Dai and Collateral Redemption during Emergency Shutdown

Level : Intermediate

Estimated Time: 60minutes

Description

This guide describes how users can interact with the Maker protocol through proxy contracts to redeem Dai and any excess collateral if the Maker system has entered into emergency shutdown. We will define the setup process, including proxy contract setup, followed by seth calls to; redeem collateral as a Dai Holder, and free excess collateral as a Vault Owner.

Learning Objectives

To redeem Dai and/or excess collateral in the event of Emergency Shutdown

Table of Contents

Setup Process

1. Installation
2. Contract Address Setup
3.

Dai Holders to Redeem Collateral

1. Check user Dai holdings
2. Approve a Proxy
3. Create Calldata
4. Execute Calldata using the MYPROXY Contract
5. Call cashETH or cashGEM functions
6. Using cashETH
7. Define calldata for our function
8. Execute cashETHcalldata
9. Alternative from step (6), Using cashGEM
10. Define calldata for our function
11. Call execute in MYPROXY
12.

Vault Owners to Redeem Excess Collateral

1. Vault Holder State
2. Redeeming ETH using the freeETH function
3. 2.1. Set Call Data
4. 2.2 Execute this calldata
5. Redeeming ETH using the freeGEM function
6.

3.1 Set Calldata

3.2 Execute this calldata

Conclusion

Setup process

1. Installation

In order to interface with the Ethereum blockchain, the user needs to install seth, a command line tool as part of the Dapp.Tools toolset. We also provide further installation information here . Once the user has installed and configured seth correctly to use the main Ethereum network and the address which holds their MKR, they can query contract balances, approvals and transfers.

1. Contract Address Setup

The user will require the following contract addresses, shown below as mainnet addresses. Rest of mainnet or testnet addresses are accessible at changelog.makerdao.com which can be verified on Etherscan . Similarly, additional information on the commands described below can be found in the End contract and the Proxy_Actions_End contract . These should be setup in the following manner and pasted into the terminal line by line:

```

Copy export DAI=0x6B175474E89094C44Da98b954EedeAC495271d0F export PROXY_ACTIONS_END=0x069B2fb501b6F16D1F5fE245B16F6993808f1008 export MCD_END=0xaB14d3CE3F733CACB76eC2AbE7d2fcb00c99F3d5 export CDP_MANAGER=0x5ef30b9986345249bc32d8928B7ee64DE9435E39 export PROXY_REGISTRY=0x4678f0a6958e4D2Bc4F1BAF7Bc52E8F3564f3fE4 export MCD_JOIN_ETH=0x2F0b23f53734252Bda2277357e97e1517d6B042A export MCD_JOIN_BAT=0x3D0B1912B66114d4096F48A8CEe3A56C231772cA export MCD_JOIN_DAI=0x9759A6Ac90977b93B58547b4A71c78317f391A28

export MYPROXY=(seth call PROXY_REGISTRY 'proxies(address)(address)' ETH_FROM)

## This creates a unique proxy address by calling the proxy registry using the users Ethereum address.

export ilk=(seth --to-bytes32 (seth --from-ascii ETH-A)) export ilkBAT=(seth --to-bytes32 (seth --from-ascii BAT-A))

## Here we have defined two ilk (collateral types) ETH and BAT.

## The number of ilk types needed will depend on the types of collateral vaults that the user had open.

export ETH_GAS=4000000 export ETH_GAS_PRICE=2500000000

## Typically gas costs are slightly increased when dealing with proxy contracts to prevent failed transactions.

export cdpId=(seth --to-dec (seth call CDP_MANAGER 'last(address)' MYPROXY))

## This is a call to the CDP Manager responsible for making the users CDP ID.

## Note, if user created multiple vaults they will have multiple CDP IDs, all of which must be referenced to retrieve collateral.

```

Dai holders to Redeem Collateral

There are two functions to be called in order to retrieve the end collateral. The first step ispack and the second step iscashETH orcashGem depending on the leftover amount of each collateral type in the system.

Depositing Dai tokens into the system can be done using thePROXY_ACTIONS_END contract library and thepack function. This function efficiently bundles together three parameters, including three parameters; theDai(join) adapter, theend contract and the amount of Dai tokens you wish to redeem for allowed collateral in one go.

```

Copy function pack( address daiJoin, address end, uint wad ) public { daiJoin_join(daiJoin, address(this), wad); VatLike vat = DaiJoinLike(daiJoin).vat(); // Approves the end to take out DAI from the proxy's balance in the vat if (vat.can(address(this), address(end)) == 0) { vat.hope(end); } EndLike(end).pack(wad); }

```

1. Check user Dai holdings

The user can check their Dai Token balance and subsequently save it in thewad variable so that it can be later used in the proxy function.

```

Copy export balance=(seth --from-wei (seth --to-dec (seth call DAI 'balanceOf(address)' ETH_FROM))) export wad=(seth --to-uint256 (seth --to-wei 13400 eth))

```

in the above, 13400 is an example Dai balance

```

1. Approve a Proxy

The user needs to approveMYPROXY in order to withdraw Dai from their wallet by using the following function.

```

Copy seth send DAI 'approve(address,uint)' MYPROXY (seth --to-uint256 (mcd --to-hex -1))

```

1. Create Calldata

Next it is necessary to bundle together the function definitions and parameters that the user needs to execute. This is done by preparing a function call toMYPROXY , defined ascalldata.

```

Copy export calldata=(seth calldata 'pack(address,address,uint)' MCD_JOIN_DAI MCD_END wad) . . .

```

# 0x33ef33d60000000000000000000000000fc0b3b61407cdf5f583b5b1e08514e68ecee4a730000000(

```

1. Execute calldata using theMYPROXY contract

The user is able to call theexecute function and utilize thePROXY_ACTIONS_END.pack() function within the environment ofMYPROXY . This approves the proxy to take Dai tokens from the user's wallet into the proxy address and deposits it into theend contract, where a proportionate amount of collateral can later be claimed. Once the DAI is packed, it cannot be unpacked.

```

Copy seth send MYPROXY 'execute(address,bytes memory)' PROXY_ACTIONS_END calldata

```

example transaction showing actions involved in 'packing' the user's Dai.

```

1. CallcashETH orcashGEM functions

Users will be able to withdraw collateral depending on the collateral that is in the VAT at the time of shutdown. For example 1 Dai will be able to claim a portion of ETH and BAT (and any other accepted collateral) which when combined will be approximately worth 1 USD. This process is completed by callingcashETH orcashGEM .

6.Using cashETH

The following functioncashETH is referenced as part of thecalldata function and should be referenced here .

```

Copy function cashETH(address ethJoin, address end, bytes32 ilk, uint wad) public { EndLike(end).cash(ilk, wad); uint wadC = mul(wad, EndLike(end).fix(ilk)) / RAY; // Exits WETH amount to proxy address as a token GemJoinLike(ethJoin).exit(address(this), wadC); // Converts WETH to ETH GemJoinLike(ethJoin).gem().withdraw(wadC); // Sends ETH back to the user's wallet msg.sender.transfer(wadC); }

```

1. Define calldata for our function

Next, we again define the calldata for our function by bundling together thecashETH parameters shown above.

```

Copy export cashETHcalldata=(seth calldata 'cashETH(address,address,bytes32,uint)' MCD_JOIN_ETH MCD_END ilk wad)

```

1. ExecutecashETHcalldata

Finally, executing thecashETHcalldata in theexecute function of the user'sMYPROXY contract will redeem ETH for DAI, and place this ETH into the user's ETH wallet.

```

Copy seth send MYPROXY 'execute(address,bytes memory)' PROXY_ACTIONS_END cashETHcalldata

```

# example successful transaction

```

1. Alternative from step (6), UsingcashGEM

It is also possible to use thecashGEM function in order to redeem different collateral types. In the below example we are referencing gemJoin as it relates to BAT.

```

Copy function cashGem( address gemJoin, address end, bytes32 ilk, uint wad ) public { EndLike(end).cash(ilk, wad); // Exits token amount to the user's wallet as a token GemJoinLike(gemJoin).exit(msg.sender, mul(wad, EndLike(end).fix(ilk)) / RAY); }

```

1. Define calldata for our function

Similarly, as done in step (7), the user needs to define the calldata to interact withcashGEM

```

Copy export cashBATcalldata=(seth calldata 'cashETH(address,address,bytes32,uint)' MCD_JOIN_BAT MCD_END ilkBAT wad)

```

1. Call execute inMYPROXY

Finally, executing thecashBATcalldata in theexecute function of the user'sMYPROXY contract will redeem BAT for DAI, and place this BAT into the user's ETH wallet.

```

Copy seth send MYPROXY 'execute(address,bytes memory)' PROXY_ACTIONS_END cashBATcalldata

```

Vault Owners to Redeem Excess Collateral

Likewise, a vault owner can use thefreeETH orfreeGEM proxy actions function to retrieve any excess collateral they may have locked in the system.

1. Vault Holder State

There are some constraints for vault holders to be aware of. For example, if a user's Vault is under-collateralised then they will not have any excess collateral to claim. Likewise, if the user's Vault is currently in a flip auction at the time of emergency shutdown, it will be necessary for the Vault holder to cancel the auction by callingskip(ilk, id) before callingfree__() .

Similarly, these functions have been completed using Maker proxy contract calls. There may be other scenarios in which 3rd party front ends such as InstaDApp have their own proxies, which will require users to exit from their proxy in order to use the below.

1. Redeeming ETH using thefreeETH function

```

Copy function freeETH( address manager, address ethJoin, address end, uint cdp ) public { uint wad = _free(manager, end, cdp); // Exits WETH amount to proxy address as a token GemJoinLike(ethJoin).exit(address(this), wad); // Converts WETH to ETH GemJoinLike(ethJoin).gem().withdraw(wad); // Sends ETH back to the user's wallet msg.sender.transfer(wad); }

```

2.1. Set calldata

Depending on how many vaults the user has, it will be necessary to repeat this process for each vault ID.

```

Copy export freeETHcalldata=(seth calldata 'freeETH(address,address,address,uint)' CDP_MANAGER MCD_JOIN_ETH MCD_END cdpId )

```

2.2. Execute this calldata

Executing theMYPROXY contract will redeem ETH and place it into the users address.

```

Copy seth send MYPROXY 'execute(address,bytes memory)' PROXY_ACTIONS_END freeETHcalldata

```

1. Redeeming ETH using thefreeGEM function

```

Copy function freeGem( address manager, address gemJoin, address end, uint cdp ) public { uint wad = _free(manager, end, cdp); // Exits token amount to the user's wallet as a token GemJoinLike(gemJoin).exit(msg.sender, wad); }

```

3.1. Set calldata

Depending on how many vaults the user has, it will be necessary to repeat this process for each vault ID.

```

Copy export freeBATcalldata=(seth calldata 'freeETH(address,address,address,uint)' CDP_MANAGER MCD_JOIN_BAT MCD_END cdpId )

```

3.2. Execute this calldata

Executing theMYPROXY contract will redeem BAT (or other collateral types) and place them into the users address.

```

Copy seth send MYPROXY 'execute(address,bytes memory)' PROXY_ACTIONS_END freeBATcalldata

```

Conclusion

The above outlines how to redeem Dai and excess Vault collateral using the command line.

In summary, we showed how to check your Dai holdings, how to approve a proxy to withdraw Dai from your wallet and then to usecashETH/GEM functions to withdraw collateral into the user's ETH wallet using theMYPROXY contract . For Vault owners, we showed how to redeem collateral by using theMYPROXY contract and thefreeGEM function.

In the event of emergency shutdown we envision that it will still be possible to sell Dai on the open market as well as by making use of economically incentivized redemption keepers to meet market needs for both Dai owners and Vaults holders.

Previous Multi Collateral Dai (MCD) CLI Next Emergency Shutdown (ES) CLI Last updated4 years ago On this page

Export as PDF