# Developer Docs

# GHO is for builders

Learn GHO Integrate GHO

### Quick Links

- Deployed GHO Contracts
- GHO Contract Source Code
- Deployed Aave Protocol Contracts >
- Aave V3 Contract Source Code
- GHO JavaScript SDK
- Solidity and JavaScript Address Registry >
- GHO Subgraph Ethereum Mainnet

## Learn GHO

- a.Aave V3 Ethereum Pool Facilitator
- 1. Mint
- 1. Repay
- 1. Liquidate
- b.FlashMinter Facilitator
- c.Discount Dynamics

GHO is an ERC20 token minted from contracts designated asFacilitators . It has been proposed to the Aave community that theAave V3 Ethereum Pool andFlashMinter Facilitator will be the first Facilitators of GHO.

The Aave Pool facilitator will facilitate the minting and burning of GHO through the standardborrow andrepay function which are utilizaed by all other Aave reserves. The flashminter facilitator will provide flashloan functionality to GHO which is not available by default.

A Facilitator can trustlessly mint and burn GHO tokens through various strategies. The Aave DAO assigns each Facilitator a Bucket with a specified Capacity, which is the upward limit of GHO that a specific Facilitator can mint. This limit is defined and can be changed by the Aave DAO.

### Aave V3 Ethereum Pool Facilitator

Interacting with GHO via the Aave Pool Facilitator is very similar to interacting with a typical reserve asset. Below are the technical guides for all GHO actions along with their contract references.

### Minting

Minting occurs through theborrow() function of the Pool where GHO is listed. To mint GHO, the process is identical to borrowing any other reserve. To mint, an address must have sufficient collateral which is performed by approving and then callingsupply() on the Aave Pool with an eligible collateral asset. Once an address has sufficient collateral, it is able to borrow up to a maximum collateral factor determined by it's collateral asset composition.

GHO can be added to an eMode category, which modifies the collateral factor and liquidation threshold. This can be queried with the following fields from theintegrating GHO section.

reserve.eModeCategoryId : id of eMode category reserve belongs touser.userEModeCategoryId : user's active eMode category

Since GHO is created and not borrowed from suppliers, GHO is not subject to restrictions on available liquidity, and instead, the Facilitator cap and collateralization requirements define the limits to which GHO can be minted as calculated below.

availableFacilitatorCap = ghoReserveData.aaveFacilitatorButcketMaxCapacity - ghoReserveData.aaveFacilitatorBucketLevel

Seecore functions for GHO borrowing integrations.

### Repay

GHO isrepaid just like any other asset, by approving the Pool contract to spend GHO tokens (by approval transaction orsigned permit andrepayWithPermit ).

What is different about GHO is the calculation of accrued interest. See thediscount dynamics section for more info on how accrued interest affects balances for repayment.

Seecore functions for GHO repay integrations.

### Liquidation

When an address has a GHO borrow position, they are eligible to be liquidated under the same conditions as any other collateralized address. If the health factor of a GHO borrow falls below one, which occurs when the sum of borrow value exceeds the weighted average of liquidation thresholds of collateral assets, then any address is eligible to make aliquidationCall on the Pool contract.

TheliquidationCall repays up to 100% of the GHO borrow position in exchange for an equivalent USD valuation of the collateral plus a liquidation bonus.

See the developersliquidation guide for more information.

### FlashMinter Facilitator

Since GHO is not borrowed like a typical Aave reserve, a separate Facilitator is used in place to replicate theflashloan functionality of the Aave Pool.

The FlashMinter Facilitator has a separate minting cap from the Aave Pool. Since all FlashMint transactions are returned in a single transaction, no GHO is ever minted against this Facilitator and the cap is applied to each transaction.

FlashMint is useful for a variety of applications such as liquidations, debt swaps, and peg arbitrage. More details on this Facilitator can be foundhere .

### Discount Dynamics

Thediscount rate strategy contract defines the parameters of a user's discount. The strategy can be updated by governance and the key parameters are a maximum discount percent (such as 20%), a discount token (such as stkAAVE), and an amount of gho borrowed at a discounted percent per discount token owned (such as 100 GHO per 1 stkAAVE).

The discount is not applied continuously as a GHO borrower accrues interest. Interest is compounded at the base borrow rate and the discount is applied when the borrow balance is queried by callingbalanceOf directly or from an internal call such asrepay orliquidate .

## Integrate GHO

GHO is operated on the public blockchain making it accessible to plug into any type of application.

The ability of an application to interact with the data and functionality of GHO is limited only by the connection to access the public blockchain.

Any Ethereum address can access the full functionality of GHO, and the real-time status and complete historical record of GHO transactions can be verified by anyone at any time.

This guide explains the best practices for integrating GHO into a variety of common application types.

1. Types of Integrations
2. 
    - a.Smart Contracts
3. 
    - b.Frontend
4. 
    - c.Data Analytics
5. GHO SDK
6. 
    - a.setup
7. 
    - b.borrow
8. 
    - c.repay
9. 
    - d.FlashMint

10.
    - e.Interface Live Data
11. Data
12.
    - a.Live Data
13.
    - b.Historical Data

# Types of Integrations

GHO can be integrated into virtually any application because all data and functionality occur through a public blockchain. This guide will give a tutorial on best practices for the most common categories of integration types

### Smart Contracts

Check out thedeployed GHO contracts to get started with contract integrations.

### Frontend

Aave Utilities is a JavaScript SDK that can be used to greatly simplify the process of integrating GHO data and functionality.

It provides imports for fetching data, formatting data, and building transactions. Tutorials with sample code for utilizaing this SDK can be foundhere

### Data Analytics

The current state of GHO, and the complete historical record data are accessible to any application with blockchain querying capabilities. The contract addresses and ABIs for GHO contracts are available in the contract docs.

Thedata section goes into detail about integrating the most common live and historical data queries for JavaScript and Python applications.

# GHO SDK

### setup

TheAave Utilities Javascript SDK is a utilty for fetching data and building transactions. Since GHO is under active development, it is not available in the main package versions. GHO versions are published fromthis branch and the current version in use on theGHO testnet app are listed below:

npm install @aave/[email protected] npm install @aave/[email protected] or

yarn add @aave/[email protected] yarn add @aave/[email protected]

### borrow

GHO is minted through theborrow method of the Aave Pool facilitator which can be accessed from thePool import of the contract-helpers package. The pre-requisite toborrow is that the address incurring the borrow position must have sufficient collateral. This is achieved by the caller supplying collateral or if the caller has been approved a credit delegation and passes in the delegators address in theonBehalfOf field.

Sample Code (JavaScript) import

{

Pool ,

InterestRate

}

from

"@aave/contract-helpers" ;

const pool =

new

Pool ( provider ,

{ POOL :

"0x3De59b6901e7Ad0A19621D49C5b52cC9a4977e52" ,

// Goerli GHO market WETH_GATEWAY :

"0x9c402E3b0D123323F0FCed781b8184Ec7E02Dd31" ,

// Goerli GHO market } ) ;

/ - *@param user The ethereum address that will receive the borrowed amount - @paramreserve The ethereum address of the reserve asset - @param amount The amount to be borrowed, in human readable units (e.g. 2.5 ETH) - @param interestRateMode//Whether the borrow will incur a stable (InterestRate.Stable) or variable (InterestRate.Variable) interest rate - @param @optional debtTokenAddress The ethereum address of the debt token of the asset you want to borrow. Only needed if the reserve is ETH mock address - @param @optional onBehalfOf The ethereum address for which user is borrowing. It will default to the user address* / const

txs :

EthereumTransactionTypeExtended [ ]

=

await pool . borrow ( { user , reserve :

"0xcbE9771eD31e761b744D3cB9eF78A1f32DD99211" ,

// Goerli GHO market amount , interestRateMode , debtTokenAddress :

"0x80aa933EfF12213022Fd3d17c2c59C066cBb91c7" ,

// Goerli GHO market onBehalfOf , referralCode , } ) ; Submit transaction usingthese instructions .

## repay

Repay also occurs seamlessly through therepay function of the Aave V3 Pool contract. Since repay transfers assets, the Pool must be approved as a spender of the tokens. There are two routes for approval:

- Approve via txn -> repay
- Approve via signed message -> repayWithPermit

There are instructions for both routes below:

Repay Sample Code (JavaScript) import

{

Pool

}

from

"@aave/contract-helpers" ;

const pool =

new

Pool ( provider ,

{ POOL :

"0x3De59b6901e7Ad0A19621D49C5b52cC9a4977e52" ,

// Goerli GHO market WETH_GATEWAY :

"0x9c402E3b0D123323F0FCed781b8184Ec7E02Dd31" ,

// Goerli GHO market } ) ;

/ - @param *user* The ethereum address that will make the deposit - @param*reserve* The ethereum address of the reserve - @param *amount* The amount to be deposited - @param*interestRateMode* // Whether stable (InterestRate.Stable) or variable (InterestRate.Variable) debt will be repaid - @param @optional *onBehalfOf* The ethereum address for which user is depositing. It will default to the user address / const

txs :

EthereumTransactionTypeExtended [ ]

=

await pool . repay ( { user , reserve :

"0xcbE9771eD31e761b744D3cB9eF78A1f32DD99211" ,

// Goerli GHO market amount , interestRateMode , onBehalfOf , } ) ; Will return an array with repay transaction and optionally an approval transaction. Submit transaction(s) usingthese instructions . RepayWithPermit Sample Code (JavaScript) import

{

Pool

}

from

'@aave/contract-helpers' ;

const pool =

new

Pool ( provider ,

{ POOL :

'0x3De59b6901e7Ad0A19621D49C5b52cC9a4977e52' ,

// Goerli GHO market WETH_GATEWAY :

'0x9c402E3b0D123323F0FCed781b8184Ec7E02Dd31' ,

// Goerli GHO market } ) ;

// Generate payload to be signed by user const approvalMsg =

await pool . signERC20Approval ( { user , reserve , amount , deadline ,

// determined by you ) } )

// User signs with wallet method such as ethers signTypedDataV4 and passed as signature variable in next call

/ - @param *user* The ethereum address that will make the deposit - @param*reserve* The ethereum address of the reserve - @param *amount* The amount to be deposited - @param*interestRateMode* // Whether stable (InterestRate.Stable) or variable (InterestRate.Variable) debt will be repaid - @param @optional *onBehalfOf* The ethereum address for which user is depositing. It will default to the user address / const

txs :

EthereumTransactionTypeExtended [ ]

=

await pool . repayWithPermit ( { user , reserve :

'0xcbE9771eD31e761b744D3cB9eF78A1f32DD99211' ,

// Goerli GHO market amount , interestRateMode , onBehalfOf , signature , } ) ; Will return an array with repay transaction and optionally an approval transaction. Submit transaction(s) usingthese instructions .

## flashmint

Coming soon to GHO SDK. In the meantime, can be accessed by following thecontracts documentation .

## Interface Live Data

There are several view contracts used to provide a summarized data source for market, incentives, and gho data. The utilities SDK exposes helper methods to fetch and format data from these contracts. To setup these packages see thesetup section.

Sample Code (JavaScript) import

{ ethers }

from

"ethers" ; import

{ dayjs }

from

"dayjs" ; import

{ UiPoolDataProvider , UiIncentiveDataProvider , ChainId , GhoService , }

from

"@aave/contract-helpers" ; import

{ formatGhoReserveData , formatGhoUserData , formatReservesAndIncentives , formatUserSummaryAndIncentives , }

from

"@aave/math-utils" ;

// ES5 Alternative imports // const { // ChainId, // UiIncentiveDataProvider, // UiPoolDataProvider, // GhoService, // } = require('@aave/contract-helpers'); // const { // formatGhoReserveData, // formatGhoUserData, // formatReservesAndIncentives, // formatUserSummaryAndIncentives, // } = require('@aave/math-utils'); // const ethers = require('ethers');

// Sample RPC address for querying ETH goerli const provider =

new

ethers . providers . JsonRpcProvider ( "https://eth-goerli.public.blastapi.io" ) ;

// User address to fetch data for, insert address here const currentAccount =

"0x464C71f6c2F760DdA6093dCB91C24c39e5d6e18c" ;

// View contract used to fetch all reserves data (including market base currency data), and user reserves // Using Aave V3 Eth goerli address for demo const poolDataProviderContract =

new

UiPoolDataProvider ( { uiPoolDataProviderAddress :

"0x3De59b6901e7Ad0A19621D49C5b52cC9a4977e52" ,

// Goerli GHO Market provider , chainId :

ChainId . goerli , } ) ; const currentTimestamp =

dayjs ( ) . unix ( ) ;

// View contract used to fetch all reserve incentives (APRs), and user incentives // Using Aave V3 Eth goerli address for demo const incentiveDataProviderContract =

new

UiIncentiveDataProvider ( { uiIncentiveDataProviderAddress :

"0xF67B25977cEFf3563BF7F24A531D6CEAe6870a9d" ,

```
// Goerli GHO Market provider , chainId :

ChainId . goerli , } ) ;

const ghoService =

new

GhoService ( { provider , uiGhoDataProviderAddress :

"0xE914D574975a1Cd273388035Db4413dda788c0E5" ,

// Goerli GHO Market } ) ;

async

function

fetchContractData ( )

{ // Object containing array of pool reserves and market base currency data // { reservesArray, baseCurrencyData } const reserves =

await poolDataProviderContract . getReservesHumanized ( { lendingPoolAddressProvider :

"0x4dd5ab8Fb385F2e12aDe435ba7AFA812F1d364D0" ,

// Goerli GHO Market } ) ;

// Object containing array or users aave positions and active eMode category // { userReserves, userEmodeCategoryId } const userReserves =

await poolDataProviderContract . getUserReservesHumanized ( { lendingPoolAddressProvider :

"0x4dd5ab8Fb385F2e12aDe435ba7AFA812F1d364D0" ,

// Goerli GHO Market user : currentAccount , } ) ;

// Array of incentive tokens with price feed and emission APR const reserveIncentives = await incentiveDataProviderContract . getReservesIncentivesDataHumanized ( { lendingPoolAddressProvider :

"0x4dd5ab8Fb385F2e12aDe435ba7AFA812F1d364D0" ,

// Goerli GHO Market } ) ;

// Dictionary of claimable user incentives const userIncentives = await incentiveDataProviderContract . getUserReservesIncentivesDataHumanized ( { lendingPoolAddressProvider :

"0x4dd5ab8Fb385F2e12aDe435ba7AFA812F1d364D0" ,

// Goerli GHO Market user : currentAccount , } ) ;

const ghoReserveData =

await ghoService . getGhoReserveData ( ) ; const ghoUserData =

await ghoService . getGhoUserData ( currentAccount ) ;

const formattedGhoReserveData =

formatGhoReserveData ( { ghoReserveData , } ) ; const formattedGhoUserData =

formatGhoUserData ( { ghoReserveData , ghoUserData , currentTimestamp , } ) ;

const formattedPoolReserves =

formatReservesAndIncentives ( { reserves : reserves . reservesData , currentTimestamp , marketReferenceCurrencyDecimals : reserves . baseCurrencyData . marketReferenceCurrencyDecimals , marketReferencePriceInUsd : reserves . baseCurrencyData . marketReferenceCurrencyPriceInUsd , reserveIncentives : reserveIncentives , } ) ;

const userSummary =

formatUserSummaryAndIncentives ( { currentTimestamp , marketReferencePriceInUsd : reserves . baseCurrencyData .
```

marketReferenceCurrencyPriceInUsd , marketReferenceCurrencyDecimals : reserves . baseCurrencyData . marketReferenceCurrencyDecimals , userReserves : userReserves . userReserves , formattedReserves : formattedPoolReserves , userEmodeCategoryId : userReserves . userEmodeCategoryId , reserveIncentives : reserveIncentives , userIncentives : userIncentives , } ) ;

let formattedUserSummary = userSummary ; // Factor discounted GHO interest into cumulative user fields if

( formattedGhoUserData . userDiscountedGhoInterest

0 )

{ const userSummaryWithDiscount =

formatUserSummaryWithDiscount ( { userGhoDiscountedInterest : formattedGhoUserData . userDiscountedGhoInterest , user , marketReferenceCurrencyPriceUSD :

Number ( formatUnits ( reserves . baseCurrencyData . marketReferenceCurrencyPriceInUsd , USD_DECIMALS ) ) , } ) ; formattedUserSummary =

{ ... userSummary , ... userSummaryWithDiscount , } ; }

console . log ( { formattedGhoReserveData , formattedGhoUserData , formattedPoolReserves , formattedUserSummary , } ) ; }

fetchContractData ( ) ;

# Data

All facilitator transactions occur through smart contracts, so querying the real-time state of facilitator and market data is possible through contract queries. The following guides will give sample code fetch live and historical data for GHO directly from contract queries and events.

## Live Data

To query live data for GHO, there are several view contracts which can give summarized information of markets, incentives, and gho data respectively:

- UiPoolDataProvider: Queries for all market and user data
- UiIncentiveDataProvider: QUeries for all available and user claimable incentives
- UiGhoDataProvider: Queries for Aave Pool facilitator and user discount

A complete example of fetching and formatting data from these contracts can be found in the GHO SDK[ive data](#) section. These steps could also be adapted for other languages or use cases.

## Historical Data

Transactions of the GHO facilitator are queryable through events on the GHO contracts, and through other indexed data sources.

Shown below are 3 methods for generalized event queries using subgraphs, RPCs, and the Etherscan API which can all be used with the GHO deployed contracts to index historical data for GHO integrations.

Query Events Subgraph The[GHO Ethereum Mainnet Subgraph](#) is a GraphQL endpoint which indexes events and can be used to query data about GHO facilitators and interactions with Aave V3 Ethereum market. The full schema for GHO-related entities can be found[here](#) Query Events RPC Note: An archival node may be necessary for the rpcUrl to query historical events

const

{ providers ,

Contract , utils }

=

require ( "ethers" ) ; // REPLACE WITH YOUR DATA HERE const abi =

require ( "./yourAbi.json" ) ; const contractAddress =

"0x0" ; const contractStartBlock =

1 ;

// deployment block of the contract, to avoid filtering from genesis block const rpcUrl =

"https://eth-mainnet.public.blastapi.io" ;

// eth mainnet example async

function

fetchEvents ( )

{ // Connect to an Ethereum node const provider =

new

providers . JsonRpcProvider ( "rpcUrk" ) ; // Create a Contract object const contract =

new

Contract ( contractAddress , abi , provider ) ; // Define the filter options const filter =

{ fromBlock :

0 , toBlock :

"latest" , address : contractAddress , } ; // Get the events from the contract const events =

await contract . provider . getLogs ( filter ) ; // Loop through the events and log them for

( const event of events )

{ const eventJson = utils . parseLog ( event ) ; console . log ( eventJson ) ; } } fetchEvents ( ) ; Query Events Etherscan API
const ethers =

require ( "ethers" ) ; const request =

require ( "request-promise-native" ) ; // REPLACE WITH YOUR DATA HERE const abi =

require ( "./yourAbi.json" ) ; const contractAddress =

"0x0" ; const contractStartBlock =

1 ;

// deployment block of the contract, to avoid filtering from genesis block const

ETHERSCAN_API_KEY

=

"" ; async

function

fetchEvents ( )

{ // Set up the API key and base URL for the Etherscan API const baseUrl =

"https://api.etherscan.io/api" ; const contractInterface =

new

ethers . utils . Interface ( abi ) ; // Set the initial page and block number let page =

1 ; let blockNumber =

"latest" ; const

PAGE_SIZE

=

1000 ; // Loop through each transaction while

( true )

{ // Set up the options for the API request const options =

{ url :

{ baseUrl } , qs :

{ module :

"account" , action :

"txlist" , address : contractAddress , startblock : contractStartBlock , endblock : blockNumber , page : page , sort :

"asc" , apikey :

ETHERSCAN_API_KEY , } , json :

true , } ; // Send the API request const response =

await

request ( options ) ; for

( const tx of response . result )

{ // Check if this transaction was sent to the contract if

( tx . to . toLowerCase ( )

=== contractAddress . toLowerCase ( ) )

{ // Add the transaction to the array const decodedData = contractInterface . decodeFunctionData ( tx . input . slice ( 0 ,

10 ) , tx . input ) ; // Use decoded data here } } if

( response . result . length

<

PAGE_SIZE )

{ break ; }

else

{ page +=

1 ; } } } fetchEvents ( ) ;Edit this page Next ERC20 * Quick Links * Learn GHO * * Aave V3 Ethereum Pool Facilitator* * Minting * * Repay * * Liquidation * * FlashMinter Facilitator * * Discount Dynamics * Integrate GHO * Types of Integrations * * Smart Contracts * * Frontend * * Data Analytics * GHO SDK * * setup * * borrow * * repay * * flashmint * * Interface Live Data * Data * * Live Data * * Historical Data