boundaries와 함께 bonding curve의 사용자 정의 매개변수화를 가능하게 한다.

Bonding curve

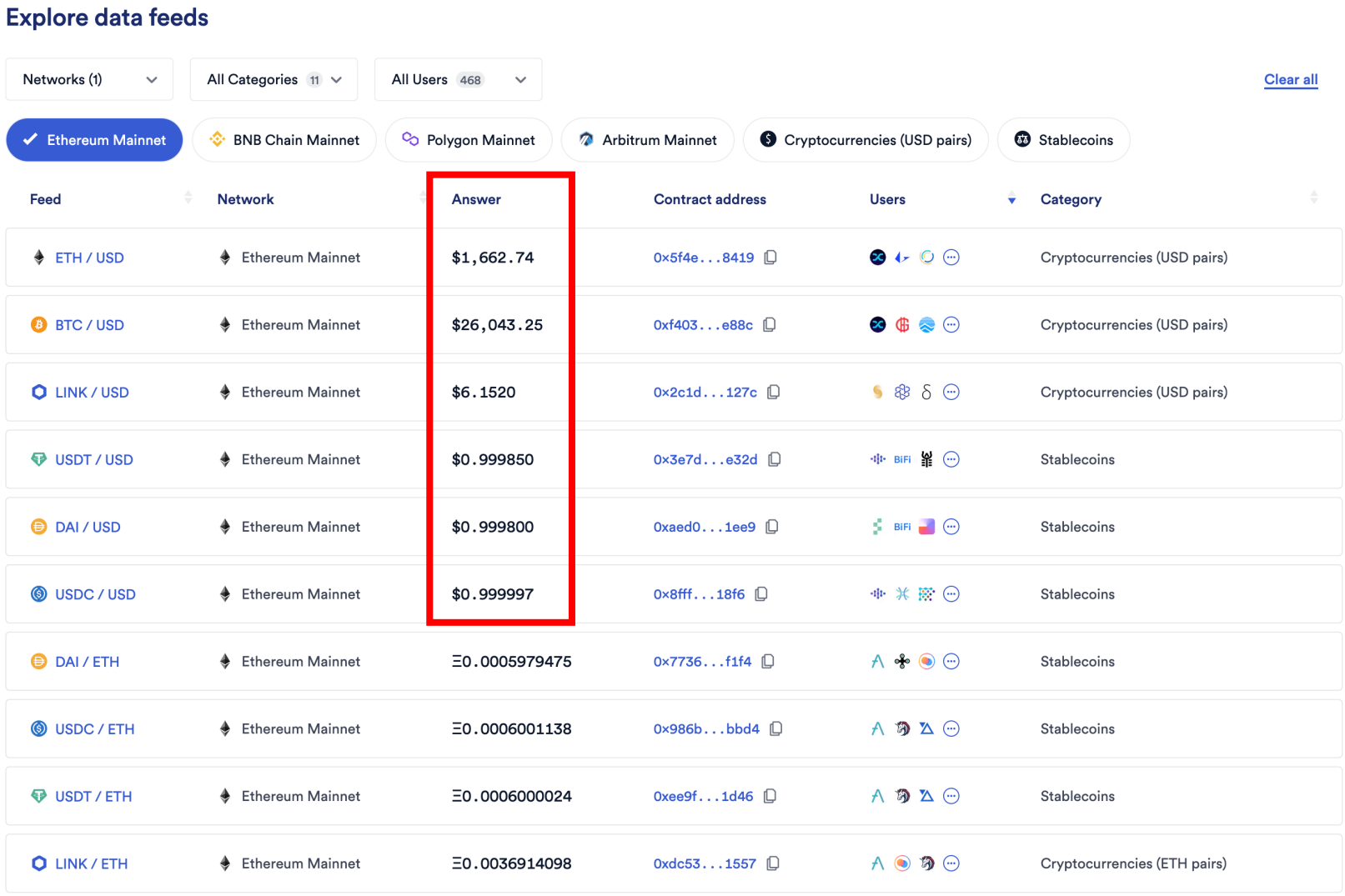
- 토큰의 가격이 정해진 곡선에 따라 결정되는 방식
- CW(Connector Weight) - 리저브 비율. 이 비율에 따라 그래프 모양이 바뀜
- Halting boundaries - 특정 상황에서 토큰 거래를 중단/일시 정지
- dynamic fees - 수수료가 시장 조건이나 특정 규칙에 따라 자동으로 조정



Assimilators

- 커브와 다른 DeFi 시스템(토큰)간의 미들웨어 역할
- 다른 스테이블코인에 대한 delegatecall 프록시 시스템 역할을 수행하여 풀이 상호작용적으로 균형을 유지하고 LP(Liquidity Provider)가 유동성을 제공할 수 있게 한다.
- 모든 토큰 가격을 USD에 기반한 누메레어(numeraire)로 변환한다. 누메레어는 서로 다른 가치의 한 쌍을 다룰 때 필요함
- -> Oracle price feeds에서 USD 기준으로 나타내기 때문

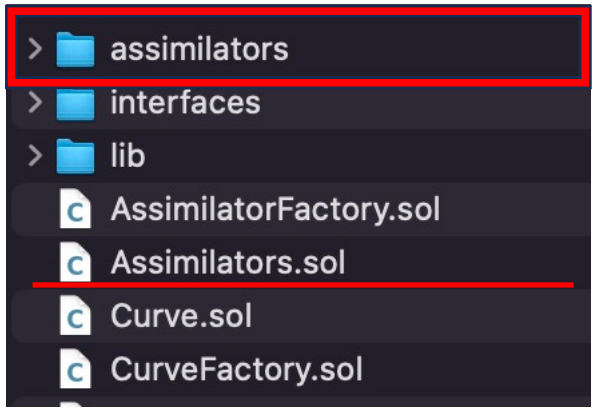
Oracle data feeds (chain.link)



Assimilators.sol

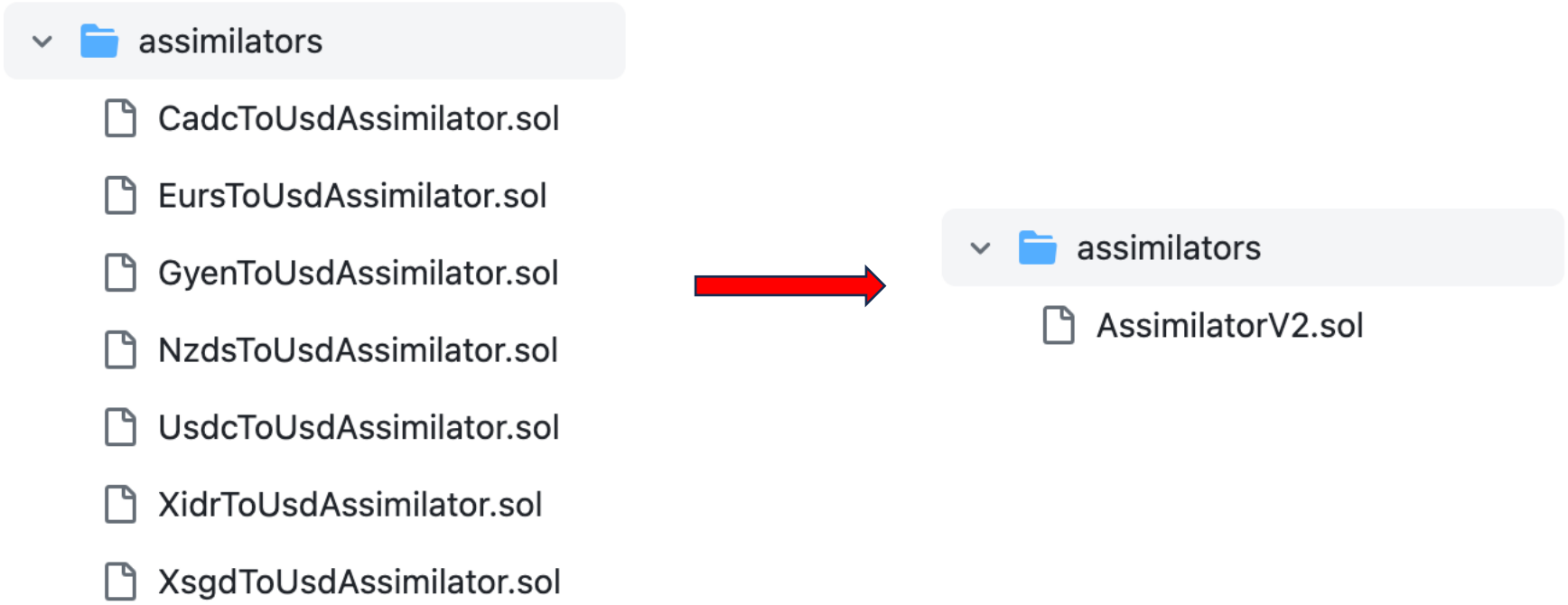
- 이 파일에 있는 모든 메소드는 단지 런타임시에 관련된 토큰에 대해 delegate execution을 위한 internal function

- assimilator architecture의 중요한 부분은 assimilators/ 폴더에 있다.



DFX Protocol V2 변화

- 각 토큰을 담당하는 어심미레이터가 하나의 파일로 통합



ex) EursToUsdAssimilator.sol

```
contract EursToUsdAssimilator is IAssimilator {
    using ABDKMath64x64 for int128;
    using ABDKMath64x64 for uint256;

    using SafeMath for uint256;

    IERC20 private constant usdc = IERC20(0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48);

    IOracle private constant oracle = IOracle(0xb49f677943BC038e9857d61E7d053CaA2C1734C1);
    IERC20 private constant eurs = IERC20(0xdB25f211AB05b1c97D595516F45794528a807ad8);
```

```
// takes raw eurs amount, transfers it in, calculates corresponding numeraire amount and returns it
function intakeRawAndGetBalance(uint256 _amount) external override returns (int128 amount_, int128 balance_) {
    bool _transferSuccess = eurs.transferFrom(msg.sender, address(this), _amount);

    require(_transferSuccess, "Curve/EURS-transfer-from-failed");

    uint256 _balance = eurs.balanceOf(address(this));

    uint256 _rate = getRate();

    balance_ = ((_balance * _rate) / 1e8).divu(1e2);

    amount_ = ((_amount * _rate) / 1e8).divu(1e2);
}
```

AssimilatorV2.sol

```
contract AssimilatorV2 is IAssimilator {
    using ABDKMath64x64 for int128;
    using ABDKMath64x64 for uint256;

    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IERC20 private constant usdc = IERC20(0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48);

    IOracle private immutable oracle;
    IERC20 private immutable token;
    uint256 private immutable oracleDecimals;
    uint256 private immutable tokenDecimals;
```

```
// takes raw eurs amount, transfers it in, calculates corresponding numeraire amount and returns it
function intakeRawAndGetBalance(uint256 _amount) external override returns (int128 amount_, int128 balance_) {
    token.safeTransferFrom(msg.sender, address(this), _amount);

    uint256 _balance = token.balanceOf(address(this));

    uint256 _rate = getRate();

    balance_ = ((_balance * _rate) / 10**oracleDecimals).divu(10**tokenDecimals);

    amount_ = ((_amount * _rate) / 10**oracleDecimals).divu(10**tokenDecimals);
}
```

Assimilator 코드 분석

Assmilator 의 구조

	Type	
IAssmilatorFactory.sol	Interface	
IAssmilators.sol	Interface	
AssmilatorFactory.sol	Contract	Assmilator 공장
Assmilators.sol	Library	Assmilator 내장 함수 (주로 Curve 및 외 파일들에서 사용)
		delegatecall 을 통해 사용
AssmilatorV2.sol	Contract	Assmilator (실제 배포)

Assmilator 의 구조

AssmilatorFactory.sol (contract)

newAssimilator 함수로 생성

AssmilatorV2.sol (contract)

Assmilators.sol (Library)

하단 파일에서
Assimilators 의 함수를 사용

Orchestrator.sol curve 세팅

ProprotionalLiquidity.sol 유동성 풀

Swap.sol 토큰 교환

Assmilator 의 구조

- AssmilatorFactory.sol (contract)

newAssimilator 함수로 생성

- AssmilatorV2.sol (contract)

```
// 새로운 Assimilator를 생성하는 함수입니다.
function newAssimilator(
    IOracle _oracle,
    address _token,
    uint256 _tokenDecimals
) external override onlyCurveFactory returns (AssimilatorV2) {
    // 주어진 토큰 주소를 인코딩하고 해시하여 Assimilator의 고유 ID를 생성
    bytes32 assimilatorID = keccak256(abi.encode(_token));
    // 이미 해당 ID로 생성된 Assimilator가 있는지 확인하고 존재하면 에러 반환
    if (address(assimilators[assimilatorID]) != address(0))
        revert("AssimilatorFactory/oracle-stablecoin-pair-already-exists");

    // 새로운 Assimilator 인스턴스를 생성 -> Oracle, 토큰주소, 토큰 decimal, Oracle decimal을 전달
    AssimilatorV2 assimilator = new AssimilatorV2(_oracle, _token, _tokenDecimals, IOracle(_oracle).decimals());
    // 아까 생성한 Assimilator의 주소에 새로 생성한 Assimilator를 저장
    assimilators[assimilatorID] = assimilator;
    emit NewAssimilator(msg.sender, assimilatorID, address(assimilator), address(_oracle), _token);
    return assimilator;
}
```

```
contract AssimilatorV2 is IAssimilator {
    using ABDKMath64x64 for int128;
    using ABDKMath64x64 for uint256;

    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    // IERC20의 토큰(USDC)
    IERC20 private constant usdc = IERC20(0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48);

    IOracle private immutable oracle;        // getRate()에 사용하는 oracle.latestRoundData를 위해서 사용하였다.
    IERC20 private immutable token;          // Transferfrom, balanceof 등 토큰의 함수를 적용하기 위해 사용하였다.
    uint256 private immutable oracleDecimals;// oracleDecimals - 오라클 서비스에서 제공되는 데이터의 정밀도나 분할성을 나타내는 값
    uint256 private immutable tokenDecimals; // tokenDecimals - 토큰의 최소 분할 단위 1토큰은 1e18의 가장 작은단위

    // solhint-disable-next-line
    constructor(
        IOracle _oracle,
        address _token, // Assimilator에서 USDC와 바꾸길 원하는 토큰을 Factory의 new Assimilator에서 인자로 넣음
        uint256 _tokenDecimals,
        uint256 _oracleDecimals
    ) {
        oracle = _oracle;
        token = IERC20(_token);
        oracleDecimals = _oracleDecimals;
        tokenDecimals = _tokenDecimals;
    }
    // IOracle private constant oracle = IOracle(0xb49f677943BC038e9857d61E7d053CaA2C1734C1);
    // IERC20 private constant eurs = IERC20(0xdB25f211AB05b1c97D595516F45794528a807ad8);
    // 기존 V1에서는 이렇게 사용했다. 그 말은 즉슨 생성자에 주소를 넣고 해당 토큰의 주소를 넣으면 해당 토큰에 대한 동화기 생성하여 사용하는 느낌
```

실제 내부 프로토콜 프로젝트엔 없음
아마 토큰 처음에 생성할 때,
해당 토큰의 Assimilator로 진행하는 것 같음
(실제 배포 – write contract에 있음)

Assmilator 의 구조

```
function getFee(Storage.Curve storage curve) private view returns (int128 fee_) {
    int128 _gLiq;

    // Always pairs
    int128[] memory _bals = new int128[](2);

    // for문 무조건 두번만 돌음
    // Q. 왜 getFee에서 _bals는 두개일까? 왜 Pairs일까?
    // -> 여기서 두개라면 모든 curve의 assimilator는 2개이다. 라는 정의가 나온다.
    for (uint256 i = 0; i < _bals.length; i++) {
        // viewNumeraireBalance : 주어진 주소의 토큰 잔액에서 토큰을 usdc로 바꿔서 보여줌
        // curve.assets : assimilators를 담음
        int128 _bal = Assimilators.viewNumeraireBalance(curve.assets[i].addr);

        _bals[i] = _bal;

        _gLiq += _bal;
    }

    // _gLiq : 총 2개의 토큰의 누적 금액 (총 자산)
    // _bals : 각 토큰의 잔액
    fee_ = CurveMath.calculateFee(_gLiq, _bals, curve.beta, curve.delta, curve.weights);
}
```

```
// 사용자가 풀에 유동성을 추가할 때 사용 (유동성이 없으면 오라클이 비율 설정 / 유동성이 있으면 기존의 풀 비율 따름)
// 인자 - _deposit : 사용자가 풀에 추가하고자 하는 ETH or 기본 토큰의 금액
// 반환값 - curves_ : 새로 발행된 풀 토큰의 수량
// 반환값 - deposits_ : 풀에 각 자산별로 추가될 금액의 배열
function proportionalDeposit(Storage.Curve storage curve, uint256 _deposit)
    external
    returns (uint256 curves_, uint256[] memory)
{
    int128 __deposit = _deposit.divu(1e18);

    uint256 _length = curve.assets.length;

    uint256[] memory deposits_ = new uint256[](_length);

    (int128 _oGLiq, int128[] memory _oBals) = getGrossLiquidityAndBalancesForDeposit(curve);

    // Needed to calculate liquidity invariant
    // (int128 _oGLiqProp, int128[] memory _oBalsProp) = getGrossLiquidityAndBalances(curve);

    // No liquidity, oracle sets the ratio
    if (_oGLiq == 0) {
        for (uint256 i = 0; i < _length; i++) {
            // Variable here to avoid stack-too-deep errors
            int128 _d = __deposit.mul(curve.weights[i]);
            deposits_[i] = Assimilators.intakeNumeraire(curve.assets[i].addr, _d.add(ONE_WEI));
        }
    } else {
        // We already have an existing pool ratio
        // which must be respected
        int128 _multiplier = __deposit.div(_oGLiq);
```

Assmilators.sol (Library)

하단 파일에서
Assimilators 의 함수를 사용

Orchestrator.sol — curve 세팅

ProprotionalLiquidity.sol — 유동성 풀

Swap.sol — 토큰 교환

AssmilatorV2 의 구조 (실제 동작되는 함수)

환율 getRate

입금
- intakeRawAndGetBalance
- intakeRaw
- IntakeNumeraire
- IntakeNumeraireLPRatio

출금
- outputRawAndGetBalance
- outputRaw
- outputNumeraire

Q. 왜 Assimilator가 직접 동작이 아닌 AssimilatorV2에서 직접 동작하는가?

A. Assmilator가 내부에서 사용되지만,
실제적으로는 delegatecall로 해당 컨트랙트를 호출하기 때문에
그래서 실제 동작되는 AssimilatorV2의 함수를 알아보겠습니다

AssmilatorV2 의 구조 (실제 동작되는 함수)

환율 getRate

```
// base token - Quote Token(USDC) 의 최근의 환율(?) 을 가져옴
function getRate() public view override returns (uint256) {
    (, int256 price, , , ) = oracle.latestRoundData();
    return uint256(price);
}
```

입금
- intakeRawAndGetBalance
- intakeRaw
- IntakeNumeraire
- IntakeNumeraireLPRatio

출금
- outputRawAndGetBalance
- outputRaw
- outputNumeraire

AssmilatorV2 의 구조 (실제 동작되는 함수)

환율 getRate

입금
- intakeRawAndGetBalance
- intakeRaw
- IntakeNumeraire
- IntakeNumeraireLPRatio

출금
- outputRawAndGetBalance
- outputRaw
- outputNumeraire

```
// 토큰을 가져와서 보내고, USDC의 금액으로 계산하여 반환한다. (토큰의 balance도 반환된다.)
// takes raw eurs amount, transfers it in, calculates corresponding numeraire amount and returns it
function intakeRawAndGetBalance(uint256 _amount) external override returns (int128 amount_, int128 balance_) {

    // approve 존재해야하는데 test나 실제 사용할 때는 꼭 필요

    // 토큰 보내는 sender -> assmilator 에게 보낼 권한을 줌
    token.safeTransferFrom(msg.sender, address(this), _amount);

    // address(this)가 갖고있는 일반 토큰의 잔액
    uint256 _balance = token.balanceOf(address(this));

    // 환율 가져옴
    uint256 _rate = getRate();

    // divu - div의 unsigned (부호가 없는)
    balance_ = ((_balance * _rate) / 10**oracleDecimals).divu(10**tokenDecimals);

    amount_ = ((_amount * _rate) / 10**oracleDecimals).divu(10**tokenDecimals);
}
```

```
// 토큰을 가져와서 보내고, USDC의 금액으로 계산하여 반환한다.
// takes raw eurs amount, transfers it in, calculates corresponding numeraire amount and returns it
function intakeRaw(uint256 _amount) external override returns (int128 amount_) {
    token.safeTransferFrom(msg.sender, address(this), _amount);

    uint256 _rate = getRate();

    amount_ = ((_amount * _rate) / 10**oracleDecimals).divu(10**tokenDecimals);
}
```

AssmilatorV2 의 구조 (실제 동작되는 함수)

환율 getRate

입금
- intakeRawAndGetBalance
- intakeRaw
- IntakeNumeraire
- IntakeNumeraireLPRatio

출금
- outputRawAndGetBalance
- outputRaw
- outputNumeraire

```
// USDC를 가져오고 토큰의 총 금액으로 계산하고 해당 금액을 반환한다.
// takes a numeraire amount, calculates the raw amount of eurs, transfers it in and returns the corresponding raw amount
function intakeNumeraire(int128 _amount) external override returns (uint256 amount_) {
    uint256 _rate = getRate();

    amount_ = (_amount.mulu(10**tokenDecimals) * 10**oracleDecimals) / _rate;

    token.safeTransferFrom(msg.sender, address(this), amount_);
}
```

```
// intakeNumeraire는 오라클 서비스에서 직접 환율을 받아서 사용한다.
// 반면에 LPRatio는 오라클이 없을때 직접 비율을 구해서 사용된다.

// LP : Liquidity Provider, LP는 AMM 플랫폼에 자금을 제공하는 개인 또는 기관이다.
// mul : 곱하기 / 1e18 : 10^18
// takes a numeraire amount, calculates the raw amount of eurs, transfers it i
function intakeNumeraireLPRatio(
    uint256 _baseWeight,
    uint256 _quoteWeight,
    address _addr,
    int128 _amount
) external override returns (uint256 amount_) {
    uint256 _tokenBal = token.balanceOf(_addr);

    if (_tokenBal <= 0) return 0;

    // 가중치에 따라 token과 usdc 잔액 조정
    _tokenBal = _tokenBal.mul(1e18).div(_baseWeight);

    uint256 _usdcBal = usdc.balanceOf(_addr).mul(1e18).div(_quoteWeight);

    // usdc,토큰 잔액의 비율을 즉, 환율을 계산한다.
    // Rate is in 1e6
    uint256 _rate = _usdcBal.mul(10**tokenDecimals).div(_tokenBal);

    // amount를 현재 환율에 맞게 변환하여 전송한다.
    amount_ = (_amount.mulu(10**tokenDecimals) * 1e6) / _rate;

    token.safeTransferFrom(msg.sender, address(this), amount_);
}
```

AssmilatorV2 의 구조 (실제 동작되는 함수)

환율 getRate

입금
- intakeRawAndGetBalance
- intakeRaw
- IntakeNumeraire
- IntakeNumeraireLPRatio

출금
- outputRawAndGetBalance
- outputRaw
- outputNumeraire

```
// 토큰 출금 -> 해당 금액의 numeraire값 반환 , 전송하고 남은 잔액 반환
// takes a raw amount of eurs and transfers it out, returns numeraire value of the raw amount
function outputRawAndGetBalance(address _dst, uint256 _amount)
    external
    override
    returns (int128 amount_, int128 balance_)
{
    uint256 _rate = getRate();

    // 보낼 토큰의 총 금액
    uint256 _tokenAmount = ((_amount) * _rate) / 10**oracleDecimals;

    // 토큰 전송함
    token.safeTransfer(_dst, _tokenAmount);

    // 전송한 토큰의 남은 잔액
    uint256 _balance = token.balanceOf(address(this));

    // 토큰 전송한 값을 amount_로 바꿈 -> 아마 전송된 토큰의 numeraire(usdc 값)
    amount_ = _tokenAmount.divu(10**tokenDecimals);

    // 전송하고 토큰의 남은 잔액
    balance_ = ((_balance * _rate) / 10**oracleDecimals).divu(10**tokenDecimals);
}
```

```
// 토큰 출금 -> 해당 금액의 numeraire값 반환
// takes a raw amount of eurs and transfers it out, returns numeraire value of the raw amount
function outputRaw(address _dst, uint256 _amount) external override returns (int128 amount_) {
    uint256 _rate = getRate();

    uint256 _tokenAmount = (_amount * _rate) / 10**oracleDecimals;

    token.safeTransfer(_dst, _tokenAmount);

    amount_ = _tokenAmount.divu(10**tokenDecimals);
}
```

```
// numeraire의 토큰 금액 출금 -> 해당 금액의 토큰 금액 반환
// takes a numeraire value of eurs, figures out the raw amount, transfers raw amount out, and returns raw amount
function outputNumeraire(address _dst, int128 _amount) external override returns (uint256 amount_) {
    uint256 _rate = getRate();

    amount_ = (_amount.mulu(10**tokenDecimals) * 10**oracleDecimals) / _rate;

    token.safeTransfer(_dst, amount_);
}
```

Assimilator 사용 (전체 Flow)

Curve.sol (contract)

예금

```
/// @notice deposit into the pool with no slippage from the numeraire assets
/// 풀이 지원하는 numeraire 자산으로부터 slippage 없이 입금
/// @param  _deposit the full amount you want to deposit into the pool which
///                  the numeraire assets of the pool
/// @return (the amount of curves you receive in return for your deposit,
///          the amount deposited for each numeraire)
function deposit(uint256 _deposit, uint256 _deadline)
    external
    deadline(_deadline)          /// modifier : 주어진 마감 시간전에 거래가 이루어져야
    globallyTransactable         /// 함수가 global하게 호출될 수 있는지 확인
    transactable                 /// 함수가 거래를 할 수 있는지 확인
    nonReentrant                 /// 재진입 공격을 방지하기 위해 함수에 재귀호출을 방지
    noDelegateCall               /// 함수가 delegate call을 통해 호출되지 않도록 한다.
    notInWhitelistingStage
    isNotEmergency               /// 비상 상황이 아닌지 확인ㄴㄴ
    returns (uint256, uint256[] memory)
{
    // (curvesMinted , deposits )
    return ProportionalLiquidity.proportionalDeposit(curve, _deposit);
}
```

ProportionalLiquidity.sol (Library)

```
// 사용자가 풀에 유동성을 추가할 때 사용 (유동성이 없으면 오라클이 비율 설정 / 유동성이 있으면 기존의 풀 비율 따름)
// 인자 - _deposit : 사용자가 풀에 추가하고자 하는 ETH or 기본 토큰의 금액
// 반환값 - curves_ : 새로 발행된 풀 토큰의 수량
// 반환값 - deposits_ : 풀에 각 자산별로 추가될 금액의 배열
function proportionalDeposit(Storage.Curve storage curve, uint256 _deposit)
    external
    returns (uint256 curves_, uint256[] memory)
{
    int128 __deposit = _deposit.divu(1e18);

    uint256 _length = curve.assets.length;

    uint256[] memory deposits_ = new uint256[](_length);

    (int128 _oGLiq, int128[] memory _oBals) = getGrossLiquidityAndBalancesForDeposit(curve);

    // Needed to calculate liquidity invariant
    // (int128 _oGLiqProp, int128[] memory _oBalsProp) = getGrossLiquidityAndBalances(curve);

    // No liquidity, oracle sets the ratio
    if (_oGLiq == 0) {
        for (uint256 i = 0; i < _length; i++) {
            // Variable here to avoid stack-too-deep errors
            int128 _d = __deposit.mul(curve.weights[i]);
            deposits_[i] = Assimilators.intakeNumeraire(curve.assets[i].addr, _d.add(ONE_WEI));
        }
    } else {
        // We already have an existing pool ratio
        // which must be respected
        int128 _multiplier = __deposit.div(_oGLiq);

        uint256 _baseWeight = curve.weights[0].mulu(1e18);
        uint256 _quoteWeight = curve.weights[1].mulu(1e18);

        for (uint256 i = 0; i < _length; i++) {
            deposits_[i] = Assimilators.intakeNumeraireLPRatio(
                curve.assets[i].addr,
                _baseWeight,
                _quoteWeight,
                _oBals[i].mul(_multiplier).add(ONE_WEI)
            );
        }
    }
```

Assimilators.sol (Library)

```
function intakeNumeraire(address _assim, int128 _amt) internal returns (uint256 amt_) {
    bytes memory data = abi.encodeWithSelector(iAsmltr.intakeNumeraire.selector, _amt);

    amt_ = abi.decode(delegate(_assim, data), (uint256));
}
```

AssimilatorV2.sol -> 실제 배포된

```
// USDC를 가져오고 토큰의 총 금액으로 계산하고 해당 금액을 반환한다.
// takes a numeraire amount, calculates the raw amount of eurs, transfers it in and ret
function intakeNumeraire(int128 _amount) external override returns (uint256 amount_) {
    uint256 _rate = getRate();

    amount_ = (_amount.mulu(10**tokenDecimals) * 10**oracleDecimals) / _rate;

    token.safeTransferFrom(msg.sender, address(this), amount_);
}
```

Assmilators 사용 (DELEGATE 호출O)

Q. 어떻게 인터페이스로 실제 배포된 함수의 선택자를 가져올 수 있을까?

A. ABI(Application Binary Interface) 를 사용해서 이더리움 스마트 컨트랙트와 상호작용한다.

Assimilators.sol

```
IAssimilator public constant iAsmltr = IAssimilator(address(0));
```

```
function intakeNumeraire(address _assim, int128 _amt) internal returns (uint256 amt_) {
    bytes memory data = abi.encodeWithSelector(iAsmltr.intakeNumeraire.selector, _amt);

    amt_ = abi.decode(delegate(_assim, data), (uint256));
}
```

abi의 encodeWithSelector를 사용
data = intakeRaw 함수선택자(4byte) + _amt (매개변수)

```
// 인터페이스 IAssimilator를 가져오고 그거에 해당하는 변수 이름을 iAsmltr이라고함. 거기에 IAssmilaotr(0)임
// => Iassmilator 타입의 iAsmltr 변수를 선언하고 이를 IAssmilaltor로 초기화한다.*****
IAssimilator public constant iAsmltr = IAssimilator(address(0));

// _callee : contract 주소 / _data : 호출하려는 함수와 그 매개변수를 인코딩한 데이터
// internal : 컨트랙트 내부에서만 사용하는 뜻
function delegate(address _callee, bytes memory _data) internal returns (bytes memory) {
    // callee가 contract 아니라면 에러메시지 *****
    require(_callee.isContract(), "Assimilators/callee-is-not-a-contract");

    // solhint-disable-next-line
    // delegatecall : 호출자의 컨텍스트에서 다른 컨트랙트의 함수를 호출함.
    // _callee : 호출하려는 컨트랙트의 주소, / _data : 호출하려는 함수와 그 매개변수 인코딩
    // return value : 성공여부, 리턴값 *****
    (bool _success, bytes memory returnData_) = _callee.delegatecall(_data); //이 함수는 어디서 나온걸까..

    // solhint-disable-next-line
    // 호출이 실패 : _success = false 이면, 반환데이터를 사용하여 트랜잭션을 되돌림 *****
    assembly {
        if eq(_success, 0) {
            revert(add(returnData_, 0x20), returndatasize())
        }
    }

    return returnData_;
}
```

AssimilatorV2.sol -> 실제 배포된 AssmilatorV2

```
contract AssimilatorV2 is IAssimilator {
```

```
// USDC를 가져오고 토큰의 총 금액으로 계산하고 해당 금액을 반환한다.
// takes a numeraire amount, calculates the raw amount of eurs, transfers it in and ret
function intakeNumeraire(int128 _amount) external override returns (uint256 amount_) {
    uint256 _rate = getRate();

    amount_ = (_amount.mulu(10**tokenDecimals) * 10**oracleDecimals) / _rate;

    token.safeTransferFrom(msg.sender, address(this), amount_);
}
```

Assmilators 사용 (DELEGATE호출X)

View나 getRate처럼
컨트랙트와 상호작용이 필요하지 않으면
abi와 delegate를 사용하지 않음

AssmilatorV2.sol

```
// numeraire(usdc)를 입력으로 받아서 해당 값을 토큰 금액으로 변환해서 보여준다. (oracle 사용)
// takes a numeraire amount and returns the raw amount
function viewRawAmount(int128 _amount) external view override returns (uint256 amount_) {
    uint256 _rate = getRate();

    amount_ = (_amount.mulu(10**tokenDecimals) * 10**oracleDecimals) / _rate;
}
```

```
// base token - Quote Token(USDC) 의 최근의 환율(?) 을 가져옴
function getRate() public view override returns (uint256) {
    (, int256 price, , , ) = oracle.latestRoundData();
    return uint256(price);
}
```

Assmilators.sol

```
function viewRawAmount(address _assim, int128 _amt) internal view retur
    amount_ = IAssimilator(_assim).viewRawAmount(_amt);
}
```

```
function getRate(address _assim) internal view returns (uint256 amount_) {
    // IAssimlator에 주소넣고 그거에 해당하는 함수 가져옴
    // 추측 : IAssmilator local에서는 주소 파라미터가 없는데 아마 온체인에 있는 녀석일 것 같음.
    amount_ = IAssimilator(_assim).getRate();
}
```

Curve 개요

Curve(Shell)

- 유동성 풀에 LP가 stablecoin을 예금하면 발행되는 토큰(지분) 지칭

- 이 토큰을 상환하면 LP는 자신의 지분만큼 stablecoin을 받는다.

- 서로 다른 토큰끼리 Swap에 용이

Curve(Shell) parameter

- AMM(Automated Market Maker) / Boding Curve

1. Ideal weights for reserve stablecoins (w)
2. Depth of 1:1 exchange (β)
3. Price slippage (elasticity) when not 1:1 exchange (Δ)
4. Maximum and minimum allocation of each reserve (α)
5. Fees (ϵ, λ)

CurveMath.sol

- 수수료(fee) 계산하는 함수

- Halt thresholds 정의:
  유동성 풀은 토큰의 현재 시장 가격에 대해 직접적으로 모름.
  따라서 만약 풀 내의 stablecoin이 peg를 잃어도 풀은 그 코인이 가치 없다는 것을 모름.
  가치 없는 토큰을 계속 수용하여 다른 토큰을 drain하면 안되기 때문에 각 토큰에 최소 및 최대 할당량을 가지게 풀을 설정해야 함.

*peg: 가치를 일정한 수준으로 고정시키는 것

Curve.sol/Flashloan 재진입 보완

```
function flash(
    address recipient,
    uint256 amount0,
    uint256 amount1,
    bytes calldata data
) external transactable noDelegateCall isNotEmergency {
    uint256 fee = curve.epsilon.mulu(1e18);

    require(IERC20(derivatives[0]).balanceOf(address(this)) > 0, 'Curve/token0-zero-liquidity-depth');
    require(IERC20(derivatives[1]).balanceOf(address(this)) > 0, 'Curve/token1-zero-liquidity-depth');

    uint256 fee0 = FullMath.mulDivRoundingUp(amount0, fee, 1e18);
    uint256 fee1 = FullMath.mulDivRoundingUp(amount1, fee, 1e18);
    uint256 balance0Before = IERC20(derivatives[0]).balanceOf(address(this));
    uint256 balance1Before = IERC20(derivatives[1]).balanceOf(address(this));

    if (amount0 > 0) IERC20(derivatives[0]).safeTransfer(recipient, amount0);
    if (amount1 > 0) IERC20(derivatives[1]).safeTransfer(recipient, amount1);

    IFlashCallback(msg.sender).flashCallback(fee0, fee1, data);

    uint256 balance0After = IERC20(derivatives[0]).balanceOf(address(this));
    uint256 balance1After = IERC20(derivatives[1]).balanceOf(address(this));

    require(balance0Before.add(fee0) <= balance0After, 'Curve/insufficient-token0-returned');
    require(balance1Before.add(fee1) <= balance1After, 'Curve/insufficient-token1-returned');

    // sub is safe because we know balanceAfter is gt balanceBefore by at least fee
    uint256 paid0 = balance0After - balance0Before;
    uint256 paid1 = balance1After - balance1Before;

    IERC20(derivatives[0]).safeTransfer(owner, paid0);
    IERC20(derivatives[1]).safeTransfer(owner, paid1);

    emit Flash(msg.sender, recipient, amount0, amount1, paid0, paid1);
}
```

@ff0857c

```
modifier nonReentrant() {
    require(notEntered, "Curve/re-entered");
    notEntered = false;
    _;
    notEntered = true;
}
```

```
function flash(
    address recipient,
    uint256 amount0,
    uint256 amount1,
    bytes calldata data
) external nonReentrant noDelegateCall transactable isNotEmergency {
    uint256 fee = curve.epsilon.mulu(1e18);

    require(IERC20(derivatives[0]).balanceOf(address(this)) > 0, 'Curve/token0-zero-liquidity-depth');
    require(IERC20(derivatives[1]).balanceOf(address(this)) > 0, 'Curve/token1-zero-liquidity-depth');

    uint256 fee0 = FullMath.mulDivRoundingUp(amount0, fee, 1e18);
    uint256 fee1 = FullMath.mulDivRoundingUp(amount1, fee, 1e18);

    uint256 balance0Before = IERC20(derivatives[0]).balanceOf(address(this));
    uint256 balance1Before = IERC20(derivatives[1]).balanceOf(address(this));

    if (amount0 > 0) IERC20(derivatives[0]).safeTransfer(recipient, amount0);
    if (amount1 > 0) IERC20(derivatives[1]).safeTransfer(recipient, amount1);

    IFlashCallback(msg.sender).flashCallback(fee0, fee1, data);

    uint256 balance0After = IERC20(derivatives[0]).balanceOf(address(this));
    uint256 balance1After = IERC20(derivatives[1]).balanceOf(address(this));

    require(balance0Before.add(fee0) <= balance0After, 'Curve/insufficient-token0-returned');
    require(balance1Before.add(fee1) <= balance1After, 'Curve/insufficient-token1-returned');

    // sub is safe because we know balanceAfter is gt balanceBefore by at least fee
    uint256 paid0 = balance0After - balance0Before;
    uint256 paid1 = balance1After - balance1Before;

    IERC20(derivatives[0]).safeTransfer(owner, paid0);
    IERC20(derivatives[1]).safeTransfer(owner, paid1);

    emit Flash(msg.sender, recipient, amount0, amount1, paid0, paid1);
}
```

@f8fdc14

참조

- https://github.com/dfx-finance/protocol-v1-deprecated/tree/main
- https://github.com/cowri/shell-solidity-v1/tree/48dac1c1a18e2da292b0468577b9e6cbdb3786a4
- https://consensys.io/diligence/audits/2020/06/shell-protocol/shell-protocol-audit-2020-06.pdf
- https://github.com/cowri/shell-solidity-v1/blob/master/Shell_White_Paper_v1.0.pdf [쉘 프로토콜 백서]
- https://medium.com/centre-blog/designing-an-upgradeable-ethereum-contract-3d850f637794 [USDC proxy]