# Proxy Actions - Detailed Documentation

A generalized wrapper for the Maker Protocol * Contract Name: * DssProxyActions.sol * Type/Category: * Proxy Module * Associated MCD System Diagram * Contract Source * Etherscan * * Proxy Actions * * Proxy Actions End * * Proxy Actions DSR * * *

1. Introduction (Summary)

The Proxy Actions contract is a generalized wrapper for the Maker Protocol. It's basically a set of proxy functions for MCD (using dss-cdp-manager). The contract's purpose is to be used via a personal ds-proxy and can be compared to the original Sag-Proxy as it offers the ability to execute a sequence of actions atomically.

1. Contract Details

Glossary (Proxy Actions)

manager - enables a formalized process for CDPs to be transferred between owners.

ilk - a collateral type.

usr - user address.

cdp - Collateralized Debt Position (now, known asVault ).

dst - refers to the destination address.

wad - quantity of tokens, usually as a fixed point integer with 10^18 decimal places.

rad - a fixed point integer, with 10^45 decimal places.

dink - change in collateral.

dart - change in debt.

ethJoin - allows native Ether to be used with the system.

gemJoin - allows standard ERC20 tokens to be deposited for use with the system.

daiJoin - allows users to withdraw their Dai from the system into a standard ERC20 token.

Key Functionalities (as defined in the smart contract)

DssProxyActions

- open()
- : creates anUrnHandler
- (cdp
- ) for the addressusr
- (for a specificilk
- ) and allows the user to manage it via the internal registry of themanager
- .
- give()
- : transfers the ownership of thecdp
- to theusr
- address in themanager
- registry.
- giveToProxy()
- : transfers the ownership ofcdp
- to the proxy ofusr
- address (viaproxyRegistry
- ) in themanager
- registry.
- cdpAllow()
- : allows/deniesusr
- address to manage thecdp
- .
- urnAllow()
- : allows/deniesusr
- address to manage themsg.sender
- address asdst

- forquit
- .
- flux()
- : moveswad
- amount of collateral fromcdp
- address todst
- address.
- move()
- : movesrad
- amount of DAI fromcdp
- address todst
- address.
- frob()
- : executesfrob
- tocdp
- address assigning the collateral freed and/or DAI drawn to the same address.
- quit()
- : movescdp
- collateral balance and debt todst
- address.
- enter()
- : movessrc
- collateral balance and debt tocdp
- .
- shift()
- : movescdpSrc
- collateral balance and debt tocdpDst
- .
- lockETH()
- : depositsmsg.value
- amount of ETH inethJoin
- adapter and executesfrob
- tocdp
- increasing the locked value.
- safeLockETH()
- : same thanlockETH
- but requiringowner == cdp owner
- .
- lockGem()
- : depositswad
- amount of collateral ingemJoin
- adapter and executesfrob
- tocdp
- increasing the locked value. Gets funds frommsg.sender
- iftransferFrom == true
- .
- safeLockGem()
- : same thanlockGem
- but requiringowner == cdp owner
- .
- freeETH()
- : executesfrob
- tocdp
- decreasing locked collateral and withdrawswad
- amount of ETH fromethJoin
- adapter.
- freeGem()
- : executesfrob
- tocdp
- decreasing locked collateral and withdrawswad
- amount of collateral fromgemJoin
- adapter.
- draw()
- : updates collateral fee rate, executesfrob
- tocdp
- increasing debt and exitswad
- amount of DAI token (minting it) fromdaiJoin
- adapter.

- wipe()
- : joinswad
- amount of DAI token todaiJoin
- adapter (burning it) and executesfrob
- tocdp
- for decreasing debt.
- safeWipe()
- : same aswipe
- but requiringowner == cdp owner
- .
- wipeAll()
- : joins all the necessary amount of DAI token todaiJoin
- adapter (burning it) and executesfrob
- tocdp
- setting the debt to zero.
- safeWipeAll()
- : same aswipeAll
- but requiringowner == cdp owner
- .
- lockETHAndDraw()
- : combineslockETH
- anddraw
- .
- openLockETHAndDraw()
- : combinesopen
- ,lockETH
- anddraw
- .
- lockGemAndDraw()
- : combineslockGem
- anddraw
- .
- openLockGemAndDraw()
- : combinesopen
- ,lockGem
- anddraw
- .
- wipeAndFreeETH()
- : combineswipe
- andfreeETH
- .
- wipeAllAndFreeETH()
- : combineswipeAll
- andfreeETH
- .
- wipeAndFreeGem()
- : combineswipe
- andfreeGem
- .
- wipeAllAndFreeGem()
- : combineswipeAll
- andfreeGem
- .
-

DssProxyActionsFlip

- exitETH()
- : exitswad
- amount of ETH fromethJoin
- adapter. This is received in thecdp
- urn after the liquidation auction is over.
- exitGem()
- : exitswad
- amount of collateral fromgemJoin
- adapter. This is received in thecdp
- urn after the liquidation auction is over.
-

DssProxyActionsEnd

- freeETH()
- : once the system is caged, this recovers the remaining ETH fromcdp
- (pays the remaining debt if exists).
- freeGem()
- : once the system is caged, this recovers the remaining token fromcdp
- (pays remaining debt if exists).
- pack()
- : once the system is caged, this packswad
- amount of DAI to be ready for cashing.
- cashETH()
- : once the system is caged, this casheswad
- amount of previously packed DAI and returns the equivalent in ETH.
- cashGem()
- : once the system is caged, this casheswad
- amount of previously packed DAI and returns the equivalent in gem token.
-

DssProxyActionsDsr

- join()
- : joinswad
- amount of DAI token todaiJoin
- adapter (burning it) and moves the balance topot
- for DAI Saving Rates.
- exit()
- : retrieveswad
- amount of DAI frompot
- and exits DAI token fromdaiJoin
- adapter (minting it).
- exitAll()
- : performs the same actions asexit
- but for all of the available amount.
-

1. Key Mechanisms & Concepts

Thedss-proxy-actions was designed to be used by the Ds-Proxy, which is owned individually by users to interact more easily with the Maker Protocol. Note that it is not intended to be used directly (this will be covered later). Thedss-proxy-actions contract was developed to serve as a library for user's ds proxies. In general, the ds proxy receives two parameters:

- Proxy library address
-
    - In this case, the dss proxy actions library
- *
- Call data
-
    - Functions and parameters you want to execute
- *
-

Reference the ds-proxy for more information here.

DSProxy Summary (as it relates to the dss-proxy-actions contract)

The ds-proxy contact's purpose is to execute transactions and sequences of transactions by proxy. The contract allows code execution using a persistent identity. This can be very useful to execute a sequence of atomic actions. Since the owner of the proxy can be changed, this allows for dynamic ownership models, e.g. a multisig.

- In the later example, we will see how theexecute function
- works by using the proxy to execute calldata _data on contract _code.
- The functions parameters are:
-
    - address _target
-
    - bytes memory _data
- *
- For the address-target you pass in, that will be the library you want to use, in this case the proxy actions library.

- For the Memory data you pass in, that will be the call data of what exactly you want to execute.
-
    - Ex:
-
    - Want to open a Vault; then the bytes memory data you will pass in will be an ABI encoder that executes open function with certain parameters.
- *
- Note:
- This is used for both SCD and MCD.
-

Proxy Action Usage Example (How a proxy call can look like)

```

Copy proxy.execute(dssProxyActions, abi.encodeWithSignature("open(address,bytes32,address)", address(manager), bytes32("ETH-A"), address(0x123)))

```

- Your ds-proxy is only for you, so we create it for a wallet, so each wallet has its own ds proxy that nobody else should be able to execute with that proxy.
- In MCD, the Vaults will not be owned by your wallet but by your ds proxy, which allows you to execute any function via the ds proxy. Such as performing actions within your Vaults and/or group a lot of actions within one transaction.
- The execution looks something like this:
-
    - Proxy execute (call to the Ethereum blockchain) where the first parameter is the contract you are using for the library (in this case, dss proxy actions). Not that this is something the frontend will do for you.Example:
-
    - When you want to open a Vault, it will send the transaction to the proxy to execute the execute function, and will then pass in the dss proxy action address and the second parameter that will be passed is the function itself that your ds proxy needs to execute from the dss proxy actions. In this case, we want to execute the open function from dss proxy action - so your proxy will delegate calling the open function from the dss proxy actions library. We need to do it this way because the second parameter is the bytes call data format parameter, so this function we ABI Encode with signature open. So, we pass the signature and then the actual parameters we want to pass to this function. In this case, the manager, the first param of the open function, the Collateral type, and the address you want to create the Vault for (In this case, the address is 0x123)
-
    - Note:
-
    - UI decides which proxy action the user will use.
- *
-

4.Gotchas

- Using dss-proxy-actions directly can result in the loss of control over your Vault
-
    - If you open a new Vault via the dss proxy actions (centralized) without a ds proxy you would be creating a Vault that is owned by the dss proxy actions that anyone could call publicly. It would be owned by the dss proxy actions contract and anyone could execute actions on your Vault. Therefore, there is significant risk if you directly use the dss proxy actions.
- *
- When interacting with the dss-proxy-actions you need a certain allowance to get Dai or MKR funds from the user's wallet. You need allowance from your wallet to the ds-proxy (not dss-proxy-actions). Because, when you execute the dss-proxy actions, you are actually performing that action in the environment of your ds-proxy, which is delegating calls or importing the function from the proxy actions and not executing them directly.
-

5.Failure Modes

- Ds proxy is a general purpose proxy
-
    - This means that as a user of the ds-proxy, you can execute whatever you want whether that be the Dss-proxy-actions or any other piece of code. Users are therefore responsible for what they are executing and thus, need to have trust in the UI they are using (similar to any other transaction you are executing from your wallet).
-
    - In terms of failure modes, this means you can execute a malicious proxy action as well as a direct action that could potentially send your ETH to a random address. To be extra cautious, you should check your wallets call

data and/or audit what your wallet does as they could potentially present users with some unwanted random call data and execute unwanted actions.

- 
    - Overall, this point is to say that there is always a risk when using a ds proxy.
- *
- 

Previous Proxy Module Next Vote Proxy - Detailed Documentation Last updated4 years ago On this page *

Export as PDF