

컴파운드(Compound)

해당 번역본은 이용자가 디지털자산 관련 정보를 편리하게 확인할 수 있도록 단순 참고용으로 번역된 것이며, 투자 권유를 목적으로 하지 않습니다. 그 내용이 정확하지 않을 수 있으므로 정확한 정보 습득을 위해서는 원문을 참고하시거나 원문 작성 주체 측에 문의하시기 바랍니다. 두나무는 해당 번역본 내용의 진실성 및 정확성을 보장하지 않고, 제공된 정보에 의한 투자 결과에 대하여 어떠한 책임도 지지 않으며, 투자 손실은 투자자에게 귀속됩니다.

컴파운드 III(Compound III)

소개(Introduction)

컴파운드 III는 EVM 호환 프로토콜로, 암호화폐 자산을 담보로 제공하여 기초 자산(base asset) 을 차입할 수 있게 한다. 계정은 또한 프로토콜에 기초 자산을 공급함으로써 이자를 얻을 수 있다.

컴파운드 III의 초기 배포는 이더리움(Ethereum) 네트워크에서 이루어졌으며, 기초 자산은 USDC이다.

#development 채널이 있는 컴파운드 커뮤니티 [Discord](#) 서버와 [comp.xyz](#) 포럼에 참여하길 바란다. 컴파운드 랩스(Compound Labs)와 커뮤니티 구성원들은 컴파운드 III 위에 애플리케이션을 구축하려는 개발자를 돕기를 기대하고 있다. 질문은 프로토콜 개선에 도움이 되므로, 이 문서에서 찾을 수 없는 내용이 있다면 주저하지 말고 문의하길 바란다.

컴파운드 v2 프로토콜 독스는 [docs.compound.finance/v2](#)에서 확인할 수 있다.

네트워크(Networks)

컨트랙트 주소가 포함된 네트워크 배포 산출물은 [Comet](#) 저장소의 `deployments/` 폴더에 있다.

v3 프록시는 컴파운드 III 인스턴스와 상호작용하기 위한 유일한 주소이다. 올바른 [Comet Interface ABI](#) (CometInterface.sol)를 생성하려면 `yarn compile` 명령어로 Comet 프로젝트를 컴파일 한다.

프로토콜 컨트랙트(Protocol Contracts)

cUSDCv3

이 컨트랙트는 첫 번째 컴파운드 III 마켓과 상호작용하기 위한 주요 프록시 컨트랙트이다. 주소는 고정되어 있으며, 향후 마켓 업그레이드와는 독립적이다. 이는 [OpenZeppelin TransparentUpgradeableProxy](#) 컨트랙트이다.

cUSDCv3 구현체(cUSDCv3 Implementation)

이 컨트랙트는 설정자(Configurator)를 통해 Comet Factory가 배포한 마켓 로직 컨트랙트의 구현체이다.

이 컨트랙트와 직접 상호작용하지 말고, Comet Interface ABI를 사용하여 cUSDCv3 프록시 주소를 통해 상호작용해야 한다.

cUSDCv3 확장체(cUSDCv3 Ext)

이 컨트랙트는 프로토콜용 보조적/독립적 인터페이스를 지원하기 위한 마켓 로직 컨트랙트의 확장 버전이다. 이는 주요 프로토콜 컨트랙트의 공간을 소모하지 않고 추가 기능을 구현하기 위해 사용된다.

이 컨트랙트와 직접 상호작용하지 말고, Comet Interface ABI를 사용하여 cUSDCv3 프록시 주소를 통해 상호작용해야 한다.

설정자(Configurator)

이 컨트랙트는 *configurator*용 [프록시](#) 컨트랙트로, Comet 프록시 컨트랙트의 매개변수를 설정하고 갱신하는 데 사용된다. 설정자는 설정에 따라 Comet 로직 컨트랙트의 구현체를 배포한다. 이 패턴은 프로토콜의 매개변수를 “상수화(constantize)”함으로써 사용자에게 상당한 가스 절감 효과를 제공한다.

설정자 구현체(Configurator Implementation)

이 컨트랙트는 설정자 컨트랙트의 구현체로, 프로토콜의 예상치 못한 변경 사항을 지원할 수 있도록 업그레이드가 가능하다.

프록시 관리자(Proxy Admin)

이 컨트랙트는 Comet 및 설정자 프록시 컨트랙트의 관리자이다. 이는 OpenZeppelin이 권장하고 구현한 업그레이드 패턴에 따른 [ProxyAdmin](#)이다.

코멧 팩토리(Comet Factory)

이 컨트랙트는 Comet 구현/로직 컨트랙트의 인스턴스를 생성할 수 있는 팩토리(Factory) 컨트랙트이며, 설정자에 의해 호출된다.

보상 컨트랙트(Rewards)

이 컨트랙트는 보상 토큰(예: COMP, WETH)을 보유할 수 있으며, 핵심 프로토콜 추종 지수에 따라 사용자가 보상을 청구할 수 있도록 한다.

벌커(Bulker)

이 컨트랙트는 Comet의 핵심 기능에는 필수적이지 않은 외부 컨트랙트이다. 이는 여러 작업을 단일 트랜잭션으로 일괄(bulk) 처리할 수 있게 한다. 컴파운드 III 사용자 인터페이스에 유용한 컨트랙트이다. 다음은 일괄 트랜잭션의 단계 예시이다:

- Ether를 WETH로 래핑
- WETH 담보 공급
- WBTC 담보 공급
- USDC 차입

공급, 차입, 래핑 외에도 bulker 컨트랙트는 프로토콜 내 담보 이전과 보상 청구 기능도 지원한다.

개발자 리소스(Developer Resources)

다음의 개발자 가이드와 코드 저장소는 컴파운드 위에 구축하려는 커뮤니티 구성원을 위한 참고 자료이다. 이들은 프로토콜 배포 과정, 새로운 기능의 구축, 컴파운드 III를 인프라로 사용하는 외부 애플리케이션 구현 예제를 자세히 다룬다.

1. [Compound III Developer FAQ](#)
2. [Scenarios, Migrations, and Workflows](#)
3. [Creating a Compound III Liquidator](#)
4. [Building a Comet Extension](#)

보안(Security)

컴파운드 프로토콜의 보안은 최우선 과제이다. 개발팀은 제3자 감사 기관 및 컨설턴트와 협력하여 안전하고 신뢰할 수 있는 프로토콜을 만들기 위해 상당한 노력을 기울였다. 모든 컨트랙트 코드와 잔액은 공개적으로 검증 가능하며, 발견되지 않은 취약점을 보고하는 보안 연구자는 버그 바운티 대상이 된다.

스마트 컨트랙트의 보안성은 규모, 가시성, 그리고 시간이 검증한다고 믿는다. 따라서 주의 깊게 검토하고, 직접 보안성과 적합성을 판단해야 한다.

감사(Audits)

컴파운드 프로토콜은 [OpenZeppelin](#)과 [ChainSecurity](#)의 검토 및 감사를 받았다.

1. [Compound III Audit by OpenZeppelin](#)
2. [Compound III Security Audit by ChainSecurity](#)

이자율(Interest Rates)

기초 자산의 잔액이 양수인 사용자는 공급 금리 모델(supply rate model)에 따라 기초 자산 단위로 이자를 얻는다. 잔액이 음수인 사용자는 차입 금리 모델(borrow rate model)에 따라 이자를 지불한다. 두 모델은 별개의 이자율 모델이며, 거버넌스(governance)에 의해 설정된다.

공급 및 차입 이자율은 기초 자산의 활용도(utilization rate)의 함수이다. 각 모델에는 활용도 "kink" 지점이 포함되어 있으며, 이 지점을 초과하면 이자율이 더 빠르게 증가한다. 이자는 블록 타임스탬프를 기준으로 매초(second) 단위로 누적된다.

담보 자산(collateral assets)은 이자를 얻거나 지불하지 않는다.

공급 금리 조회(Get Supply Rate)

이 함수는 초당(per second) 공급 금리를 반환하며, 이는 백분율의 소수 표현을 10^{18} 배로 스케일한 값이다. 공급 금리를 계산하는 공식은 다음과 같다:

```
## 활용도가 Kink 파라미터보다 작거나 같을 경우

SupplyRate = supplyPerSecondInterestRateBase +
supplyPerSecondInterestRateSlopeLow * utilization

## 그 외의 경우(Else)

SupplyRate = supplyPerSecondInterestRateBase +
supplyPerSecondInterestRateSlopeLow * supplyKink +
supplyPerSecondInterestRateSlopeHigh * (utilization - supplyKink)
```

컴파운드 III의 공급 APR을 백분율로 계산하려면, 현재 활용도를 이 함수에 전달하고 결과를 10^{18} 로 나눈 뒤, 1년의 대략적인 초 단위 값으로 곱하고 100을 곱해 백분율로 스케일한다.

```
Seconds Per Year = 60 * 60 * 24 * 365
Utilization = getUtilization()
Supply Rate = getSupplyRate(Utilization)
Supply APR = Supply Rate / (10 ^ 18) * Seconds Per Year * 100
```

Comet

```
function getSupplyRate(uint utilization) public view returns (uint64)
```

- **utilization:** 금리를 계산할 활용도 값.
- **RETURNS:** 백분율의 소수 표현을 10^{18} 배로 스케일한 초당 공급 금리. 예: **317100000**은 대략 1% APR을 의미함.

Solidity

```
Comet comet = Comet(0xCometAddress);
uint supplyRate = comet.getSupplyRate(0.8e18);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const supplyRate = await comet.callStatic.getSupplyRate(0.8e18);
```

차입 금리 조회(Get Borrow Rate)

이 함수는 초당(per second) 차입 금리를 반환하며, 이는 백분율의 소수 표현을 10^{18} 배로 스케일한 값이다. 차입 금리를 계산하는 공식은 다음과 같다:

```
## 활용도가 Kink 파라미터보다 작거나 같을 경우

BorrowRate = borrowPerSecondInterestRateBase +
borrowPerSecondInterestRateSlopeLow * utilization

## 그 외의 경우(Else)

BorrowRate = borrowPerSecondInterestRateBase +
borrowPerSecondInterestRateSlopeLow * borrowKink +
borrowPerSecondInterestRateSlopeHigh * (utilization - borrowKink)
```

컴파운드 III의 차입 APR을 백분율로 계산하려면, 현재 활용도를 이 함수에 전달하고 결과를 10^{18} 로 나눈 뒤, 1년의 대략적인 초 단위 값으로 곱하고 100을 곱해 백분율로 스케일한다.

```
Seconds Per Year = 60 * 60 * 24 * 365
Utilization = getUtilization()
Borrow Rate = getBorrowRate(Utilization)
Borrow APR = Borrow Rate / (10 ^ 18) * Seconds Per Year * 100
```

Comet

```
function getBorrowRate(uint utilization) public view returns (uint64)
```

- **utilization:** 금리를 계산할 활용도 값.
- **RETURNS:** 백분율의 소수 표현을 10^{18} 배로 스케일한 초당 차입 금리. 예: **317100000**은 대략 1% APR을 의미함.

Solidity

```
Comet comet = Comet(0xCometAddress);
uint borrowRate = comet.getBorrowRate(0.8e18);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const borrowRate = await comet.callStatic.getBorrowRate(0.8e18);
```

활용도 조회(Get Utilization)

이 함수는 기초 자산의 현재 프로토콜 활용도를 반환한다. 활용도를 계산하는 공식은 다음과 같다:

활용도 = 총 차입량(TotalBorrows) / 총 공급량(TotalSupply)

Comet

```
function getUtilization() public view returns (uint)
```

- **RETURNS:** 현재 프로토콜의 활용도를 백분율의 소수 표현으로 반환하며, 부호 없는 정수 형태로 $10 \wedge 18$ 배로 스케일되어 있다. 예: **1e17** 또는 **1000000000000000000**은 10% 활용도를 의미함.

Solidity

```
Comet comet = Comet(0xCometAddress);
uint utilization = comet.getUtilization(); // example: 1000000000000000000
(1%)
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const utilization = await comet.callStatic.getUtilization();
```

담보 및 차입(Collateral & Borrowing)

사용자는 supply 함수를 사용하여 자신의 계정에 담보 자산을 추가할 수 있다. 담보는 해당 마켓이 supplyCap 이하일 때만 추가할 수 있으며, 이는 프로토콜이 담보 자산에 대해 노출되는 리스크를 제한한다.

각 담보 자산은 자산의 borrowCollateralFactor에 따라 사용자의 차입 한도(borrowing capacity)를 증가시킨다. 차입 담보 계수(borrow collateral factor)는 담보 가치 중 차입에 사용할 수 있는 비율을 나타내는 백분율이다.

예를 들어, WBTC의 차입 담보 계수가 85%라면, 계정은 공급한 WBTC의 USD 가치의 최대 85%까지 기초 자산(base asset)으로 차입할 수 있다. 담보 계수는 Get Asset Info By Address 함수를 통해 조회할 수 있다.

기초 자산은 withdraw 함수를 사용하여 차입할 수 있으며, 그 결과 생성된 차입 잔액은 담보 계수 요건을 충족해야 한다. 만약 차입 계정이 이후 담보 계수 요건을 충족하지 못하면, 추가 차입은 불가능하며, 더 많은 담보를 공급하거나 supply 함수를 통해 차입 잔액을 줄여야 한다.

계정의 잔액(balance) 은 기초 자산 기준으로 부호가 있는 정수(signed integer)로 저장된다. 잔액이 0보다 크면 기초 자산이 공급된 것이고, 잔액이 0보다 작으면 기초 자산이 차입된 것을 의미한다.

참고: 18자리 소수를 사용하는 자산의 기초 자산 잔액은 값이 $2^{103}/10^{18} \approx 10$ 조 달러 수준에서 오버플로우가 발생할 수 있다.

계정 잔액은 Comet 내부적으로 principal 값(역시 부호 있는 정수)으로 저장된다. principal 값, 또는 day-zero balance라고 불리는 이 값은 시간 T_o 에서의 잔액이 현재 시점까지 이자가 누적된 후의 잔액과 동일하도록 계산된 초기값이다.

공급 및 차입에 대한 전역 인덱스(indices) 는 부호 없는 정수(unsigned integer)로, 시간이 지남에 따라 각 측면에서 발생한 이자를 반영하며 증가한다. 계정이 프로토콜과 상호작용할 때 인덱스는 업데이트되어 저장된다. 계정의 현재 잔액은 아래의 공식을 사용해 현재 인덱스로 계산할 수 있다.

$$\text{Balance} = \text{Principal} * \text{BaseSupplyIndex} \quad [\text{Principal} > 0]$$
$$\text{Balance} = \text{Principal} * \text{BaseBorrowIndex} \quad [\text{Principal} < 0]$$

공급(Supply)

supply 함수는 자산을 프로토콜로 전송하고 계정의 잔액에 추가한다. 이 함수는 **담보 공급, 기초 자산 공급, 기초 자산 차입 상황**의 세 가지 용도로 사용될 수 있다.

기초 자산을 공급하여 계정 잔액이 0보다 커지는 경우, 해당 기초 자산은 현재 공급 금리에 따라 이자를 얻게 된다. 단, 공급된 담보 자산은 이자를 발생시키지 않는다.

컴파운드 III에 자산을 공급하는 방법은 세 가지가 있다. 첫 번째는 호출자 본인(on behalf of the caller)을 위한 것이고, 두 번째는 별도의 계정으로 공급하는 것이며, 세 번째는 관리자가 특정 계정을 대신하여 공급하는 것이다.

자산을 컴파운드 III에 공급하기 전에, 호출자는 반드시 해당 자산의 ERC-20 approve 함수를 실행하여 Comet 컨트랙트에 대한 승인 절차를 완료해야 한다.

Comet

```
function supply(address asset, uint amount)

function supplyTo(address dst, address asset, uint amount)

function supplyFrom(address from, address dst, address asset, uint amount)
```

- **asset**: 자산의 스마트 컨트랙트 주소.
- **amount**: 컴파운드 III에 공급할 자산의 양을 정수로 표현한 값. `MaxUint256` 값을 지정하면 `dst`의 기초 자산 차입 잔액을 전액 상환한다.
- **dst**: 프로토콜 내에서 공급된 자산이 입금(credit)되는 계정 주소.
- **from**: 자산을 공급할 발신자 주소. 이 계정은 `Supply`를 호출하기 전에 `Allow` 메서드를 사용하여 송신자가 자신의 토큰을 전송할 수 있도록 허용해야 한다.
- **RETURN**: 반환값 없음. 오류 발생 시 트랜잭션이 취소(revert)된다.

Solidity

```
Comet comet = Comet(0xCometAddress);
comet.supply(0xERC20Address, 1000000);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
await comet.supply(usdcAddress, 1000000);
```

인출 또는 차입(Withdraw or Borrow)

`withdraw` 메서드는 현재 진행형인 차입을 담보하지 않고 있는 **담보 자산을 인출**하는 데 사용된다. 또한, 계정이 충분한 담보를 공급한 경우, 프로토콜로부터 **기초 자산을 차입**하는 데에도 사용된다. 이 메서드는 허용된 관리자 주소(allowed manager address)로부터도 호출할 수 있다.

컴파운드 III는 최소 차입 포지션 크기(minimum borrow position size)를 구현하며, 이는 프로토콜 설정에서 `baseBorrowMin` 값으로 확인할 수 있다.

차입을 위한 `withdraw` 트랜잭션 실행 결과 계정의 차입 규모가 `baseBorrowMin`보다 작게 되면 트랜잭션은 취소된다.

Comet

```
function withdraw(address asset, uint amount)

function withdrawTo(address to, address asset, uint amount)

function withdrawFrom(address src, address to, address asset, uint amount)
```

- **asset**: 트랜잭션에서 인출 또는 차입되는 자산의 스마트 컨트랙트 주소.
- **amount**: 인출하거나 차입할 자산의 양. `MaxUint256` 값을 지정하면 `src`의 기초 자산 잔액을 전부 인출한다.
- **to**: 인출되거나 차입된 자산이 전송될 주소.
- **src**: 인출 또는 차입을 수행하는 계정의 주소. `withdrawFrom` 메서드는 허용된 관리자(allowed manager)만 호출할 수 있다.

- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
comet.withdraw(0xwbtcAddress, 100000000);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
await comet.withdraw(usdcAddress, 100000000);
```

담보 잔액(Collateral Balance)

이 함수는 프로토콜 내 특정 계정이 보유한 담보 자산의 현재 잔액을 반환한다.

Comet

```
function collateralBalanceOf(address account, address asset) external  
view returns (uint128)
```

- account: 담보 잔액을 조회할 계정의 주소.
- asset: 담보 자산의 스마트 컨트랙트 주소.
- RETURNS: 지정된 계정이 프로토콜 내에서 보유한 담보 자산의 잔액을 반환하며, 이는 자산 컨트랙트의 “decimals” 정수 자릿수만큼 10의 거듭제곱으로 스케일된 부호 없는 정수(unsigned integer) 값이다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint balance = comet.collateralBalanceOf(0xAccount, 0xUsdcAddress);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const balance = await comet.callStatic.collateralBalanceOf('0xAccount',  
'0xUsdcAddress');
```

차입 담보 상태(Borrow Collateralization)

이 함수는 전달된 계정이 차입 담보 계수에 따라 비음수 유동성(non-negative liquidity)을 가지고 있으면 `true`를 반환한다. 계정이 차입 포지션을 확장하기에 충분한 유동성을 보유하지 않은 경우, 이 함수는 `false`를 반환한다. 단, 반환값이 `false`라고 해서 해당 계정이 현재 청산(liquidation) 가능한 상태임을 의미하는 것은 아니다. (자세한 내용은 [isLiquidatable](#) 함수를 참조)

Comet

```
function isBorrowCollateralized(address account) public view returns (bool)
```

- `account`: 담보 상태를 검사할 계정의 주소.
- `RETURNS`: 계정이 차입에 충분한 유동성을 보유하고 있으면 `true`를 반환한다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
bool isCollateralized = comet.isBorrowCollateralized(0xAccount);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const isCollateralized = await  
comet.callStatic.isBorrowCollateralized('0xAccount');
```

최소 차입 잔액(Minimum Borrow Balance)

이 함수는 기초 자산 단위로 허용되는 최소 차입 잔액을 반환한다. 계정의 최초 차입 규모는 이 값 이상이어야 하며, 이후의 추가 차입은 어떤 크기든 가능하다.

Comet

```
function baseBorrowMin() public view returns (uint256)
```

- `RETURNS`: 프로토콜에서 허용하는 최소 차입 잔액을 반환하며, 기초 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 부호 없는 정수(unsigned integer) 값이다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint baseBorrowMin = comet.baseBorrowMin();
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const baseBorrowMin = await comet.callStatic.baseBorrowMin();
```

청산(Liquidation)

청산은 청산 담보 계수(liquidation collateral factors)에 의해 결정된다. 이는 차입 한도를 결정하는 차입 담보 계수와는 별개이자 더 높은 값으로 설정되며, 모든 신규 포지션에 대해 가격 완충(price buffer)을 보장함으로써 차입자와 프로토콜을 보호한다. 또한 이 메커니즘은 기존 포지션을 청산시키지 않고도 거버넌스가 차입 담보 계수를 조정할 수 있도록 한다. 계정의 차입 잔액이 청산 담보 계수로 설정된 한도를 초과하면, 해당 계정은 청산 대상이 된다. 청산자(liquidator, 즉 봇, 컨트랙트, 또는 사용자)는 absorb 함수를 호출하여 계정의 담보 소유권을 포기시키고, 담보 자산의 가치에서 벌금(liquidationFactor)을 차감한 금액을 기초 자산(base asset)으로 사용자에게 반환한다. 청산된 사용자는 더 이상 부채를 가지지 않으며, 일반적으로 기초 자산의 초과(이자 발생) 잔액을 보유하게 된다. 각 청산은 프로토콜의 기초 자산 준비금(reserves)으로 충당되며, 그 대가로 프로토콜은 담보 자산을 획득한다. 남은 준비금이 거버넌스에서 설정한 target보다 적은 경우, 청산자는 기초 자산을 사용하여 담보를 discount된 가격으로 buy할 수 있으며, 이는 프로토콜의 기초 자산 준비금을 증가시킨다.

청산 가능 계정(Liquidatable Accounts)

이 함수는 전달된 계정이 청산 담보 계수를 기준으로 음수 유동성(negative liquidity)을 가지고 있으면 true를 반환한다. 반환값이 true인 경우, 해당 계정은 현재 청산 가능한 상태임을 의미한다.

Comet

```
function isLiquidatable(address account) public view returns (bool)
```

- account: 청산 가능 여부를 검사할 계정의 주소.
- RETURN: 계정이 현재 청산 가능한 상태일 경우 true를 반환한다.

Solidity

```
Comet comet = Comet(0xCometAddress);
bool isLiquidatable = comet.isLiquidatable(0xAccount);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const isLiquidatable = await
comet.callStatic.isLiquidatable('0xAccount');
```

흡수/청산(Absorb)

이 함수는 담보 가치가 부족한(underwater) 계정을 청산하기 위해 어떤 주소든 호출할 수 있다. 함수 실행 시 계정의 부채가 프로토콜 계정으로 이전되며, 계정의 차입금을 상환하기 위해 현금 준비금(cash reserves)이 감소하고, 담보 자산이 프로토콜의 자체 잔액으로 추가된다. 함수 호출자가 소비한 가스의 양은 기록되는데, 향후 거버넌스를 통해 이에 대한 보상을 받을 수 있다.

Comet

```
function absorb(address absorber, address[] calldata accounts)
```

- **absorber**: 청산이 성공적으로 수행될 때 청산자 포인트(liquidator points)를 부여받는 계정.
- **accounts**: 청산 대상인 담보 부족(underwater) 계정들의 배열.
- **RETURN**: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
comet.absorb(0xMyAddress, [ 0xUnderwaterAddress ]);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
await comet.absorb('0xMyAddress', [ '0xUnderwaterAddress' ]);
```

담보 매수(Buy Collateral)

이 함수는 어떤 계정이든 기초 자산을 사용하여 프로토콜로부터 담보 자산을 매수할 수 있도록 한다. 매수 시 가격은 프라이스 피드(Price Feed)에서 제공하는 시세보다 할인(discount)된 가격이 적용된다. 매수자는 허용 가능한 최대 슬리피지를 나타내기 위해 최소 담보 매수량을 지정해야 한다.

이 함수는 계정이 청산된 후, 매수 가능한 담보가 존재할 때 사용할 수 있다.

이 작업은 프로토콜의 준비금(reserves)을 증가시킨다. 매수 가능한 담보의 양은 담보 잔액(Collateral Balance) 함수를 호출하여 확인할 수 있으며, 담보의 가격은 quoteCollateral 함수를 통해 확인할 수 있다.

Comet

```
function buyCollateral(address asset, uint minAmount, uint baseAmount,  
address recipient) external
```

- **asset**: 담보 자산의 스마트 컨트랙트 주소.
- **minAmount**: 구매자가 수령해야 하는 담보 토큰의 최소 수량으로, 담보 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 값.
- **baseAmount**: 담보 매수에 사용되는 기초 자산의 수량으로, 기초 자산 컨트랙트의 “decimals”

자릿수만큼 10의 거듭제곱으로 스케일된 값.

- **recipient**: 구매된 담보 자산을 수령할 계정의 주소.
- **RETURN**: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
comet.buyCollateral(0xAssetAddress, 5e18, 5e18, 0xRecipient);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
await comet.buyCollateral('0xAssetAddress', 5e18, 5e18, '0xRecipient');
```

매도 호가(Ask Price)

청산된 계정의 차입금을 상환하기 위해, 프로토콜은 압류된(seized) 담보 자산을 판매해야 한다. *매도 호가*는 거버넌스에 의해 설정된 할인율로 판매되는 자산의 가격을 의미한다. 이 함수는 프로토콜의 프라이스 피드(price feed)에서 반환된 가격을 사용하며, 자산의 할인율은 StoreFrontPriceFactor와 자산의 LiquidationFactor를 이용한 다음 공식을 통해 계산된다.

```
DiscountFactor = StoreFrontPriceFactor * (1e18 -  
Asset.LiquidationFactor)
```

Comet

```
function quoteCollateral(address asset, uint baseAmount) public view  
returns (uint)
```

- **address**: 시세가 산정되는 자산의 스마트 컨트랙트 주소.
- **amount**: 할인된 담보를 구매하기 위해 사용되는 기초 자산의 양으로, 기초 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 정수 값.
- **RETURN**: 해당 기초 자산으로 구매할 수 있는 담보 자산의 양을 반환하며, 담보 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 정수 값이다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint askPrice = comet.quoteCollateral(0xERC20Address, 10000000000);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const askPrice = await
comet.callStatic.quoteCollateral('0xERC20Address', 1000000);
```

청산자 포인트(Liquidator Points)

프로토콜은 `absorb` 함수가 성공적으로 실행된 횟수를 기록하며, 청산자가 사용한 가스량과 함께 청산자 “포인트(liquidator points)”를 집계하여 추적한다.

Comet

```
mapping(address => LiquidatorPoints) public liquidatorPoints;
```

- `address`: 청산자 계정의 주소.
- `RETURN`: 청산자 계정과 관련된 저장된 데이터를 포함하는 구조체를 반환한다.
- `numAbsorbs`: `absorb` 함수가 성공적으로 호출된 횟수를 나타내는 Solidity `uint32` 값.
- `numAbsorbed`: 청산자의 `absorb` 함수 호출 결과 프로토콜에 의해 성공적으로 청산된 계정 수를 나타내는 Solidity `uint64` 값.
- `approxSpend`: 청산자가 `absorb` 함수를 호출하면서 소비한 모든 가스의 총합을 나타내는 Solidity `uint128` 값.

Solidity

```
Comet comet = Comet(0xCometAddress);
LiquidatorPoints pointsData =
comet.liquidatorPoints(0xLiquidatorAddress);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const [ numAbsorbs, numAbsorbed, approxSpend ] = await
comet.callStatic.liquidatorPoints('0xLiquidatorAddress');
```

준비금(Reserves)

준비금은 프로토콜 내부에 저장된 기초 자산 또는 담보 자산의 잔액으로, 부실 채무(bad debt)로부터 사용자를 자동으로 보호한다. 준비금은 또한 거버넌스 절차를 통해 인출되거나 사용할 수 있다.

준비금은 두 가지 방식으로 생성된다. 첫째, 차입자가 지불하는 이자와 기초 자산 공급자가 받는 이자의 차액이 프로토콜의 준비금으로 누적된다. 둘째, 청산(liquidation) 과정에서, 거버넌스가 설정한 목표 준비금(target reserve) 수준에 따라 준비금이 사용되거나 추가될 수 있다.

기초 자산 준비금 조회(Get Base Asset Reserves)

이 함수는 프로토콜이 보유한 기초 자산의 준비금 양을 정수(integer) 형태로 반환한다.

Comet

```
function getReserves() public view returns (int)
```

- RETURNS: 프로토콜에 준비금으로 저장된 기초 자산의 양을 반환하며, 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 부호 없는 정수(unsigned integer) 값이다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint reserves = comet.getReserves();
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const reserves = await comet.callStatic.getReserves();
```

담보 자산 준비금 조회(Get Collateral Asset Reserves)

이 함수는 지정된 담보 자산에 대해 프로토콜이 보유한 준비금의 양을 정수 형태로 반환한다.

Comet

```
function getCollateralReserves(address asset) public view returns (uint)
```

- RETURNS: 프로토콜에 준비금으로 저장된 담보 자산의 양을 반환하며, 담보 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 부호 없는 정수(unsigned integer) 값이다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint reserves = comet.getCollateralReserves(0xCollateralAsset);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const reserves = await  
comet.callStatic.getCollateralReserves(collateralAssetAddress);
```

목표 준비금(Target Reserves)

이 불변(immutable) 값은 기초 자산의 목표 준비금 양을 나타낸다. 프로토콜이 이 값보다 크거나 같은 준비금을 보유한 경우, [buyCollateral](#) 함수는 더 이상 성공적으로 호출될 수 없다.

Comet

```
function targetReserves() public view returns (uint)
```

- RETURN: 기초 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 정수 형태의 기초 자산 목표 준비금 값을 반환한다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint targetReserves = comet.targetReserves();
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const targetReserves = await comet.callStatic.targetReserves();
```

계정 관리(Account Management)

컴파운드 III 계정은 자체 관리(self-management) 외에도, 다른 주소가 해당 계정에 대한 쓰기 권한(write permission)을 가질 수 있도록 허용할 수 있다. 계정 관리자는 프로토콜 내에서 다른 계정을 대신 하여 담보를 인출하거나 전송할 수 있다. 이는 계정이 [allow](#) 함수를 통해 권한을 활성화한 후에만 가능하다.

허가(Allow)

다른 주소가 송신자(sender)의 주소를 대신하여 인출하거나 전송할 수 있도록 허가하거나, 이를 금지한다.

Comet

```
function allow(address manager, bool isAllowed)
```

- msg.sender: 관리자 권한을 부여하거나 해제할 계정의 주소.
- manager: 소유자의 관리자(manager)로 지정되거나, 더 이상 관리자가 아닌 계정의 주소.
- isAllowed: 관리자를 추가하려면 true, 제거하려면 false로 설정.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
comet.allow(0xmanager, true);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
await comet.allow(managerAddress, true);
```

서명 기반 허가(Allow By Signature)

이 함수는 EIP-712 오프라인 서명(EIP-712 offline signature)을 사용하여 권한을 제출할 수 있도록 하는 allow 함수의 별도 버전이다.

오프라인 서명을 생성하는 방법에 대한 자세한 내용은 [EIP-712](#)를 참고하라.

Comet

```
function allowBySig(  
    address owner,  
    address manager,  
    bool isAllowed_,  
    uint256 nonce,  
    uint256 expiry,  
    uint8 v,  
    bytes32 r,  
    bytes32 s  
) external
```

- owner: 관리자 권한을 부여하거나 해제할 계정의 주소. 서명자는 반드시 해당 소유자(owner) 주소여야 한다.
- manager: 소유자의 관리자(manager)로 지정되거나, 더 이상 관리자가 아닌 계정의 주소.
- isAllowed: 관리자를 추가하려면 true, 제거하려면 false로 설정.
- nonce: 서명 검증 시 일치해야 하는 컨트랙트 상태 값으로, 컨트랙트의 공개 userNonce 매핑에서 조회할 수 있다.
- expiry: 서명의 만료 시점을 나타내며, 유닉스 시간(초 단위)의 블록 타임스탬프(uint) 값이다.
- v: 서명의 복구 바이트(recovery byte).
- r: ECDSA 서명 쌍의 절반에 해당하는 값.
- s: ECDSA 서명 쌍의 나머지 절반에 해당하는 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
comet.allowBySig(0xowner, 0xmanager, true, nonce, expiry, v, r, s);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
await comet.allowBySig('0xowner', '0xmanager', true, nonce, expiry, v, r, s);
```

사용자 논스(User Nonce)

이 함수는 EVM 계정의 논스(nonce)와 유사한 사용자 논스를 반환하며, allowBySig 함수에서 사용된다.

Comet

```
function userNonce(address) returns (uint)
```

- address: 논스를 조회할 계정의 주소.
- RETURN: 지정된 계정의 논스 값을 나타내는 정수.

Solidity

```
Comet comet = Comet(0xCometAddress);
uint nonce = comet.userNonce(0xAccount);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const nonce = await comet.callStatic.userNonce('0xAccount');
```

버전(Version)

이 함수는 allowBySig 함수에서 사용되는 프로토콜 버전 값을 반환한다.

Comet

```
function version() view returns (string memory)
```

- RETURN: 프로토콜 버전 번호를 나타내는 문자열.

Solidity

```
Comet comet = Comet(0xCometAddress);
uint version = comet.version();
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const version = await comet.callStatic.version();
```

계정 권한(Account Permissions)

이 함수는 특정 계정의 관리자 주소에 대한 권한 상태를 나타내는 불리언(boolean) 값을 반환한다.

Comet

```
function hasPermission(address owner, address manager) public view
returns (bool)
```

- owner: 다른 계정에 의해 관리될 수 있는 계정의 주소.
- manager: 다른 계정을 관리할 수 있는 관리자 권한을 가진 계정의 주소.
- RETURNS: manager 주소가 현재 owner 주소의 관리자일 경우 true를 반환한다.

Solidity

```
Comet comet = Comet(0xCometAddress);
bool isManager = comet.hasPermission(0xOwner, 0xManager);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const isManager = await comet.callStatic.hasPermission('0xOwner',
'0xManager');
```

전송(Transfer)

이 함수는 프로토콜 내에서 자산을 다른 주소로 전송하는 데 사용된다. 계정의 관리자는 해당 계정을 대신하여 전송을 수행할 수도 있다. 이 함수 실행 시 계정의 잔액은 변하지만, 자산은 프로토콜 컨트랙트 밖으로 이동하지 않는다. 만약 전송으로 인해 계정이 청산 가능한 상태가 된다면, 전송은 실패한다.

transfer 함수에는 두 가지 변형이 있다: transfer와 transferAsset. transfer는 ERC-20 표준을 따르며 기초 자산을 전송하고, transferAsset은 전송할 특정 자산을 명시적으로 지정해야 한다.

Comet

```
function transfer(address dst, uint amount)
```

```
function transferFrom(address src, address dst, uint amount)
```

```
function transferAsset(address dst, address asset, uint amount)
```

```
function transferAssetFrom(address src, address dst, address asset, uint amount)
```

- **dst**: 트랜잭션에서 자산을 수신하는 계정의 주소.
- **src**: 트랜잭션에서 자산을 전송하는 계정의 주소. 이 전송 메서드는 허용된 관리자(allowed manager)만 호출할 수 있다.
- **asset**: 트랜잭션에서 전송되는 자산의 ERC-20 주소.
- **amount**: 전송할 자산의 양. `MaxUint256` 값을 지정하면 **src**의 기초 자산 잔액 전체가 전송된다.
- **RETURN**: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
comet.transfer(0xreceiver, 0xwbtcAddress, 1000000000);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
await comet.transfer(receiverAddress, usdcAddress, 1000000000);
```

인터페이스 및 ERC-20 호환성(Interfaces & ERC-20 Compatibility)

Comet 컨트랙트는 기초 자산을 위한 완전한 ERC-20 호환 래퍼(wrapper)이다. ERC-20의 모든 인터페이스 메서드는 기초 자산을 공급하거나 차입하는 계정에 대해 외부로 노출되어 있다.

CometInterface.sol 컨트랙트 파일에는 Comet 컨트랙트를 위한 Solidity 인터페이스 예제가 포함되어 있다.

프로토콜 보상(Protocol Rewards)

컴파운드 III는 프로토콜을 사용하는 계정에 대한 보상을 추적하는 내장 시스템을 갖추고 있다. 기초 자산을 공급하거나 차입한 계정의 전체 보상 누적 이력(accrual history)이 추적된다. 보상은 임의의 ERC-20 토큰일 수 있다. 컴파운드 III 계정에 보상이 누적되기 위해서는, 설정에서 정의된 기초 자산 총 공급량의 임계값(baseMinForRewards)을 충족해야 한다.

보상 누적 추적(Reward Accrual Tracking)

보상 누적은 Comet 컨트랙트 내에서 추적되며, 사용자는 외부 Comet Rewards 컨트랙트로부터 보상을 청구할 수 있다. 보상은 최대 6자리 소수 정밀도(6 decimals of precision)로 계산된다.

Comet

```
function baseTrackingAccrued(address account) external view returns
(uint64);
```

- RETURN: 지정된 계정이 프로토콜 내에서 기초 자산을 사용한 정도에 따라 누적된 보상 토큰의 양을 반환하며, 10^6 배로 스케일된 값이다.

Solidity

```
Comet comet = Comet(0xCometAddress);
uint64 accrued = comet.baseTrackingAccrued(0xAccount);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const accrued = await comet.callStatic.baseTrackingAccrued('0xAccount');
```

누적 보상 조회(Get Reward Accrued)

계정이 아직 청구하지 않은 누적 보상 토큰의 양은 외부 Comet Rewards 컨트랙트에서 조회할 수 있다.

Comet Rewards

```
struct RewardOwed {
    address token;
    uint owed;
}
```

```
function getRewardOwed(address comet, address account) external returns
(RewardOwed memory)
```

- RETURNS: 아직 청구되지 않은 누적 보상 토큰의 양을 반환하며, 보상 토큰 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 값이다.

Solidity

```
CometRewards rewards = CometRewards(0xRewardsAddress);
RewardOwed reward = rewards.getRewardOwed(0xCometAddress, 0xAccount);
```

Ethers.js v5.x

```
const rewards = new ethers.Contract(contractAddress, abiJson, provider);
const [ tokenAddress, amtOwed ] = await
rewards.callStatic.getRewardOwed(cometAddress, accountAddress);
```

보상 청구 (Claim Rewards)

모든 계정은 특정 계정의 보상을 청구할 수 있다. 또한 계정 소유자와 관리자는 특정 주소로 보상을 청구할 수도 있다. 이 보상 청구 함수들은 외부 Comet Rewards 컨트랙트에서 제공된다.

Comet Rewards

```
function claim(address comet, address src, bool shouldAccrue) external
```

```
function claimTo(address comet, address src, address to, bool  
shouldAccrue) external
```

- comet: Comet 컨트랙트의 주소.
- src: 보상을 청구할 계정의 주소.
- to: 청구된 보상을 전송받을 계정의 주소.
- shouldAccrue: true로 설정하면, 프로토콜은 전송 전에 현재 블록 시점까지 해당 계정에 누적된 보상을 반영한다.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

Solidity

```
CometRewards rewards = CometRewards(0xRewardsAddress);  
rewards.claim(0xCometAddress, 0xAccount, true);
```

Ethers.js v5.x

```
const rewards = new ethers.Contract(contractAddress, abiJson, provider);  
await rewards.claim(cometAddress, accountAddress, true);
```

거버넌스(Governance)

컴파운드 III는 COMP 토큰 보유자 및 그 대리인(delegate)에 의해 운영되는 탈중앙화 프로토콜이다. 거버넌스를 통해 커뮤니티는 컴파운드 III 프로토콜의 관리용 스마트 컨트랙트 기능을 이용하여 제안(propose), 투표(vote), 변경사항을 실행(implement)할 수 있다. 거버넌스와 타임락(Timelock)에 대한 자세한 내용은 원본 [거버넌스\(governance\)](#) 섹션을 참고하라.

모든 컴파운드 III 인스턴스는 컴파운드 v2 프로토콜의 관리자와 동일한 타임락 컨트랙트에 의해 제어된다. 거버넌스 시스템은 각 *프록시*, *설정자 구현체*, *코멧 팩토리*, 그리고 *코멧 구현체*를 제어할 권한을 가진다.

거버넌스 제안을 통해 불변 매개변수가 설정될 때마다, Comet 팩토리를 통해 새로운 Comet 구현체가 배포되어야 한다. 커뮤니티에서 제안이 승인되면, 실행 시 프록시는 새로운 구현체를 가리키도록 업데이트된다.

프로토콜의 특정 매개변수를 제안 내에서 설정하려면, 타임락은 먼저 *Configurator* 컨트랙트의 관련 `set` 메서드들을 호출한 후, *CometProxyAdmin* 컨트랙트에서 `deployAndUpgradeTo` 메서드를 호출해야 한다.

멀티체인 거버넌스(Multi-chain Governance)

컴파운드 III 프로토콜은 모든 EVM 체인에 배포될 수 있다. 배포된 인스턴스는 온체인 자산 가격 정보와, 이더리움 메인넷에서 전달된 거버넌스 메시지에 접근할 수 있어야 한다. 메인넷의 [Timelock](#) 컨트랙트는 커뮤니티가 승인한 모든 컴파운드 III 인스턴스의 관리자 역할을 한다.

메인넷 외의 체인에 배포될 때는 해당 체인에 [Bridge Receiver](#) 및 로컬 타임락(Local Timelock) 컨트랙트가 존재해야 한다. 메인넷에서 실행된 거버넌스 제안은 각 체인의 브리지를 통해 읽히며, 브리지 리시버(Bridge Receiver)에 전달되어야 한다. 로컬 타임락은 제안 실행을 통해 Comet 관리자 함수가 호출되기 전, 추가적인 지연을 가진다.

컴파운드 III 인스턴스의 초기화 내역은 [ENS 텍스트 레코드 시스템](#)을 통해 온체인에 기록된다. 이 텍스트 레코드는 거버넌스 제안에 의해서만 수정될 수 있으며, 브라우저 네트워크가 이더리움 메인넷으로 설정되어 있을 때 v3-additional-grants.compound-community-licenses.eth에서 확인할 수 있다.

코멧 팩토리 설정(Set Comet Factory)

이 함수는 코멧 팩토리의 공식 컨트랙트 주소를 설정한다. 이 함수를 호출할 수 있는 유일한 주체는 거버너(Governor)이다.

Configurator

```
function setFactory(address cometProxy, address newFactory) external
```

- `cometProxy`: 설정을 적용할 Comet 프록시의 주소.
- `newFactory`: 새롭게 지정할 Comet 컨트랙트 팩토리의 주소.
- `RETURN`: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

거버너 설정(Set Governor)

이 함수는 이후 제안 실행을 위해 컴파운드 III 프로토콜의 공식 거버너(Governor) 컨트랙트 주소를 설정한다.

Configurator

```
function setGovernor(address cometProxy, address newGovernor) external
```

- `cometProxy`: 설정을 적용할 Comet 프록시의 주소.
- `newGovernor`: 새로 지정할 컴파운드 III 거버너의 주소.
- `RETURN`: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

일시정지 관리자 설정(Set Pause Guardian)

이 함수는 컴파운드 III 프로토콜의 공식 일시정지 관리자(Pause Guardian) 컨트랙트 주소를 설정한다. 이 주소는 컴파운드 III 내에서 공급(supply), 전송(transfer), 인출(withdraw), 흡수/청산(absorb), 담보 매수(buy collateral) 등의 작업을 일시정지할 수 있는 권한을 가진다.

COMP 토큰 보유자들은 [커뮤니티 멀티시그\(Community Multi-Sig\)](#)에 의해 관리되는 일시정지 관리자 주소를 지정한다.

Configurator

```
function setPauseGuardian(address cometProxy, address newPauseGuardian)
external
```

프로토콜 기능 일시정지(Pause Protocol Functionality)

이 함수는 예기치 못한 취약성이 발견될 경우, 지정된 프로토콜 기능을 일시적으로 중단(paused)시킨다. 이 함수를 호출할 수 있는 주소는 거버너와 일시정지 관리자뿐이다.

Comet

```
function pause(
    bool supplyPaused,
    bool transferPaused,
    bool withdrawPaused,
    bool absorbPaused,
    bool buyPaused
) override external
```

- **supplyPaused**: 모든 계정의 자산 공급 기능을 활성화하거나 비활성화한다.
- **transferPaused**: 모든 계정의 프로토콜 내 자산 전송 기능을 활성화하거나 비활성화한다.
- **withdrawPaused**: 모든 계정의 프로토콜 내 자산 인출 기능을 활성화하거나 비활성화한다.
- **absorbPaused**: 프로토콜의 청산/흡수 기능을 활성화하거나 비활성화한다.
- **buyPaused**: 프로토콜이 흡수된 담보를 판매하는 기능을 활성화하거나 비활성화한다.
- **RETURN**: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

공급 일시정지 여부(Is Supply Paused)

이 함수는 프로토콜의 자산 공급 기능이 현재 일시정지 상태인지 여부를 나타내는 불리언 값을 반환한다.

Comet

```
function isSupplyPaused() override public view returns (bool)
```

- **RETURN**: 프로토콜의 공급 기능이 현재 일시정지 상태인지 여부를 나타내는 불리언 값.

전송 일시정지 여부(Is Transfer Paused)

이 함수는 프로토콜의 자산 전송 기능이 현재 일시정지 상태인지 여부를 나타내는 불리언 값을 반환한다.

Comet

```
function isTransferPaused() override public view returns (bool)
```

- RETURN: 프로토콜의 전송 기능이 현재 일시정지 상태인지 여부를 나타내는 불리언 값.

인출 일시정지 여부(Is Withdraw Paused)

이 함수는 프로토콜의 자산 인출 기능이 현재 일시정지 상태인지 여부를 나타내는 불리언 값을 반환한다.

Comet

```
function isWithdrawPaused() override public view returns (bool)
```

- RETURN: 프로토콜의 인출 기능이 현재 일시정지 상태인지 여부를 나타내는 불리언 값.

청산/흡수 일시정지 여부(Is Absorb Paused)

이 함수는 프로토콜의 청산/흡수 기능이 현재 일시정지 상태인지 여부를 나타내는 불리언 값을 반환한다.

Comet

```
function isAbsorbPaused() override public view returns (bool)
```

- RETURN: 프로토콜의 흡수 기능이 현재 일시정지 상태인지 여부를 나타내는 불리언 값.

담보 매도 일시정지 여부(Is Buy Paused)

이 함수는 프로토콜에 의한 청산/흡수된 담보 매도(selling of absorbed collateral) 기능이 현재 일시정지 상태인지 여부를 나타내는 불리언 값을 반환한다.

Comet

```
function isBuyPaused() override public view returns (bool)
```

- RETURN: 프로토콜의 담보 매도 기능이 현재 일시정지 상태인지 여부를 나타내는 불리언 값.

기초 자산 프라이스 피드 설정(Set Base Token Price Feed)

이 함수는 프로토콜의 기초 자산에 대한 공식 프라이스 피드 컨트랙트 주소를 설정한다.

Configurator

```
function setBaseTokenPriceFeed(address cometProxy, address  
newBaseTokenPriceFeed) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newBaseTokenPriceFeed: 새로 지정할 프라이스 피드 컨트랙트의 주소.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

확장 델리게이트 설정(Set Extension Delegate)

이 함수는 프로토콜의 Comet 확장 델리게이트(Comet extension delegate)의 공식 컨트랙트 주소를 설정한다. CometExt.sol에 정의된 메서드들은 Comet.sol과 동일한 프록시(proxy)를 통해 호출할 수 있다.

Configurator

```
function setExtensionDelegate(address cometProxy, address  
newExtensionDelegate) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newExtensionDelegate: 새로 지정할 확장 델리게이트 컨트랙트의 주소.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

차입 Kink 설정(Set Borrow Kink)

이 함수는 컴파운드 III 기초 자산의 차입 이자율 활용도 곡선에서의 Kink 값을 설정한다.

Configurator

```
function setBorrowKink(address cometProxy, uint64 newKink) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newKink: 새로 설정할 Kink 파라미터 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

차입 이자율 기울기 설정 - 하한(Set Borrow Interest Rate Slope - Low)

이 함수는 1년을 초 단위로 근사한 값 기준으로, 차입 이자율 곡선의 기울기 하한(slope low bound)를 설정한다.

Configurator

```
function setBorrowPerYearInterestRateSlopeLow(address cometProxy, uint64 newSlope) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newSlope: 기울기 하한을 나타내는 부호 없는 정수 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

차입 이자율 기울기 설정 - 상한(Set Borrow Interest Rate Slope - High)

이 함수는 1년을 초 단위로 근사한 값 기준으로, 차입 이자율 곡선의 기울기 상한(slope high bound)을 설정한다.

Configurator

```
function setBorrowPerYearInterestRateSlopeHigh(address cometProxy, uint64 newSlope) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newSlope: 기울기 상한 를 나타내는 부호 없는 정수 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

차입 이자율 기울기 설정 - 기준(Set Borrow Interest Rate Slope - Base)

이 함수는 1년을 초 단위로 근사한 값 기준으로, 차입 이자율 곡선의 기준 기울기(slope base)를 설정한다.

Configurator

```
function setBorrowPerYearInterestRateBase(address cometProxy, uint64 newBase) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newSlope: 기준 기울기를 나타내는 부호 없는 정수 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

공급 Kink 설정(Set Supply Kink)

이 함수는 컴파운드 III 기초 자산의 공급 이자율 활용도 곡선에서의 Kink 값을 설정한다.

Configurator

```
function setSupplyKink(address cometProxy, uint64 newKink) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newKink: 새로 설정할 Kink 파라미터 값.

- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

공급 이자율 기울기 설정 - 하한(Set Supply Interest Rate Slope - Low)

이 함수는 1년을 초 단위로 근사한 값 기준으로, 공급 이자율 곡선의 기울기 하한을 설정한다.

Configurator

```
function setSupplyPerYearInterestRateSlopeLow(address cometProxy, uint64 newSlope) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newSlope: 기울기 하한을 나타내는 부호 없는 정수 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

공급 이자율 기울기 설정 - 상한(Set Supply Interest Rate Slope - High)

이 함수는 1년을 초 단위로 근사한 값 기준으로, 공급 이자율 곡선의 기울기 상한을 설정한다.

Configurator

```
function setSupplyPerYearInterestRateSlopeHigh(address cometProxy, uint64 newSlope) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newSlope: 기울기 상한을 나타내는 부호 없는 정수 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

공급 이자율 기울기 설정 - 기준(Set Supply Interest Rate Slope - Base)

이 함수는 1년을 초 단위로 근사한 값 기준으로, 공급 이자율 곡선의 기준 기울기를 설정한다.

Configurator

```
function setSupplyPerYearInterestRateBase(address cometProxy, uint64 newBase) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newSlope: 기준 기울기를 나타내는 부호 없는 정수 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

스토어프론트 가격 계수 설정 (Set Store Front Price Factor)

이 함수는 청산 시 담보 매수자에게 지급되는 벌금(liquidation penalty)의 비율을 설정한다. 이 계수는 계정 청산/흡수 과정에서 담보 매도 시 적용되는 할인율을 계산하는 데 사용된다. 이 값은 10^{18} 배로 스케일된 소수 형태로 표현된다.

Configurator

```
function setStoreFrontPriceFactor(address cometProxy, uint64 newStoreFrontPriceFactor) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newStoreFrontPriceFactor: 새로 설정할 가격 계수로, 10^{18} 배로 스케일된 소수를 부호 없는 정수 형태로 표현한 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

기초 자산 공급 보상 속도 설정(Set Base Tracking Supply Speed)

이 함수는 기초 자산을 공급하는 계정이 보상을 누적하는 속도를 설정한다.

Configurator

```
function setBaseTrackingSupplySpeed(address cometProxy, uint64 newBaseTrackingSupplySpeed) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newBaseTrackingSupplySpeed: 연이율 기준으로 표현된 보상 누적 속도로, 10^{18} 배로 스케일된 소수 형태의 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

기초 자산 차입 보상 속도 설정(Set Base Tracking Borrow Speed)

이 함수는 기초 자산을 차입한 계정이 보상을 누적하는 속도를 설정한다.

Configurator

```
function setBaseTrackingBorrowSpeed(address cometProxy, uint64 newBaseTrackingBorrowSpeed) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newBaseTrackingBorrowSpeed: 연이율 기준으로 표현된 보상 누적 속도로, 10^{18} 배로 스케일된 소수 형태의 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

보상을 수취하기 위한 기초 자산 최소량 설정(Set Base Minimum For Rewards)

이 함수는 계정이 보상을 누적하기 위해 프로토콜에 공급해야 하는 최소 기초 자산 양을 설정한다.

Configurator

```
function setBaseMinForRewards(address cometProxy, uint104  
newBaseMinForRewards) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newBaseMinForRewards: 기초 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 기초 자산의 양.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

최소 차입 한도 설정(Set Borrow Minimum)

이 함수는 차입이 허용되는 기초 자산의 최소 금액을 설정한다.

Configurator

```
function setBaseBorrowMin(address cometProxy, uint104 newBaseBorrowMin)  
external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- setBaseBorrowMin: 기초 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 최소 차입 금액을 나타내는 부호 없는 정수 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

목표 준비금 설정(Set Target Reserves)

이 함수는 목표 준비금 금액을 설정한다. 프로토콜의 기초 자산 준비금이 이 금액에 도달하면, 청산자는 프로토콜로부터 담보를 매수할 수 없게 된다.

Configurator

```
function setTargetReserves(address cometProxy, uint104  
newTargetReserves) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newTargetReserves: 기초 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 기초 자산 준비금의 양을 나타내는 부호 없는 정수 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

신규 자산 추가(Add a New Asset)

이 함수는 거버넌스를 통해 프로토콜에 새로운 자산을 추가한다.

Configurator

```
function addAsset(address cometProxy, AssetConfig calldata assetConfig)
external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- assetConfig: 프로토콜의 자산 구성 배열에 추가될 자산의 설정값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

기존 자산 업데이트(Update an Existing Asset)

이 함수는 기존 자산을 구성하는 파라미터를 수정한다.

Configurator

```
function updateAsset(address cometProxy, AssetConfig calldata
newAssetConfig) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- newAssetConfig: 프로토콜의 자산 구성 배열 내에서 수정될 자산의 설정값으로, 모든 파라미터가 덮어쓰기(overwrite)된다.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

자산 가격 피드 업데이트(Update Asset Price Feed)

이 함수는 특정 자산의 가격 피드 컨트랙트 주소를 업데이트한다.

Configurator

```
function updateAssetPriceFeed(address cometProxy, address asset, address
newPriceFeed) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- asset: 기초 자산의 스마트 컨트랙트 주소.
- newPriceFeed: 새로 지정할 가격 피드 스마트 컨트랙트의 주소.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

차입 담보 계수 업데이트(Update Borrow Collateral Factor)

이 함수는 프로토콜 내 특정 자산의 차입 담보 계수를 업데이트한다.

Configurator

```
function updateAssetBorrowCollateralFactor(address cometProxy, address
asset, uint64 newBorrowCF) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.

- **asset**: 기초 자산의 스마트 컨트랙트 주소.
- **newBorrowCF**: 담보 계수를 나타내는 정수 값으로, 소수 값을 $10 \wedge 18$ 배로 스케일한 형태.
- **RETURN**: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

청산 담보 계수 업데이트(Update Liquidation Collateral Factor)

이 함수는 프로토콜 내 특정 자산의 청산 담보 계수를 업데이트한다.

Configurator

```
function updateAssetLiquidateCollateralFactor(address cometProxy,
address asset, uint64 newLiquidateCF) external
```

- **cometProxy**: 설정을 적용할 Comet 프록시의 주소.
- **asset**: 기초 자산의 스마트 컨트랙트 주소.
- **newLiquidateCF**: 담보 계수를 나타내는 정수 값으로, 소수 값을 $10 \wedge 18$ 배로 스케일한 형태.
- **RETURN**: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

청산 계수 업데이트(Update Liquidation Factor)

이 함수는 프로토콜 내 특정 자산의 청산 계수를 업데이트한다.

청산 계수는 0 이상 1 이하(포함)의 소수 값으로, 청산 시 담보 부족 계정에 지급되는 금액의 비율을 결정한다.

다음은 컴파운드 III 청산에서 청산 계수가 어떤 역할을 하는지에 대한 예시이다:

한 담보 부족 계정이 \$100 상당의 WBTC를 담보로 공급했다고 가정하자.

만약 WBTC의 청산 계수가 0.9라면, 청산자가 해당 계정을 청산/흡수할 때 사용자는 기초 자산으로 \$90을 지급받게 된다.

Configurator

```
function updateAssetLiquidationFactor(address cometProxy, address asset,
uint64 newLiquidationFactor) external
```

- **cometProxy**: 설정을 적용할 Comet 프록시의 주소.
- **asset**: 기초 자산의 스마트 컨트랙트 주소.
- **newLiquidationFactor**: 청산 계수를 나타내는 정수 값으로, 소수 값을 $10 \wedge 18$ 배로 스케일한 형태.
- **RETURN**: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

자산 공급 한도 설정(Set Asset Supply Cap)

이 함수는 프로토콜에 공급할 수 있는 자산의 최대 한도를 설정한다. 총 공급량이 이 값을 초과하게 되는 공급 트랜잭션은 취소되어 실행되지 않는다.

Configurator

```
function updateAssetSupplyCap(address cometProxy, address asset, uint128 newSupplyCap) external
```

- cometProxy: 설정을 적용할 Comet 프록시의 주소.
- asset: 기초 자산의 스마트 컨트랙트 주소.
- newSupplyCap: 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 자산의 양을 나타내는 부호 없는 정수 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

ERC-20 관리자 주소 승인(ERC-20 Approve Manager Address)

이 함수는 특정 관리자 주소에 대해 Comet 컨트랙트가 보유한 자산의 ERC-20 허용량(allowance)을 설정한다. 이 함수는 거버너만 호출할 수 있다.

거버너스 공격이 발생할 경우, 공격자는 이 함수를 악용하여 자신에게 Comet 컨트랙트 내 모든 ERC-20 토큰을 자유롭게 전송할 수 있는 권한을 부여하는 제안을 생성할 수 있다.

이론적으로, 공격자는 막대한 투표권을 확보하거나, 무해해 보이면서 인기있는 제안에 악의적인 단계를 추가 하여 커뮤니티가 이를 승인 전에 감지하지 못하도록 해야 한다.

Comet

```
function approveThis(address manager, address asset, uint amount) override external
```

- manager: 허용량이 변경될 관리자 계정의 주소.
- asset: 자산의 스마트 컨트랙트 주소.
- amount: 관리자에게 승인된 자산의 양으로, 정수 형태로 표현된 값.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

거버너 이전(Transfer Governor)

이 함수는 설정자의 거버너 주소를 변경한다.

Configurator

```
function transferGovernor(address newGovernor) external
```

- newGovernor: 설정자의 새 거버너 주소.
- RETURN: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

준비금 인출(Withdraw Reserves)

이 함수는 거버넌스가 프로토콜의 기초 자산 준비금을 인출하여 지정된 주소로 전송할 수 있도록 한다. 이 함수는 거버너 주소만 호출할 수 있다.

Comet

```
function withdrawReserves(address to, uint amount) external
```

- **to**: 기초 자산 토큰을 수령할 수신자 주소.
- **amount**: 전송할 기초 자산의 양으로, 기초 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 값.
- **RETURN**: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

헬퍼 함수(Helper Functions)

총 공급량(Total Supply)

프로토콜에 공급된 기초 자산의 총량과, 공급자에게 누적된 이자를 합한 값을 나타낸다.

Comet

```
function totalSupply() override external view returns (uint256)
```

- **RETURN**: 기초 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 기초 자산의 양.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint256 totalSupply = comet.totalSupply();
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const totalSupply = await comet.callStatic.totalSupply();
```

총 차입량(Total Borrow)

프로토콜에서 현재 차입된 기초 자산의 총량과, 모든 차입에 대해 누적된 이자를 합한 값을 나타낸다.

Comet

```
function totalBorrow() virtual external view returns (uint256)
```

- RETURN: 기초 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 기초 자산의 양.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint256 totalBorrow = comet.totalBorrow();
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const totalBorrow = await comet.callStatic.totalBorrow();
```

총 담보량(Total Collateral)

프로토콜은 모든 계정이 공급한 담보 자산의 현재 총량을 추적한다. 각 유효한 담보 자산의 총합은 자산 주소를 키로 하는 매핑(mapping)을 통해, 해당 자산 정보를 담은 구조체에 연결되어 관리된다.

Comet

```
struct TotalsCollateral {  
    uint128 totalSupplyAsset;  
    uint128 _reserved;  
}  
  
mapping(address => TotalsCollateral) public totalsCollateral;
```

- address: 담보 자산의 컨트랙트 주소.
- RETURN: 프로토콜 내 담보 자산 총합과 관련된 데이터를 포함하는 구조체(struct)를 반환한다.
- totalSupplyAsset: 프로토콜에 저장된 담보 자산의 총합을 나타내는 Solidity uint128 값으로, 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 값.

Solidity

```
Comet comet = Comet(0xCometAddress);  
TotalsCollateral totalsCollateral =  
comet.totalsCollateral(0xERC20Address);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const [ totalSupplyAsset ] = await
comet.callStatic.totalsCollateral('0xERC20Address');
```

공급된 기초 자산 잔액(Supplied Base Balance)

이 함수는 프로토콜 내 특정 계정이 보유한 기초 자산의 현재 잔액(이자 포함)을 반환한다. 계정이 현재 차입 중이거나 기초 자산을 공급하지 않은 경우, 이 함수는 0을 반환한다.

Comet

```
function balanceOf(address account) external view returns (uint256)
```

- **account**: 기초 자산 잔액을 조회할 계정의 주소.
- **RETURNS**: 지정된 계정이 프로토콜 내에서 보유한 기초 자산의 잔액(이자 포함)을 반환하며, 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 부호 없는 정수 값이다.

Solidity

```
Comet comet = Comet(0xCometAddress);
uint balance = comet.balanceOf(0xAccount);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const balance = await comet.callStatic.balanceOf('0xAccount');
```

차입 잔액(Borrow Balance)

이 함수는 프로토콜 내 특정 계정이 차입한 기초 자산의 현재 잔액(이자 포함)을 반환한다. 계정의 기초 자산 잔액이 0 이상(비차입 상태)인 경우, 이 함수는 0을 반환한다.

Comet

```
function borrowBalanceOf(address account) external view returns
(uint256)
```

- **account**: 차입된 기초 자산 잔액을 조회할 계정의 주소.
- **RETURNS**: 지정된 계정이 프로토콜 내에서 차입한 기초 자산의 잔액(이자 포함)을 반환하며, 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 부호 없는 정수 값이다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint owed = comet.borrowBalanceOf(0xAccount);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const owed = await comet.callStatic.borrowBalanceOf('0xAccount');
```

계정 데이터(Account Data)

프로토콜은 기초 자산을 공급하거나 차입하는 각 계정에 대해 원금 및 인덱스 등의 데이터를 추적한다. 이 데이터는 계정 주소를 키로 하는 매핑에 저장되며, 해당 주소는 관련 정보를 담은 구조체를 가리킨다.

Comet

```
struct UserBasic {  
    int104 principal;  
    uint64 baseTrackingIndex;  
    uint64 baseTrackingAccrued;  
    uint16 assetsIn;  
}  
  
mapping(address => UserBasic) public userBasic;
```

- address: 프로토콜을 사용한 계정의 주소.
- RETURN: 해당 계정과 관련된 저장 데이터를 포함하는 구조체를 반환한다.
- principal: 계정이 프로토콜에 공급한(0보다 큰 값) 또는 프로토콜로부터 차입한(0보다 작은 값) 기초 자산의 양을 나타내는 Solidity int104 값.
- baseTrackingIndex: 계정의 인덱스를 나타내는 Solidity uint64 값.
- baseTrackingAccrued: 계정이 누적한 이자 금액을 나타내는 Solidity uint64 값.
- assetsIn: 계정이 담보로 공급한 자산들을 추적하는 Solidity uint16 값. 이 저장 방식은 내부 구현용이며, 가스 절감을 위해 사용된다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
UserBasic userBasic = comet.userBasic(0xAccount);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const [ principal, baseTrackingIndex, baseTrackingAccrued, assetsIn ] =  
await comet.callStatic.userBasic('0xAccount');
```

자산 정보 조회(Get Asset Info)

이 함수는 담보 자산에 대한 정보를 반환하며, 여기에는 담보 계수(collateral factors), 자산의 프라이스 피드 주소 등 다양한 데이터가 포함된다. 모든 자산의 정보를 반복(loop)하여 가져오려면, 현재 프로토콜에서 지원하는 담보 자산의 총 개수를 나타내는 상수 numAssets를 사용하면 된다.

Comet

```
struct AssetInfo {
    uint8 offset;
    address asset;
    address priceFeed;
    uint64 scale;
    uint64 borrowCollateralFactor;
    uint64 liquidateCollateralFactor;
    uint64 liquidationFactor;
    uint128 supplyCap;
}

function getAssetInfo(uint8 i) public view returns (AssetInfo memory)
```

- i: 프로토콜에 자산이 추가된 순서에 따른 담보 자산의 인덱스. 인덱스는 0부터 시작한다.
- RETURNS: AssetInfo라는 구조체 형태로 담보 자산 정보를 반환한다.
- offset: 프로토콜에 자산이 추가된 순서에 따른 담보 자산의 인덱스.
- asset: 자산의 스마트 컨트랙트 주소.
- priceFeed: 해당 담보 자산의 프라이스 피드 컨트랙트 주소.
- scale: 담보 자산의 소수 자릿수를 기준으로 10^x 값을 가지는 정수(integer).
- borrowCollateralFactor: 계정이 담보 가치를 기준으로 차입할 수 있는 비율(이자 포함)을 나타내는 차입 담보 계수.
반환값은 10^{18} 배로 스케일된 소수를 나타내는 정수 값이다.
예: 6500000000000000000 → 65%.
- liquidateCollateralFactor: 계정이 청산되기 전에 차입할 수 있는 담보 가치 비율(이자 포함)을 나타내는 청산 담보 계수.
반환값은 10^{18} 배로 스케일된 소수를 나타내는 정수 값이다.
예: 8500000000000000000 → 85%.
- liquidationFactor: 청산 시 적용되는 계수로, 10^{18} 배로 스케일된 소수를 나타내는 정수 값이다.
예: 9300000000000000000 → 7%의 청산 벌금을 의미한다.
- supplyCap: 담보 자산의 공급 한도를 나타내는 정수 값으로, 자산의 소수 자릿수를 기준으로 10^x 배로 스케일된다.

Solidity

```
Comet comet = Comet(0xCometAddress);
AssetInfo info = comet.getAssetInfo(0);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);
const infoObject = await comet.callStatic.getAssetInfo(0);
```

주소 기반 자산 정보 조회(Get Asset Info By Address)

이 함수는 특정 담보 자산에 대한 정보를 반환한다.

Comet

```
struct AssetInfo {
    uint8 offset;
    address asset;
    address priceFeed;
    uint64 scale;
    uint64 borrowCollateralFactor;
    uint64 liquidateCollateralFactor;
    uint64 liquidationFactor;
    uint128 supplyCap;
}

function getAssetInfoByAddress(address asset) public view returns
(AssetInfo memory)
```

- address: 담보 자산의 주소.
- RETURNS: AssetInfo라는 구조체 형태로 담보 자산 정보를 반환한다.
- offset: 프로토콜에 자산이 추가된 순서에 따른 담보 자산의 인덱스.
- asset: 담보 자산의 스마트 컨트랙트 주소.
- priceFeed: 해당 담보 자산의 프라이스 피드 컨트랙트 주소.
- scale: 담보 자산의 소수 자릿수를 기준으로 10^x 값을 가지는 정수(integer).
- borrowCollateralFactor: 계정이 담보 가치를 기준으로 차입할 수 있는 비율(이자 포함)을 나타내는 차입 담보 계수.
반환값은 10^{18} 배로 스케일된 소수를 나타내는 정수 값이다.
예: 6500000000000000000 → 65%.
- liquidateCollateralFactor: 계정이 청산되기 전에 차입할 수 있는 담보 가치 비율(이자 포함)을 나타내는 청산 담보 계수.
반환값은 10^{18} 배로 스케일된 소수를 나타내는 정수 값이다.
예: 8500000000000000000 → 85%.
- liquidationFactor: 청산 시 적용되는 계수로, 10^{18} 배로 스케일된 소수를 나타내는 정수 값이다.

예: 93000000000000000000 → 7%의 청산 벌금을 의미한다.

- supplyCap: 담보 자산의 공급 한도를 나타내는 정수 값으로, 자산의 소수 자릿수를 기준으로 10^x 배로 스케일된다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
AssetInfo info = comet.getAssetInfoByAddress(0xAsset);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const infoObject = await  
comet.callStatic.getAssetInfoByAddress('0xAsset');
```

가격 조회(Get Price)

프로토콜의 자산 가격은 체인링크 프라이스 피드에 의해 업데이트된다. 자산의 현재 가격을 조회하려면, 해당 자산의 프라이스 피드 컨트랙트 주소를 getPrice 함수에 전달해야 한다.

이 함수는 자산의 USD 기준 가격을 반환하며, 소수점 이하 8자리까지 표시된다.

Comet

```
function getPrice(address priceFeed) public view returns (uint128)
```

- priceFeed: 해당 자산의 체인링크 프라이스 피드 컨트랙트의 ERC-20 주소.
- RETURNS: USD 기준 가격을 반환하며, 소수점 이하 8자리를 가지는 값으로, 10^8 배로 스케일 된 부호 없는 정수.
예: 500000000000 → 자산의 가격이 \$5000 USD를 의미한다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint price = comet.getPrice(0xAssetAddress);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const price = await comet.callStatic.getPrice(usdcAddress);
```

계정 이자 및 보상 누적(Accrue Account)

이 함수는 특정 계정에 대해 이자와 보상의 수동 누적(manual accrual)을 발동한다.

Comet

```
function accrueAccount(address account) override external
```

- **account**: 이자와 보상을 누적시킬 계정의 주소.
- **RETURN**: 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint price = comet.accrueAccount(0xAccount);
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
await comet.accrueAccount('0xAccount');
```

프로토콜 설정 조회(Get Protocol Configuration)

이 함수는 설정자 컨트랙트에 저장된 특정 Comet 인스턴스의 설정 구조체를 반환한다.

Configurator

```
struct Configuration {  
    address governor;  
    address pauseGuardian;  
    address baseToken;  
    address baseTokenPriceFeed;  
    address extensionDelegate;  
  
    uint64 supplyKink;  
    uint64 supplyPerYearInterestRateSlopeLow;  
    uint64 supplyPerYearInterestRateSlopeHigh;  
    uint64 supplyPerYearInterestRateBase;  
    uint64 borrowKink;  
    uint64 borrowPerYearInterestRateSlopeLow;  
    uint64 borrowPerYearInterestRateSlopeHigh;  
    uint64 borrowPerYearInterestRateBase;  
    uint64 storeFrontPriceFactor;  
    uint64 trackingIndexScale;  
    uint64 baseTrackingSupplySpeed;  
    uint64 baseTrackingBorrowSpeed;  
    uint104 baseMinForRewards;  
    uint104 baseBorrowMin;
```

```

uint104 targetReserves;

AssetConfig[] assetConfigs;
}

function getConfiguration(address cometProxy) external view returns
(Configuration memory)

```

- cometProxy: 설정 정보를 조회할 Comet 프록시의 주소.
- RETURNS: 프로토콜 설정 정보를 반환한다.
 - governor: 프로토콜 거버너의 주소.
 - pauseGuardian: 프로토콜 일시정지 관리자의 주소.
 - baseToken: 프로토콜의 기초 자산 스마트 컨트랙트 주소.
 - baseTokenPriceFeed: 프로토콜 기초 자산의 프라이스 피드 스마트 컨트랙트 주소.
 - extensionDelegate: Comet.sol에 포함되지 않은 추가 메서드를 위한 델리게이트 주소 (CometExt.sol).
 - supplyKink: 공급 측 곡선의 Kink 지점에서의 이자율 활용도.
 - supplyPerYearInterestRateSlopeLow: 공급 측 이자율 곡선의 기울기 하한.
 - supplyPerYearInterestRateSlopeHigh: 공급 측 이자율 곡선의 기울기 상한.
 - supplyPerYearInterestRateBase: 공급 측 이자율 곡선의 기준 기울기.
 - borrowKink: 차입 측 곡선의 Kink 지점에서의 이자율 활용도.
 - borrowPerYearInterestRateSlopeLow: 차입 측 이자율 곡선의 기울기 하한.
 - borrowPerYearInterestRateSlopeHigh: 차입 측 이자율 곡선의 기울기 상한.
 - borrowPerYearInterestRateBase: 차입 측 이자율 곡선의 기준 기울기.
 - storeFrontPriceFactor: 청산 시 담보 매수자에게 전달되는 벌금의 비율.
 - trackingIndexScale: 프로토콜 보상 추적 인덱스의 스케일.
 - baseTrackingSupplySpeed: 공급자에게 누적되는 프로토콜 보상의 속도.
 - baseTrackingBorrowSpeed: 차입자에게 누적되는 프로토콜 보상의 속도.
 - baseMinForRewards: 계정이 보상을 누적하기 위해 프로토콜에 공급해야 하는 최소 기초 자산 양.
 - baseBorrowMin: 허용되는 최소 차입 금액.
 - targetReserves: 프로토콜이 흡수된 담보를 더 이상 판매하지 않게 되는 준비금 한도.
 - assetConfigs: 프로토콜에서 지원되는 모든 자산 구성의 배열.

Solidity

```

Configurator configurator = Configurator(0xConfiguratorAddress);
Configuration config = configurator.getConfiguration(0xCometProxy);

```

Ethers.js v5.x

```
const configurator = new ethers.Contract(contractAddress, abiJson,
provider);
const config = await
configurator.callStatic.getConfiguration('0xCometProxy');
```

코멧 팩토리 조회(Get Comet Factory)

이 함수는 코멧 팩토리 컨트랙트의 주소를 조회한다.

Configurator

```
function factory(address cometProxy) public view returns (address
cometFactory)
```

- cometProxy: Comet 프록시 컨트랙트의 주소.
- RETURNS: 지정된 Comet 인스턴스에 대한 코멧 팩토리 컨트랙트의 주소.

Solidity

```
Configurator configurator = Configurator(0xConfiguratorAddress);
CometFactory factory = configurator.factory(0xCometProxy);
```

Ethers.js v5.x

```
const configurator = new ethers.Contract(contractAddress, abiJson,
provider);
const factoryAddress = await
configurator.factory.getConfiguration('0xCometProxy');
```

기초 자산 마켓 정보 조회(Get Base Asset Market Information)

이 함수는 프로토콜 마켓의 현재 주요 매개변수 값들을 조회한다.

Comet

```

struct TotalsBasic {
    uint64 baseSupplyIndex;
    uint64 baseBorrowIndex;
    uint64 trackingSupplyIndex;
    uint64 trackingBorrowIndex;
    uint104 totalSupplyBase;
    uint104 totalBorrowBase;
    uint40 lastAccrualTime;
    uint8 pauseFlags;
}

```

```

function totalsBasic() public override view returns (TotalsBasic memory)

```

- RETURNS: CometStorage.sol에 정의된 TotalsBasic 구조체 형태로 기초 자산 마켓 정보를 반환한다.
- baseSupplyIndex: 공급자에게 누적된 이자를 계산하기 위한 전역 기초 자산 공급 인덱스 (global base supply index).
- baseBorrowIndex: 차입자가 지불해야 할 이자를 계산하기 위한 전역 기초 자산 차입 인덱스 (global base borrow index).
- trackingSupplyIndex: 기초 자산을 공급하는 계정들의 참여도를 추적하는 전역 인덱스 (global index).
- trackingBorrowIndex: 기초 자산을 차입하는 계정들의 참여도를 추적하는 전역 인덱스.
- totalSupplyBase: 프로토콜에 현재 공급된 기초 자산의 총량으로, 기초 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 부호 없는 정수.
- totalBorrowBase: 프로토콜에서 현재 차입된 기초 자산의 총량으로, 기초 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 부호 없는 정수.
- lastAccrualTime: 프로토콜의 이자 누적이 전역적으로 마지막으로 계산된 시점으로, 유닉스 에 포크 이후 초 단위의 블록 타임스탬프.
- pauseFlags: 데이터 저장 효율성을 위해 패킹된 프로토콜 기능 일시정지 상태를 나타내는 정수 플래그. 자세한 내용은 [Pause Protocol Functionality](#) 항목을 참고하길 바란다.

Solidity

```

Comet comet = Comet(0xCometAddress);
TotalsBasic tb = comet.totalsBasic();

```

Ethers.js v5.x

```

const comet = new ethers.Contract(contractAddress, abiJson, provider);
const [ baseSupplyIndex, baseBorrowIndex, trackingSupplyIndex,
trackingBorrowIndex, totalSupplyBase, totalBorrowBase, lastAccrualTime,
pauseFlags ] = await comet.callStatic.totalsBasic();

```

기초 자산 누적 스케일 조회(Get Base Accrual Scale)

이 함수는 기초 자산의 보상 및 이자 누적에 사용되는 스케일 값을 조회한다.

Comet

```
function baseAccrualScale() override external pure returns (uint64)
```

- RETURNS: 기초 자산 누적 값을 소수 단위로 계산할 때 스케일을 축소하는 데 사용되는 정수 값.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint baseAccrualScale = comet.baseAccrualScale();
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const baseAccrualScale = await comet.callStatic.baseAccrualScale();
```

기초 자산 인덱스 스케일 조회(Get Base Index Scale)

이 함수는 기초 자산 인덱스의 스케일 값을 조회한다.

Comet

```
function baseIndexScale() override external pure returns (uint64)
```

- RETURNS: 인덱스를 소수 단위로 계산할 때 스케일을 축소하는 데 사용되는 정수 값.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint baseIndexScale = comet.baseIndexScale();
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const baseIndexScale = await comet.callStatic.baseIndexScale();
```

계수 스케일 조회(Get Factor Scale)

이 함수는 프로토콜의 모든 계수, 예를 들어 차입 담보 계수 등에 사용되는 스케일 값을 조회한다.

Comet

```
function factorScale() override external pure returns (uint64)
```

- RETURNS: 계수를 소수 단위로 계산할 때 스케일을 축소하는 데 사용되는 정수 값.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint factorScale = comet.factorScale();
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const factorScale = await comet.callStatic.factorScale();
```

가격 스케일 조회(Get Price Scale)

이 함수는 프로토콜 내 USD 기준 가격에 사용되는 스케일(scale) 정수 값을 조회한다. 예: 8 decimals = 1e8

Comet

```
function priceScale() override external pure returns (uint64)
```

- RETURNS: 가격 값을 소수 단위로 계산할 때 스케일을 축소하는 데 사용되는 정수 값.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint priceScale = comet.priceScale();
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const priceScale = await comet.callStatic.priceScale();
```

최대 자산 수 조회(Get Max Assets)

이 함수는 컴파운드 III에서 동시에 지원할 수 있는 자산의 최대 개수를 조회한다.

Comet

```
function maxAssets() override external pure returns (uint8)
```

- RETURNS: 컴파운드 III에서 동시에 지원할 수 있는 자산의 최대 개수.

Solidity

```
Comet comet = Comet(0xCometAddress);  
uint maxAssets = comet.maxAssets();
```

Ethers.js v5.x

```
const comet = new ethers.Contract(contractAddress, abiJson, provider);  
const maxAssets = await comet.callStatic.maxAssets();
```

일괄 작업(Bulk Actions)

컴파운드 III 코드베이스에는 Bulker라는 외부 컨트랙트의 소스 코드가 포함되어 있으며, 이 컨트랙트는 여러 개의 Comet 함수들을 단일 트랜잭션 내에서 연속적으로 실행할 수 있도록 설계되어 있다.

Bulker 컨트랙트의 사용 사례는 다음과 같으며, 이에 국한되지 않는다:

- 담보 자산을 공급하고 동시에 기초 자산을 차입하기.
- 네이티브 EVM 토큰(예: Ether)을 직접 공급하거나 인출하기.
- 계정에 잔여 소량(dust)을 남기지 않고 기초 자산을 전송하거나 인출하기.

호출(Invoke)

이 함수는 호출자가 여러 개의 액션 코드(action codes)와 콜데이터(calldatas) 배열을 전달하여, 단일 트랜잭션 내에서 순차적으로 하나씩 실행할 수 있도록 한다.

Bulker

```
/// @notice The action for supplying an asset to Comet  
bytes32 public constant ACTION_SUPPLY_ASSET = "ACTION_SUPPLY_ASSET";  
  
/// @notice The action for supplying a native asset (e.g. ETH on  
Ethereum mainnet) to Comet  
bytes32 public constant ACTION_SUPPLY_NATIVE_TOKEN =  
"ACTION_SUPPLY_NATIVE_TOKEN";  
  
/// @notice The action for transferring an asset within Comet  
bytes32 public constant ACTION_TRANSFER_ASSET = "ACTION_TRANSFER_ASSET";  
  
/// @notice The action for withdrawing an asset from Comet
```

```

bytes32 public constant ACTION_WITHDRAW_ASSET = "ACTION_WITHDRAW_ASSET";

/// @notice The action for withdrawing a native asset from Comet
bytes32 public constant ACTION_WITHDRAW_NATIVE_TOKEN =
"ACTION_WITHDRAW_NATIVE_TOKEN";

/// @notice The action for claiming rewards from the Comet rewards
contract
bytes32 public constant ACTION_CLAIM_REWARD = "ACTION_CLAIM_REWARD";

function invoke(bytes32[] calldata actions, bytes[] calldata data)
external payable

```

- **actions:** 컨트랙트에 정의된 액션에 대응하는 bytes32 문자열의 배열.
- **data:** invoke 트랜잭션에서 호출될 각 액션에 대한 콜데이터 배열.
 - 자산 공급, 자산 인출, 자산 전송
 - **comet:** 상호작용할 Comet 인스턴스의 주소.
 - **to:** 프로토콜 내부 또는 외부의 목적지 주소.
 - **asset:** ERC-20 자산 컨트랙트의 주소.
 - **amount:** 자산 컨트랙트의 “decimals” 자릿수만큼 10의 거듭제곱으로 스케일된 부호 없는 정수 형태의 자산 수량.
 - 네이티브 토큰 공급, 네이티브 토큰 인출(예: 이더리움 메인넷의 ETH)
 - **comet:** 상호작용할 Comet 인스턴스의 주소.
 - **to:** 프로토콜 내부 또는 외부의 목적지 주소.
 - **amount:** 네이티브 EVM 토큰의 소수 자릿수(정밀도)에 맞춰 10의 거듭제곱으로 스케일된 부호 없는 정수 형태의 네이티브 토큰 수량.
- **RETURN:** 반환값 없음. 오류 발생 시 트랜잭션이 취소된다.

Solidity

```

Bulkier bulkier = Bulkier(0xBulkierAddress);
// ERC-20 `approve` the bulkier. Then Comet `allow` the bulkier to be a
manager before calling `invoke`.
bytes memory supplyAssetCalldata = (abi.encode('0xCometAddress',
'0xAccount', '0xAsset', amount));
bulkier.invoke([ 'ACTION_SUPPLY_ASSET' ], [ supplyAssetCalldata ]);

```

Ethers.js v5.x

```
const bulker = new ethers.Contract(contractAddress, abiJson, provider);
// ERC-20 `approve` the bulker. Then Comet `allow` the bulker to be a
manager before calling `invoke`.
const supplyAssetCalldata =
ethers.utils.defaultAbiCoder.encode(['address', 'address', 'address',
'uint'], ['0xCometAddress', '0xAccount', '0xAsset', amount]);
await bulker.invoke([ 'ACTION_SUPPLY_ASSET' ], [ supplyAssetCalldata ]);
```