Liquidation 2.0 Module

The Maker Protocol's Collateral Auction House (Liquidation System 2.0) * Contract Source(s): *clip.sol * ,dog.sol * ,abacus.sol * Type/Category: * DSS —> Liquidation 2.0 Module * Etherscan * *Dog * * Clip (ETH-A) * * Abacus (ETH-A) * * *

1. Introduction

Summary: In the context of the Maker protocol, aliquidation is the automatic transfer of collateral from an insufficiently collateralized Vault, along with the transfer of that Vault's debt to the protocol. In the liquidation contract (theDog), an auction is started promptly to sell the transferred collateral for DAI in an attempt to cancel out the debt now assigned to the protocol.

1. Features

Instant Settlement

Unlike the old Liquidation 1.2 system which utilized English auctions, in which DAI bids are placed, with a participant's capital remaining locked until they are outbid or until the auction terminates, Liquidation 2.0 uses Dutch auctions which settle instantly. They do so according to a price calculated from the initial price and the time elapsed since the auction began. Price versus time curves are discussed more later. The lack of a lock-up period mitigates much of the price volatility risk for auction participants and allows for faster capital recycling.

Flash Lending of Collateral

This feature, enabled by instant settlement, eliminates any capital requirement for bidders (excepting gas)—in the sense that even a participant with zero DAI (and nothing to trade for DAI) could still purchase from an auction by directing the sale of the auction's collateral into other protocols in exchange for DAI. Thus, all DAI liquidity available across DeFi can be used by any participant to purchase collateral, subject only to gas requirements. The exact mechanics are discussed above, but essentially a participant needs to specify a contractwho (which conforms to a particular interface), andcalldata to supply to it, and the auction contract will automatically invoke whatever logic is in the external contract.

Price as a Function of Time

Price-versus-time curves are specified through an interface that treats price at the current time as a function of the initial price of an auction and the time at which that price was set. How to determine the most effective price curve for a given collateral is still an active area of research; some initial options (linear, step-wise exponential, and continuous exponential) have been implemented for research purposes and initial deployment. Other candidates besides these include a piecewise linear curve and a piecewise exponential curve. This module is configurable and can be replaced in the course of innovation.

It is important for bidders to take into account that while the price almost always decreases with time, there are infrequent occasions on which the price of an active auction may increase, and this could potentially result in collateral being purchased at a higher price than intended. The most obvious event that would increase the price in a running auction is if that auction is reset (viaredo), but changes to the configurable parameters of a price decrease calculator (or even the switching out of one calculator for another) by governance can also increase the price. It is recommended that bidders (or bidding UIs that abstract this detail away from the user) choose the maximum acceptable price (max) argument totake carefully to ensure a desirable outcome if the price should unexpectedly increase.

Resetting an Auction

As mentioned above, auctions can reach a defunct state that requires resetting for two reasons:

- too much time has elapsed since the auction started (controlled by thetail
- governance parameter)
- the ratio of the current price to the initial price has fallen below a certain level (specified by thecusp
- governance parameter).
-

The reset function, when called, first ensures that at least one of these conditions holds. Then it adjusts the starting time of the auction to the current time, and sets the starting price in exactly the same way as is done in the initialization function (i.e. the current OSM price increased percentage-wise by thebuf parameter). This process will repeat until all collateral has been sold or the whole debt has been collected (unless the auction is canceled viayank , e.g. during Emergency Shutdown); contrast this behavior with the current auctions, which reset until at least one bid is received.

Improved Keeper Wallet Security

If keepers decide to use theclipperCallee pattern, then they need not store DAI or collateral on that account. This means a keeper need only hold enough ETH to execute transactions that can orchestrate theClipper.take call, sending collateral to a

contract that returns DAI to themsg.sender to pay for the collateral all in one transaction. The contract implementing theclipperCallee interface can send any remaining collateral or DAI beyondowe to a cold wallet address inaccessible to the keeper.NOTE: If the keeper chooses to behave as an EOA address, then the DAI and collateral would be exposed just as in LIQ-1.2 unless special care is taken to create a proxy contract.

1. Contract Details

Parameters Set By Governance (throughfile)

Abacus/LinearDecrease --tau [seconds]

Seconds after auction start when the price reaches zero.

Abacus/StairstepExponentialDecrease --cut [ray]

Per-step multiplicative factor.cut = 0.99 * RAY specifies a 1% drop everystep seconds.

Abacus/StairstepExponentialDecrease --step [seconds]

Length of time between price drops.

Abacus/ExponentialDecrease --cut [ray]

Per-second multiplicative factor.cut = 0.99 * RAY specifies a 1% drop every second.

Clipper --buf [ray]

The multiplicative factor to increase the starting price of an auction. E.g. if the current OSM price of an asset is 1,000 andbuf = 1.2 * RAY (20% above), then the initial price of that auction will be 1,200.

Clipper --calc [address]

The contract address of the price calculator function. Adheres toAbacus interface. Some examples of price functions are found inabaci.sol file.

Clipper --chip [wad]

Percentage oftab tosuck fromvow to incentivize keepers when liquidating a vault or resetting an already existing auction.chip = 0.02 * WAD is 2%.

Clipper --cusp [ray]

Percentage price drop that can occur before an auction must be reset. Together withtail , this parameter determines when an auction needs to be reset. E.g. if the initial price of an auction (top) is set to 1,200 andcusp = 0.6 * RAY (60% of the starting price), then the auction will need to be reset when reaching just below the price of 720.

Clipper --dog [address]

The address of the liquidation module contract.

Clipper --spotter [address]

The Collateral price module contract address.

Clipper --tail [seconds]

Seconds that can elapse before an auction must be reset. Together withcusp , this parameter determines when an auction needs to be reset. E.g. iftail is 1800 seconds, then if an auction is not complete after 30 minutes have elapsed, it will need to be reset.

Clipper --tip [rad]

Flat fee tosuck fromvow to incentivize keepers when liquidating a vault or resetting an already existing auction.tip = 100 * RAD is 100 DAI.

Clipper --vow [address]

The address of the accounting system contract. The recipient of DAI raised in auctions.

Dog --Hole [rad]

Max DAI needed to cover debt + liquidation penalty of active auctions.Hole = 10,000,000 * RAD is 10M DAI.

Dog --ilk.chop [wad]

Liquidation Penalty per collateral (ilk). E.g. if there is a vault ready to be liquidated that has a debt of 1,000 DAI andchop = 1.13 * WAD (13% above), then the max amount to be raised by the auction will be 1,130 DAI.

Dog --ilk.clip [address]

The address of the auction module contract. Oneclip per collateral (ilk).

Dog --ilk.hole [rad]

Max DAI needed to cover debt + liquidation penalty of active auctions per collateral (ilk).hole = 10,000,000 * RAD is 10M DAI.

Dog --vow [address]

The address of the accounting system contract. The recipient of the bad debt coming from a vault when it's liquidated.

Vault Liquidation

https://github.com/makerdao/dss/blob/liq-2.0/src/dog.sol

The Vault liquidation function (Dog.bark) takes three caller supplied arguments:

- ilk
- : the collateral ilk
- urn
- : the Vault to be liquidated
- kpr
- : the address where DAI incentives will be sent
-

Dog.bark performs several actions:

- confiscates the given Vaulturn
- if it's undercollateralized and
-
    - sends the collateral to theilk
-
    - 'sClipper
-
    - increments thevow
-
    - 's bad debt accumulator
- *
- pushes the bad debt into the debt queue
- adds the bad debt plus the liquidation penalty to theHole
- with theDirt
- accumulator
- adds the bad debt plus the liquidation penalty to theilk.hole
- with theilk.dirt
- accumulator
- initiates the auction by callingClipper.kick()
- fires theBark()
- event
-

In the context of the Maker protocol, a "liquidation" is the automatic transfer of collateral from an insufficiently collateralized Vault, along with the transfer of that Vault's debt to the protocol. In both the Liquidation 1.2 version (theCat) and the Liquidation 2.0 version (theDog), an auction is started promptly to sell the transferred collateral for DAI in an attempt to cancel out the debt now assigned to the protocol. This makes the behavior of the new contract very similar to that of the old one, but there are some important differences, explained below.

Partial vs. Total Liquidations

In the current system, in each call to the liquidation function (Cat.bite) transfers a fixed amount of debt (thedunk) from the affected Vault, along with a proportional amount of the Vault's collateral to the protocol. For example, if 50% of a Vault's debt is taken by the protocol, then half of its collateral is taken as well. If the Vault's debt is less than thedunk , then all debt and collateral is taken. In 2.0, all debt is taken when the liquidation function (Dog.bark) is called, and no analogue of thedunk parameter exists. The reasoning behind this change is that because the new auctions allow partial purchases of collateral, the liquidity available to a participant no longer limits their ability to participate in auctions, so instead the total number of auctions should be minimized. Just to emphasize, there is no longer a minimum DAI liquidity requirement for the sale of collateral on a per-participant basis.

Limits on DAI Needed to Cover Debt and Fees of Active Auctions

In situations involving large amounts of collateral at auction, the current and new designs modify the behavior described in the previous section. Both liquidations 1.2 and 2.0 implement a limit on the total amount of DAI needed to cover the summed debt and liquidation penalty associated with all active auctions. InLIQ-1.2 this is called thebox , and inLIQ-2.0 we call this theHole . Whenever the maximum possible addition to this value is less than the amount of debt+fees that would otherwise be sent to auction, a partial liquidation is performed so as not to exceed this amount. In the current system there is only a global limit; in 2.0, in addition to the global limit, there is also a per-collateral limit. Similar to how there is anilk.line for a collateral's debt ceiling and aLine for the overall system debt ceiling, there is now anilk.hole to correspond with theHole . This ensures that typical market depth against DAI can be taken into account on a per-collateral basis by those determining risk parameters. TheDirt accumulator tracks the total DAI needed to cover the debt and penalty fees of all active auctions, and must be less thanHole for a call tobark to succeed. For each collateral type,ilk.dirt tracks the total DAI needed to cover the debt and penalty fees of a all active auctions for thatilk , and must be less thanilk.hole for a call tobark (on a Vault of thatilk) to succeed.

Auction Initiation

https://github.com/makerdao/dss/blob/liq-2.0/src/clip.sol

The auction initiation function (Clipper.kick) takes four caller supplied arguments:

- tab
- : the target DAI to raise from the auction (debt
- +stability fees
- +liquidation penalty
-) [rad]
- lot
- : the amount of collateral available for purchase [wad]
- usr
- : the address to return any excess collateral to
- kpr
- : the address where DAI incentives will be sent
-

Clipper.kick performs several checks and actions:

- checks that the caller is authorized (only theDog
- or governance may callClipper.kick
-)
- checks that liquidations are enabled in the four-stage circuit breaker
- increments a counter and assigns a unique numericalid
- to the new auction
- inserts theid
- into a list tracking active auctions
- creates a structure to record the parameters of the auction; this includes:
  - it's position in the active auctions list
  - the target amount of DAI to raise from bidders (tab
  -)
  - the amount of collateral available for purchase (lot
  -)
  - the Vault that was liquidated to create this auction
  -
    - allows return of collateral if not all of it is sold

-
  -
    - allows the return of collateral andtab
-
  -
    - whenClipper.yank
-
  -
    - is called byEnd.snip

-
  - the timestamp of the auctions creation (as a Unix epoch)
-
  - the initial price of collateral in the auction (top
-
  -)
- *
- sends an incentive denominated in DAI to thekpr
- fires theKick()
- event
-

The initial price is set by reading the current price in the corresponding Oracle Security Module (OSM) and multiplying by a configurable percentage (thebuf parameter). Note that thecurrent OSM price is between one and two hours delayed relative to the actual market price. A keeper doesn't make a call toClipper.kick directly, but rather makes a call toDog.bark which in turn callsClipper.kick .

Liquidation Incentive Mechanism

In this design, there is less incentive to quickly liquidate Vaults than in the current system, because there is no inherent advantage obtained by doing so. In contrast, the current auction system grants the account triggering a liquidation the privilege of making the first bid in the resulting auction. It is unclear whether this matters significantly in practice, or whether some stronger incentive should be added (for example, a small DAI reward paid to liquidators).

To ensure there was a remedy for this potential issue, an incentive mechanism was added for liquidators. The form of the incentive is, on a per-collateral type basis, a constant amount of DAI plus an amount of DAI that scales linearly with the amount of debt associated with the liquidation. Either contribution can be set to zero. Such a structure is justified by the following:

- The reward is set per-collateral to give maximum flexibility to include not just per-collateral risk parameters likemat
- (collateralization ratio) andchop
- (liquidation penalty) in its setting, but also to allow for unique market conditions that might only apply to one or a few collateral types.
- The component of the reward that increases linearly with the total Vault debt is intended to be used to reward liquidators for reducing risk to the system, as risk itself scales with the size of undercollateralized Vaults—a Vault that is twice as big as another represents twice the risk of bad debt accrual. Or viewed another way, liquidating two vaults of size X represents the same risk reduction as liquidating one Vault of size 2X—thus the reward to a liquidator ought to be similar. Further, the system canafford
- to pay more for larger liquidations, because the liquidation penalty is also proportional to the amount of debt outstanding for a given Vault.
- The constant component of the reward can be used to cover gas costs (which are per-Vault for liquidators) or to allow MKR holders to effectively pay Keepers to clear small Vaults that would otherwise not be attractive for liquidation.
-

These parameters must be set extremely carefully, lest it be possible to exploit the system by "farming" liquidation rewards (e.g. creating Vaults with the intention of liquidating them and profiting from the too-high rewards). Generally, the liquidation reward should remain less than the minimum liquidation penalty by some margin of safety so that the system is unlikely to accrue a deficit. This doesn't necessarily prevent farming, it just helps ensure the system remains solvent. For example, incentives can be farmed in a capital-efficient way whenDirt is close toHole or whenilk.dirt is close toilk.hole for some collateral type. An attacker would purchase a small amount of collateral from a running auction, freeing up a small amount of room relative to eitherHole orilk.hole , then liquidate a small portion of an existing Vault to collect the reward, and repeat the process. This can be done over and over in the same transaction, and the activation of EIP 2929 (scheduled for the Berlin hard fork at the time of writing) will significantly reduce the associated gas costs (due to warm storage reads and writes). Note that since the attacker does not need to create their own Vaults, the size of the liquidation penalty in relation to the incentive value is not a deterrent; only gas costs matter to the attacker. The fact that the Dog prevents partial liquidations that remove less thanilk.dust debt from a Vault helps to mitigate this scenario--so long as the liquidation penalty exceeds the total reward for this minimal liquidation size,and the liquidation penalty is reliably collected by the resulting auction (which may not hold under conditions of market or network perturbation), the system should not on balance accrue bad debt.

Four-Stage Liquidation Circuit Breaker

In this section, we'll cover the four stages of the liquidation circuit breaker. In contrast toLIQ-1.2 , Liquidations 2.0 comes with a four-stage circuit breaker built into theClipper contract. The stages are:

- liquidations enabled
- (0
-): This means the breaker is not tripped and the protocol can liquidate new Vaults as well as service old liquidations.
- new liquidations disabled
- (1
-): This means no new liquidations (Clipper.kick
-).
- new liquidations and resets disabled
- (2
-): This means no new liquidations (Clipper.kick
-), and auctions that have reached either a price or time endpoint cannot be reset (Clipper.redo
-).
- liquidations disabled
- (3
-): This means no new liquidations (Clipper.kick
-), no takes (Clipper.take
-), and no resets (Clipper.redo
-).
-

Just like inLIQ-1.2 , the circuit breaker will be available through aClipperMom contract giving governance the ability to bypass theGSM delay for circuit breaker actions.

Bidding (Purchasing)

https://github.com/makerdao/dss/blob/liq-2.0/src/clip.sol

The purchasing function (Clipper.take) takes five caller supplied arguments:

- id
- : the numerical id of the auction to bid on
- amt
- : the maximum amount of collateral to buy (amt
-) — a purchase behaves like a limit order [wad]
- max
- : the maximum acceptable price in DAI per unit collateral (max
-) [ray]
- who
- : address that will receive the collateral (who
-)
- data
- : an arbitrary bytestring (if provided, the addresswho
- , if it is neither theDog
- norVat
- address stored by the Clipper, is called as a contract via an interface described below, to which this data is passed) [bytes]
-

Clipper.take performs several initial checks:

- a reentrancy check to ensure the function is not being recursively invoked
- that the four-stage circuit breaker is not tripped
- that the auction id corresponds to a valid auction
- that the auction does not need to be reset, either due to having experienced too large a percentage decrease in price, or having existed for too long of a time duration
- that the caller's specified maximum price is at least the current auction price
-

Then, the amount of collateral to attempt to purchase is computed as the minimum of the collateral left in the auction (lot) and the caller's specified quantity (amt)—the resulting value is theslice . This value is then multiplied by the current price of the auction to compute the DAI owed in exchange (owe). Ifowe exceeds the DAI collection target of the auction (tab), thenowe is adjusted to be equal totab , andslice is set totab / price (i.e. the auction will not sell more collateral than is needed to cover debt+fees from the liquidated Vault).

To make it less likely that auctions are left in a state that is unattractive to bid on, further logic is applied if there will be both

left over DAI target and collateral based on the initial determinations forslice andowe . If the remainingtab in such a case would be less thanchost (a value stored by the contract, asynchronously set toilk.dust * ilk.chop / WAD by theupchost() function), then: 1) If the overall DAI target is less thanchost ,the function reverts. Callers should be aware of this possibility and account for it when necessary. 2) Otherwise,owe is set totab - chost , andslice is set to(tab - chost) / price .

This heuristic for preventing a too-small remaining collateral amount and/or DAI target is imperfect and may fail in some situations; if this occurs, there is no serious harm to the protocol; the auction will simply remain uncompleted. However, governance may wish to fully clear such "lost" auctions viatake oryank to avoid cluttering the list of active auctions eachClipper maintains, which could impair the performance of DEX and aggregator integrations (and possibly off-chain bots too, depending on how they are written).

Next, collateral is transferred (within the protocol'saccounting module) to thewho address provided by the caller. If the caller provided a bytestring with greater than zero length, an external call is made to thewho address, assuming it exposes a function, follow Solidity conventions, with the following signature:

```

Copy clipperCall( address,// recipient of DAI uint256,// owe [rad] uint256,// slice [wad] bytes calldata )

```

The first argument is the recipient of DAI. That is, this is the address theclipperCallee must return DAI to. The second argument is DAI owed (as a 45 decimal digit fixed-precision integer, orrad), the third argument is the collateral being purchased (as an 18 decimal digit fixed-precision integer, orwad), regardless of the precision of the external token contract, and the last argument is identical to what bytestring the caller originally supplied to the purchase function. As mentioned earlier, a locking mechanism prevents reentry into the purchase function during this external call, and theClipper.redo call, for security reasons. Example implementations of theClipperCallee interface can be found in theexchange-callees repository.

After the external call completes (or immediately following the transfer of collateral, if no external call was executed), DAI is transferred (internally, within the coreaccounting module) frommsg.sender to the protocol.

Lastly, various values are updated to record the changed state of the auction: the DAI needed to cover debt and fees for outstanding auctions, and outstanding auctions of the given collateral type, are reduced (via a callback to the liquidator contract) is reduced byowe , and thetab (DAI collection target) andlot (collateral for sale) of the auction are adjusted as well. If all collateral has been drained from an auction, all its data is cleared and it is removed from the active auctions list. If collateral remains, but the DAI collection target has been reached, the same is done and excess collateral is returned to the liquidated Vault.

Example Liquidation

?

Figure 1 : NOTE: in the above figure,tau istail .

In this example we can see a linear decrease function (calc), with anETH-A OSM price of200 DAI , abuf of20% , atail (tau) of21600 seconds, atab of*60,000 DAI with alot of347.32 ETH. There are two bidders,Alice andBob .Alice callstake first and is willing to give*50,000 DAI in return for256.41 ETH collateral, a price of195 DAI per ETH. The price of ETH continues to fall over time, andBob picks up the remaining90.91 ETH for10,000 DAI , a price of110 DAI per ETH. If more thantail seconds have elapsed since the start of the auction, or if the price has fallen to less thancusp percent oftop , thenClipper.take would revert if called, and the auction would need to be reset with aClipper.redo call.

Incentive to callredo()

https://github.com/makerdao/dss/blob/liq-2.0/src/clip.sol

The auction initiation function (Clipper.redo) takes two caller supplied arguments:

- id
- : the current auction id
- kpr
- : the address where DAI incentives will be sent
-

Clipper.redo performs several checks and actions:

- a reentrancy check to ensure the function is not being recursively invoked
- that the four-stage circuit breaker is not tripped
- that the auction id corresponds to a valid auction
- that the auction needs to be reset, either due to having experienced too large a percentage decrease in price, or

having existed for too long of a time duration
- updates several fields of the existing auction
-
    - tic
-
    - to reset the auction start time
-
    - top
-
    - with the current OSM price andbuf
-
    - percent
- *
- vat.suck
- s DAI to thekpr
- as an incentive if eligible
- fires theRedo()
- event
-

An auction that has expired or which is currently offered at a value higher than the oracle threshold will likely not complete at favorable values. The system therefore provides a direct incentive toClipper.redo the auction, resetting it's expiration and setting the starting price to match the current oracle price + buf. The redo includes the same Dai incentive to the keeper as theClipper.kick , which is based on the flat fee plus the governance-defined percentage of collateral. There is one exception to this incentive. If thetab orlot * price remaining is under theClipper.chost limit, then there will be no incentive paid toredo the auction. This is to help prevent incentive farming attacks where no keepers bid on dusty lots, andClipper.redo is called repeatedly. If auctions in this state build up, governance may choose to pay a keeper to clean them up.

Emergency Shutdown

https://github.com/makerdao/dss/blob/liq-2.0/src/end.sol

A started auction can be reverted via theauth function calledyank inClipper contract. This function requires that the auction exists and executes the following actions:

- callsdog.digs
- in order to decrease itsDirt
- andilk.dirt
- values by the remaining auctiontab
- sends the remaining collateral to itsmsg.sender
- removes the auction struct data
- fires theYank()
- event
-

This function might be thought of as a general purpose operation to migrate existing auctions to a new contract; however, the only use-case now is in theEnd module, which takes care of system shutdown.

TheEnd module will be upgraded together with the auction contracts as a new functionEnd.snip is required.

This functionEnd.snip is responsible for callingClipper.yank for any running auction when shutdown is triggered. It will receive the collateral fromClipper.yank and will send it back to the vault owner together with the remaining debt to recover. One consideration to note is that the debt that the vault owner will receive includes the liquidation penalty part already charged.

Interfaces

Dog Interface

```

Copy function wards(address) external view returns (uint256); function rely(address) external; function deny(address) external;

```

Standard MakerDAO authorization structure.

```

Copy function vat() external view returns (address); function vow() external view returns (address);
```

```

DSS core address introspection.

```

Copy function ilks(bytes32 ilk) external view returns (address clip, uint256 chop, // wad uint256 hole, // rad uint256 dirt); // rad

```

Returns values configured for a given ilk (ex.ETH-A ).

```

Copy function live() external view returns (uint256);

```

Returns1 if the system is active.

```

Copy function Hole() external view returns (uint256); function Dirt() external view returns (uint256);

```

Getters for the global Hole and Dirt configuration. Both return a rad-precision value.

```

Copy function file(bytes32 what, address data) external; function file(bytes32 what, uint256 data) external; function file(bytes32 ilk, bytes32 what, address data) external; function file(bytes32 ilk, bytes32 what, uint256 data) external;

```

(Authenticated) Parameter modification functions, available to governance. The precision for a numeric argument should match the parameter being set.

```

Copy function chop(bytes32 ilk) external view returns (uint256);

```

Getter for thechop value of a givenilk .chop has wad precision.

```

Copy function bark(bytes32 ilk, address urn, address kpr) external returns (uint256 id);

```

The main liquidation function. Initiates an auction. A keeper calls this function to begin the auction ofurn for a particularilk .kpr is the address where the keeper incentive is sent, allowing keepers to have liquidation rewards sent to a different address than the caller.

```

Copy function digs(bytes32 ilk, uint256 rad) external;

```

(Authenticated) Removes collateral from the accumulator. Called by theClipper . Therad argument has rad precision.

```

Copy function cage() external;

```

(Authenticated) Deactivates theDog and setslive to 0.
```

Clipper Interface

```

Copy function wards(address) external view returns (uint256); function rely(address) external; function deny(address) external;

```

Standard MakerDAO authorization structure.

```

Copy function ilk() external view returns (bytes32);

```

The ilk that this Clipper is associated with.

```

Copy function dog() external view returns (address); function vow() external view returns (address); function spotter() external view returns (address);

```

DSS core address introspection.

```

Copy function calc() external view returns (address);

```

The address of the pricing function used by this Clipper.

```

Copy function buf() external view returns (uint256); // ray function tail() external view returns (uint256); // seconds function cusp() external view returns (uint256); // ray function chip() external view returns (uint256); // wad function tip() external view returns (uint256); // rad

```

Getters for governance-configured auction parameters.

```

Copy function chost() external view returns (uint256); // rad

```

Getter for the stored product ofVat.ilk.dust andDog.ilk.chop . This value is used as a heuristic for whether a partial purchase will leave an auction with too little collateral and whether incentives should be given out whenredo is called.

```

Copy function kicks() external view returns (uint256);

```

Auction counter. Increments each time an auction is initiated.

```

Copy function active(uint256 pos) external view returns (uint256);

```

Returns the id of the auction at indexpos in the list of active auctions.

```

Copy function sales(uint256) external view returns ( uint256 pos, uint256 tab, // rad uint256 lot, // wad address usr, uint96 tic,
```

// Unix epoch uint256 top); // ray

```

Returns information on a particular auction. Completed auctions are removed from the mapping.

```

Copy function stopped() external view returns (uint256);

```

Returns the current circuit breaker status.0 for all functions allowed,1 ifkick cannot be called,2 ifkick andredo cannot be called, and3 ifkick ,redo andtake cannot be called.

```

Copy function file(bytes32 what, address data) external; function file(bytes32 what, uint256 data) external;

```

(Authenticated) Parameter modification functions, available to governance. Numeric arguments should match the precision of the parameter being set.

```

Copy function kick(uint256 tab, uint256 lot, address usr, address kpr) external returns (uint256 id);

```

(Authenticated) Initiates the auction. Called by theDog .tab is rad-precion,lot is wad-precision.

```

Copy function redo(uint256 id, address kpr) external;

```

Called to reset an auction due to expiry or price deviation.

```

Copy function take(uint256 id, uint256 amt, // wad uint256 max, // ray address who, bytes calldata data) external;

```

Called to purchase collateral.

```

Copy function list() external view returns (uint256[] memory); function count() external view returns (uint256);

```

Helpers for iterating the list of active auctions. Uselist() to get the unsorted array of auctions. Get the number of active auctions withcount() . The functionactive(uint256 pos) (see above) can be used to access individual entries without needing to calllist() .

```

Copy function getStatus(uint256 id) external view returns (bool needsRedo, uint256 price);

```

Returns a bool if the auction is eligible for redo and the current price.

```

Copy function upchost() external;

```

Updates thechost value stored in the contract to equal the product ofVat.ilk.dust andDog.ilk.chop (the precision of the final value is rad). This allowstake andredo to obtainchost with a singleSLOAD (readingdust from theVat requires 5SLOAD
```

operations due to how the API is structured, and readingchop is an additionalSLOAD).

```

Copy function yank(uint256 id) external;

```

(Authenticated) Allows an auction to be removed during Emergency Shutdown or via a goveranance action.

1. Known Risks

This section covers some of the known risks with Liquidations 2.0

Incentive Farming

Periodically, governance may increase theilk.dust amount. When this happens, it's usually because gas has become so expensive it impacts the efficiency of liquidations. That is, the cost of callingDog.bark() ,Clipper.take() , orClipper.redo() may exceed the collateral offered. Incentives may be used as a remedy to this potential issue and possibly a way for the protocol to keepilk.dust lower; however, governance must take care when increasing theilk sdust ,tip , orchip not to incentivize the creation of many Vaults to farm this incentive. An example of an exploit is as follows:

1. governance decides to incrementdust
2. by1500 DAI
3. at the same time they scaleilk.tip
4. to subsidize auctions.
5. an attacker realizes it would be profitable between gas and thechop
6. to shard (fork
7.) their existing Vault into many Vaults or create many new Vaults.
8. the spell is voted on and passed
9. using one transaction the attacker puts their Vaults at the edge of unsafe,poke()
10. s the OSM when next price is going down, then callsDog.bark()
11. on all their Vaults to collect the incentive.
12. Using the gains, they can also slightly overbid for their Vaults auctions.
13.

In order to thwart this attack governance must be careful when settingilk.tip andilk.chip so as not to create this perverse incentive.

Price Decreases Too Quickly

If the price decreases too quickly it can have the following consequences:

- the auction ends without any bid, then it needs to be reset and possibly this will keep happening
- bidders end up having reverts due the auction ended before tx confirmation
- bidders end up paying much less than what they were willing to pay (possibly generating permanent bad debt)
-

Price Decreases Too Slowly

If the price decreases too slowly it can have the following consequences:

- Auction price never catches up with the market price, eventually being reset
-
    - After the reset the price catches up, but is less than an optimal market price
-
    - After the reset the price catches up, but still leaves bad debt
-
    - After the reset the auction price still might not catch up, causing more resets and very likely leaving bad debt
- *
-

Front-Running

InLIQ-1.2 there is limited front-running risk as it requires capital to participate in auctions; however, in liquidations 2.0 if a keeper chooses to participate with no capital, there is substantial front-running risk from generalized front-running bots. The easier it is to replace thefrom address of the transaction with one's own, the greater the risk. To mitigate this risk keepers are encouraged to used authorized proxy contracts to interact with liquidations 2.0 and provide some amount of their own capital when bidding. More aggressive gas prices may also work. Unfortunately, we found no great way to prevent generalized front-running that preserves single-block composability.

OSM Risk for Start Price

BecauseClipper.kick andClipper.redo consult the OSM for the collateral price, we are vulnerable to an oracle attack that can only be mitigated by the oracle delay,Dog.Hole , andilk.hole . We must rely on the number of guards in place to prevent price manipulation and oracle attacks. The fact that the price is delayed by one hour, however, prseents a risk of its own: since the price is out-of-date relative to the market, it may be either too high or too low to allow for efficient settlement given other parameters like 'buf' and the price decrease function. The consequences of either case are effectively covered by the sections on the risk of the price decreasing either too quickly or too slowly.

Setting Hole or ilk.hole Too High

WhileDog.Hole andilk.hole can be set much higher in liquidations 2.0, there are still risks to setting this too high. A value forDog.Hole that's set too high could result in far too much DAI demand, breaking the peg high. This is somewhat mitigated by the PSM and stablecoin collateral types, but should still factor in to how this parameter is set. Anilk.hole that is set too high, may have the additional result of causing a downward spiral as the liquidations push the asset price lower. In addition, if there is an oracle attack, this parameter can be thought of as our maximum exposure.

Setting Hole or ilk.hole Too Low

If we set eitherDog.Hole orilk.hole too low, we run the risk of not being able to liquidate enough collateral at once. This could lead to a buildup of undercollateralized positions in the system, eventually causing the accrual of bad debt.

Auction Parameter Changes Affect Running Auctions

Parameters, e.g.tail ,cusp , or the price decrease function or any of its parameters, can be changed at any time by governance and will affect the behavior of running auctions. Integrators should take this possibility into account, reasoning through how sudden changes in parameters would impact their bidding strategies. Governance should endeavor to change parameters infrequently and if possible, only when there are not any auctions that will be affected.

1. Invariants

These invariants have not yet been formally proven, and thus should only be considered intended behavior for now.

Dog Invariants

```

Copy sum_over_all_ilks(Dog.ilk.dirt) == Dog.Dirt (c26.1)

```

Clipper Invariants

```

Copy sum_over_auction_ids(Clipper.sales[id].lot) <= Vat.gem(Clipper) (c27.1)

```

The above is an inequality (and not anequality) because it is impossible to prevent a user or contract from choosing to send collateral to aClipper , even though it only makes sense for theDog to do so. Governance (or a new module) could create a violation of this property by callingkick without transferring appropriate collateral to theClipper . Thus this is actually a pseudo-invariant that only holds if callers ofkick respect the requirement to send collateral to theClipper , but it should hold for the live system if deployment and authorization is done correctly. It should hold if only active auctions are summed over as well.

Invariants involving quantities from both theDog andClipper

```

Copy sum_over_auctions_ids_and_ilks(Dog.ilk.clip.sales[id].tab) == Dog.Dirt (c28.1)

```

The above should hold if only active auctions are summed over as well.

```

Copy sum_over_auctions_ids(Dog.ilk.clip.sales[id].tab) == Dog.ilk.dirt (c28.2)

```

The above should hold if only active auctions are summed over as well.

Note that (c28.1) and (c28.2) together imply (c26.1).

Previous Join - Detailed Documentation Next System Stabilizer Module Last updated2 years ago On this page *

Export as PDF