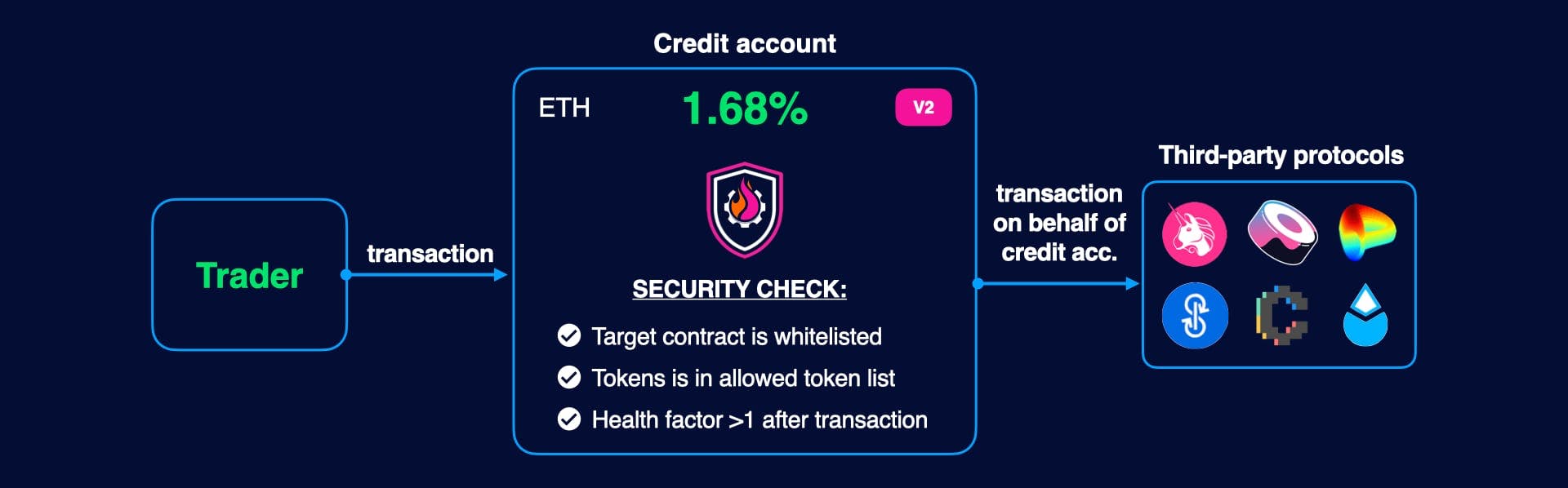
Credit accounts

Execute transactions

**Executing transactions**

**Overview**

User can consider credit account as your additional account, because all operations are executed on behalf of it.



To interact with 3rd party protocol, each it's contract should be connected with CreditManager via special contract called [Adapter](https://dev.gearbox.fi/docs/documentation/integrations/intro). This contract has the same API as original contract, so you should "prepare" your calldata and send it to adapter instead of original contract.

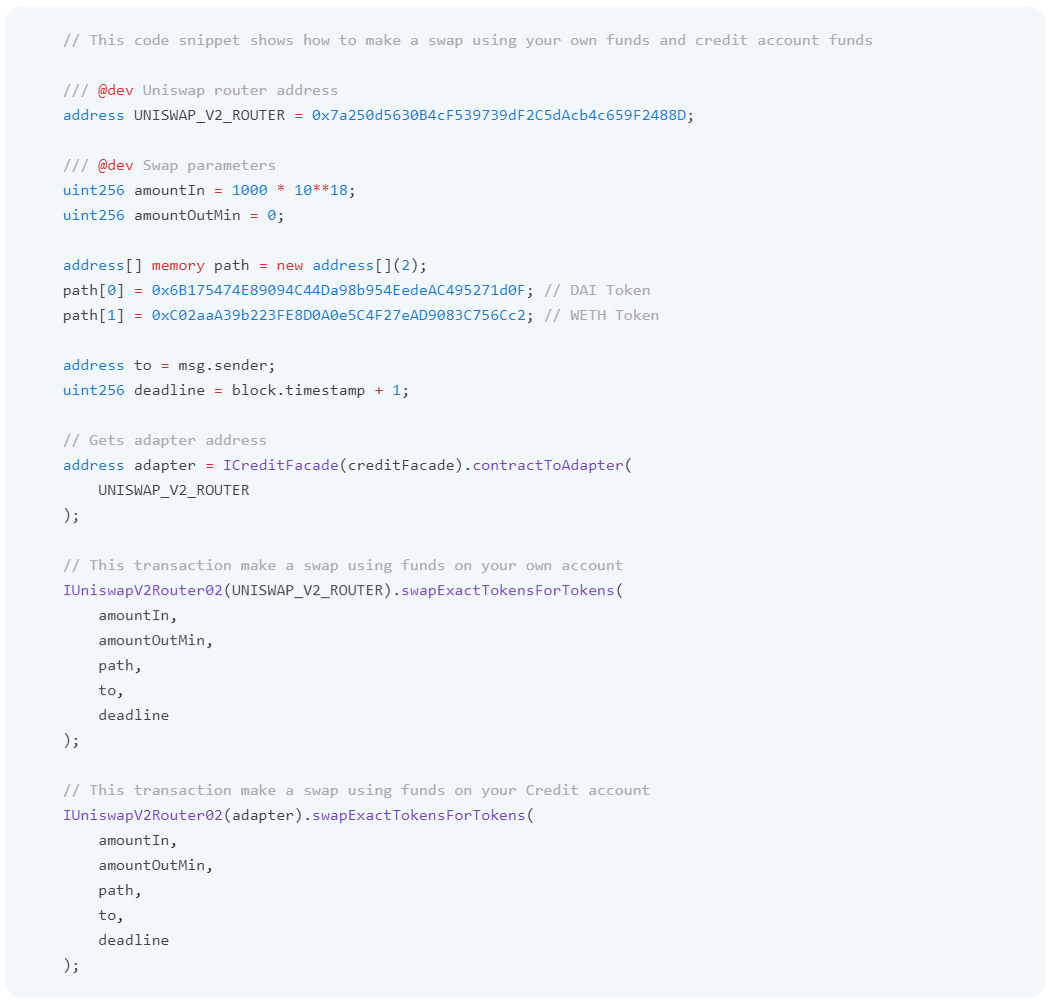
Transaction would be reverted if:

* If token which would be sent as result is forbidden
* If health factor after operation becomes < 1

**How to make a transaction**

To make a transaction, developer should get address of particular adapter and make the same call as with original contract.

Solidity



Credit accounts

Multicall

# Multicall

## Overview

Multicall is a new feature in Gearbox V2, which allows users to execute a batch of adapter or Credit Facade calls in one transaction. There is a number of advantages to using multicalls as opposed to individual adapter calls:

* **Gas efficiency**  
  Multicall only performs one collateral check after executing all actions, compared to a fastCollateralCheck or fullCollateralCheck being invoked each time after separate calls. As health checks are some of the most expensive actions in the system, this saves a lot of gas.
* **One-click strategies**  
  Multicalls can be used while opening, closing, or liquidating accounts, or run separately to manage an existing account. From the standpoint of a user, very complex levered strategies can be entered or exited with a single transaction signing.
* **Low-cost liquidations**  
  Instead of using flash loans, receiving collateral and then selling it (which is how liquidations are typically performed in overcollateralized systems), multicalls enable the liquidator to take temporary control of the account, convert collateral to underlying within the system, and then repay the loan. This flow significantly reduces liquidation costs, as borrowing/repaying a flash loan and sending a potentially large number of assets to the liquidator are no longer required.

## Building a multicall

To build a multicall, the developer would prepare an array of Multicall structs, providing the list of call targets and corresponding calldata. They would then pass the array to one of the functions in CreditFacade that supports multicalls:

struct MultiCall {

address target;

bytes callData;

}

/// @dev Opens credit account and run a bunch of transactions for multicall

/// - Opens credit account with desired borrowed amount

/// - Executes multicall functions for it

/// - Checks that the new account has enough collateral

/// - Emits OpenCreditAccount event

///

/// @param borrowedAmount Debt size

/// @param onBehalfOf The address that we open credit account. Same as msg.sender if the user wants to open it for his own wallet,

/// or a different address if the beneficiary is a different wallet

/// @param calls Multicall structure for calls. Basic usage is to place addCollateral calls to provide collateral in

/// assets that differ than undelyring one

/// @param referralCode Referral code which is used for potential rewards. 0 if no referral code provided

function openCreditAccountMulticall(

uint256 borrowedAmount,

address onBehalfOf,

MultiCall[] calldata calls,

uint256 referralCode

) external payable;

/// @dev Run a bunch of transactions for multicall and then close credit account

/// - Wraps ETH to WETH and sends it msg.sender is value > 0

/// - Executes multicall functions for it (the main function is to swap all assets into undelying one)

/// - Close credit account:

/// + It checks underlying token balance, if it > than funds need to be paid to pool, the debt is paid

/// by funds from creditAccount

/// + if there is no enough funds in credit Account, it withdraws all funds from credit account, and then

/// transfers the diff from msg.sender address

/// + Then, if sendAllAssets is true, it transfers all non-zero balances from credit account to address "to"

/// + If convertWETH is true, the function converts WETH into ETH on the fly

/// - Emits CloseCreditAccount event

///

/// @param to Address to send funds during closing contract operation

/// @param skipTokenMask Tokenmask contains 1 for tokens which needed to be skipped for sending

/// @param convertWETH It true, it converts WETH token into ETH when sends it to "to" address

/// @param calls Multicall structure for calls. Basic usage is to place addCollateral calls to provide collateral in

/// assets that differ than undelyring one

function closeCreditAccount(

address to,

uint256 skipTokenMask,

bool convertWETH,

MultiCall[] calldata data

) external payable;

/// @dev Run a bunch of transactions (multicall) and then liquidate credit account

/// - Wraps ETH to WETH and sends it msg.sender (liquidator) is value > 0

/// - It checks that hf < 1, otherwise it reverts

/// - It computes the amount which should be paid back: borrowed amount + interest + fees

/// - Executes multicall functions for it (the main function is to swap all assets into undelying one)

/// - Close credit account:

/// + It checks underlying token balance, if it > than funds need to be paid to pool, the debt is paid

/// by funds from creditAccount

/// + if there is no enough funds in credit Account, it withdraws all funds from credit account, and then

/// transfers the diff from msg.sender address

/// + Then, if sendAllAssets is false, it transfers all non-zero balances from credit account to address "to".

/// Otherwise no transfers would be made. If liquidator is confident that all assets were transffered

/// During multicall, this option could save gas costs.

/// + If convertWETH is true, the function converts WETH into ETH on the fly

/// - Emits LiquidateCreditAccount event

///

/// @param to Address to send funds during closing contract operation

/// @param skipTokenMask Tokenmask contains 1 for tokens which needed to be skipped for sending

/// @param convertWETH It true, it converts WETH token into ETH when sends it to "to" address

/// @param calls Multicall structure for calls. Basic usage is to place addCollateral calls to provide collateral in

/// assets that differ than undelyring one

function liquidateCreditAccount(

address borrower,

address to,

uint256 skipTokenMask,

bool convertWETH,

MultiCall[] calldata data

) external payable;

/// @dev Executes a bunch of transactions and then make full collateral check:

/// - Wraps ETH and sends it back to msg.sender address, if value > 0

/// - Execute bunch of transactions

/// - Check that hf > 1 ather this bunch using fullCollateral check

/// @param calls Multicall structure for calls. Basic usage is to place addCollateral calls to provide collateral in

/// assets that differ than undelyring one

function multicall(MultiCall[] calldata calls)

external

payable;

### Examples

The following is an example for constructing a multicall:

Solidity



### Multicall implementation

There are several essential steps to a multicall:

* The credit account ownership is transferred to the CreditFacade, as adapters generally locate the CA owner by msg.sender;
* Calls are executed sequentially;
* A full collateral check is performed;
* Account ownership is returned to the original owner;

### Functions supported in multicalls

During a multicall, the following functions can be called:

* Any functions in adapters allowed within a CreditManager (**note:** the adapter address must be passed as the target, instead of the original contract);
* A number of CreditFacade functions: addCollateral, increaseDebt, decreaseDebt, enableToken;
* ICreditFacadeBalanceChecker.revertIfBalanceLessThan: (see more in a section below);

### Multicall slippage protection

A signature revertIfBalanceLessThan(address token, uint256 minBalance) is defined within the ICreditFacadeBalanceChecker interface.

While this function has no formal implementation, it can be encoded as calldata and passed to CreditFacade to protect from slippage. Upon receiving this call, CreditFacade will check whether the balance of token is at least minBalance, and revert if not.

Since multicalls support arbitrarily complex strategies, the call can be made at any point during a multicall and for any token, allowing the developer fine control over slippage protection.

### Restrictions

* It is forbidden to increase and then decrease debt within one multicall;
* It is forbidden to decrease debt if a multicall is a part of openCreditAccountMulticall;
* All creditFacade functions are forbidden during closure / liquidation multicalls.

Credit accounts

Adding collateral

# Adding collateral

We strongly recommend to use addCollateral method in CreditFacade contract instead of tranferring tokens directly to credit account. This method is designed for your safety, let's check advantages:

* It works with borrower address instead of credit account address, so it would be reverted, if borrower closed his account or such account was liquidated
* It enables token, if such a token wasn't enabled before

## Method

### Normal call

function addCollateral( address onBehalfOf, address token, uint256 amount ) external payable;

| **Parameter** | **Description** |
| --- | --- |
| onBehalfOf | Address of borrower to add funds |
| token | Token address, it should be whitelisted on CreditManagert, otherwise it reverts |
| amount | Amount to add |

### Adding collateral in multicall

During multicall it's possible to addCollateral:

