Collateral Onboarding Spell Crafting Guide

Spells are essentially contracts that do something (or some things) once. They are generally used to change parameters inside the Maker Protocol like adding new collateral types, changing existing vault variables (such as debt ceilings, liquidation amounts, and stability fees), and also managing MKR and DAI distributions.

The engineering process is quite straightforward, but at the same time is specific to each case and also requires comprehensive testing.

Currently, the main spell repositories are located atspells-goerli andspells-mainnet . Working development branch follows aexec-YY-MM-DD branch naming according to the sprint end date, so the Goerli repo should be forked, thoroughly tested, and then handed over to the Protocol Engineering Core Unit for Mainnet execution.

Past implementations of spells can be found in the archive/ folder of each repository alongside test files.

Useful links

- dss-exec-lib:https://github.com/makerdao/dss-exec-lib/
- ilk-registry:https://github.com/makerdao/ilk-registry
- dss-gem-joins:https://github.com/makerdao/dss-gem-joins
- Maker Glossary:https://docs.makerdao.com/other-documentation/system-glossary
- Maker Change Log for contract addresses:https://changelog.makerdao.com/
- Bytes32 converter:https://web3-type-converter.onbrn.com/
- Auditing Executive Spells MD by Derek (PE):https://hackmd.io/ELaxHNjBRn-cVMF6JcML4w?view
-

Intro

Spells are essentiallyDssSpellAction/SpellAction contracts extended from theDssAction abstract contract fromdss-exec-lib . AlongsideDssExecLib , which contains the high-level functions used to interact with the Maker Protocol, andDssExec , the actual spell deployer, this lib bootstraps a good part of the on-chain interactions required to achieve whatever collateral engineering demands coming from the community/CUs.

Taken fromdss-exec-lib itself:

The developer must override the actions() function and place all spell actions within. This is called by the execute() function in the pause, which is subject to an optional limiter for office hours.

Overall, an executive contract can only be executed once it has more votes than the previously-executed executive proposal. After a certain proposal has been voted on, a governance delay (at the time of writing this is 48 hours) is in place before any spell is cast. This grace period is a security consideration to prevent governance attacks.

Polling is used for off-chain voting on deciding how to aggregate multiple proposals, as there's only one executive vote up for voting at a time. This also optimizes the timeline for collateral engineering versus the risk factor of changing too many parameters of the protocol at once.

Different assets can have multiple Maker vaults/collateral types (ilks) to deposit collateral with varying risk factors. That's why you'll observe cases like ETH-A/B/C.

DssSpellAction

Thedss-exec-lib README is pretty comprehensive and goes over every available function exposed, so refer to it whenever necessary to get more granular into specific requirements. The scope of this doc is to provide a more collateral-engineering-focused view of the spell crafting process and embed any notes that might be helpful.

Constants

Due to the fact that the delegate proxy actually casts the spells in practice, all variables need to be defined as constants in the contract. Spells should not rely on or interact with the contract storage for this very reason.

Rates

- uint256 ZERO_PCT_RATE
- = Null multiplier (1).
- uint256 {ADJUSTED}_PCT_RATE
- = Per-second-adjusted stability fee for 1.5% cases. If multiplied by the number of seconds in a year (6060
- 24*365), should return theuint256
- multiplier for 1.5%. Naming convention follows 1.75% =ONE_SEVEN_FIVE_PCT_RATE
- . Common rates can be found inrates.sol

- .
-

If the rate is not already declared in the file above, it can be calculated usingbc :

```

Copy bc-l<<<'scale=27; e( l(1.08) / (60 * 60 * 24 *365) )'# 8%

```

Which will result1.000000002440418608258400030 .

Optionally therates.sol file can be updated to include it:

```

Copy // The key is the pct in basis points 8% = 800 b.p. rates[800]=1000000002440418608258400030;// Notice we rejamove de dot!

```

Precision-related variables for mathematical operations

uint256 MILLION = 10 to the power of 6.

uint256 WAD = 10 to the power of 18.

uint256 RAY = 10 to the power of 27.

uint256 RAD = 10 to the power of 45 (RAY * WAD).

Contract addresses

These contracts below need to be deployed before the creation of the Spell on a per-case basis. Deployed factories to create them can be found on theMaker Changelog after navigating to → chosen network → latest release → Contract addresses →VAR_KEY

Contracts owner needs to be theMCD_PAUSE_PROXY address proxy, which ends up being authorized as the highest-level admin to cast the spells.

- address MCD_JOIN_{TOKEN-L}
- = Contract where users will deposit their assets so the Maker Protocol can internally abstract away the actual tokens used as unlocked collateral (gem). This means that users are actually credited an internal balance once this Join adapter receives deposited tokens so the system is more standardized.
-

VAR_KEY = JOIN_FAB

You can refer to thedss-gem-joins repo for alternate gem join cases according to different token paradigms/functionalities (such as upgradeable contract cases and decimal places) for quick redeploys of similar assets. Be wary of the initial comment in each join for more detailed explanations.

- address MCD_CLIP_{TOKEN-L}
- = Collateral action house instance for this particular ilk. Can be deployed using a more constant factory (without many possibilities like join gems).
-

VAR_KEY = CLIP_FAB

Factory Parameters:

owner =MCD_PAUSE_PROXY.

vat = MCD_VAT, Maker's main lending engine.

spotter = MCD_SPOT, Maker's oracle price feed interface.

dog = MCD_DOG, Maker's liquidation contract.

ilk = Bytes32 representation of collateral case.

- address MCD_CLIP_CURVE_{TOKEN-L}
- = Collateral auctions follows a dutch auction format with a price curve set by this specific contract, feeding the clip contract itself. There are three different curve options - Exponential, Linear & Stairstep - one of which will be outlined in the governance proposal.
-

VAR_KEY = CALC_FAB

- address {GEM}
- = Token address of the underlying asset to be onboarded as collateral.
- address PIP_{GEM}
- = Oracle price feed address. Both gems most up-to-date parameters can be fetched from change log by leveraging theDssExecLib.getChangelogAddress("GEM//PIP_GEM")
- helper function.
-

Functions

officeHours() = Public function the contract overrides from DssAction that returns a bool related to casting spells in a specific weekday timeframe.

action() = Executive function that changes the maker protocol. Should finish by first calling DssExecLib.setChangelogAddress("VAR_ADD ",VAR_ADD) for each new contract deployed then calling the DssExecLib.setChangelogVersion("1.X.YY") function to increase versioning.

Can include multiple actions of different types (e.g. multiple collateral onboarding, multiple stability fees, or debt ceiling changes).

Common Actions

- setIlkStabilityFee
- = Public function used to increase or decrease vault stability fee.
- Parameters:
- bytes32 _ilk
- uint256 _rat
- bool _doDrip
- = relates to the accumulation stability fees for the collateral. Usually is true.
- setIlkAutoLineDebtCeiling
- = Public function used to increase or decrease vault debt ceiling.
- Parameters:
- bytes32 _ilk
- uint256 _amount
- setIlkAutoLineParameters
- = More general version ofsetIlkAutoLineDebtCeiling
- that also includes the gradual gap increase parameters to gradually manage debt ceiling risk.
- Parameters:
- bytes32 _ilk
- uint256 _amount
- uint256 _gap
- = can be fetched by inputting the bytes32 of the ilk (that can be obtained using a tool such as Web3 Type Converter
-) in the ilks method of the DssAutoLine contract
- .
- uint256 _ttl
- = amount in seconds of each gap interval
- setIlkMinVaultAmount
- = Public function to increase or decrease the dust parameter. Parameters:
- bytes32 _ilk
- uint256 _amount
-

Collateral Onboarding

Adding new collateral types to the Maker Protocol is first of all guided by an on-chain governance proposal such as this one . There, we can find many of the necessary parameters for the spell engineering process, leveraging some DssExecLib helpers to kickstart the process. The main one is:

- addNewCollateral
- = Public function to add a new collateral type to the Maker Protocol. Takes in a struct param in the form of ColleteralOpts.
-

CollateralOpts

bytes32 ilk = Bytes32 result of the hyphenated, all caps token vault name (e.g. WBTC-C). Can be passed down directly as a string.

address gem = Address of the gem of the particular underlying token ({GEM} variable), found already deployed in the Maker Changelog with {TOKEN_TICKER } (e.g.WBTC) as its key.

address join = Address of join gem (MCD_JOIN_{TOKEN-L}).

address clip = Address of auction clip (MCD_CLIP_{TOKEN-L}).

address calc = Address of auction price discovery curve (MCD_CLIP_CALC_{TOKEN-L}). Type of contract to be deployed is decided by theAuction Price Function (calc)gov proposal parameter.

address pip = Address of the price feed oracle to be used(PIP_{GEM}).

bool isLiquidatable = Usually true, the collateral can indeed be liquidated.

bool isOSM = True if there's an OSM contract being used for the price feed.

bool whitelistOSM = True if there's the need to whitelist an OSM contract being onboarded to the Maker Protocol.

uint256 ilkDebtCeiling =Debt Ceiling (line) gov proposal parameter. Set in DAI. Debt ceiling is the maximum amount of DAI able to be minted from a given vault collectively.

uint256 minVaultAmount =Debt Floor (dust) gov proposal parameter. Set in DAI. Dust parameter is the minimum amount of DAI possible to generate in a given vault.

uint256 maxLiquidationAmount =Local Liquidation Limit (ilk.hole) gov proposal parameter. Set in DAI.

uint256 liquidationPenalty = Basis point percentage of penalty fee (e.g. 13% = 1300). Usually 1300 for most cases.

uint256 ilkStabilityFee = Per-second-rate obtained from theStability Fee gov proposal parameter ({ADJUSTED }_PCT_RATE).

uint256 startingPriceFactor =Auction Price Multiplier (buf)gov proposal parameter. Set in basis points (1.2 = 12000). Starting price multiplier is how much higher than the oracle price auction starts at.

uint256 breakerTolerance =Breaker Price Tolerance (tolerance) gov proposal parameter. Set in basis points (0.5 = 5000). How large of a price drop is tolerated before liquidations are paused.

uint256 auctionDuration =Maximum Auction Duration (tail)gov proposal parameter. Given in minutes (Solidity time unit).

uint256 permittedDrop =Maximum Auction Drawdown (cusp) gov proposal parameter. Set in basis points (0.4 = 4000). Maximum percentage drop in collateral price during a collateral auction before the auction is reset.

uint256 liquidationRatio =Liquidation Ratio gov proposal parameter. Set in basis points (175% = 17500). Maximum amount of DAI debt that a vault user can draw from their vault given the value of their collateral locked in that vault.

uint256 kprFlatReward =Flat Kick Incentive (tip) gov proposal parameter. Set in DAI. Flat DAI reward a keeper receives for triggering liquidations (to compensate for gas costs).

uint256 kprPctReward =***Proportional Kick Incentive (chip) *** gov proposal parameter. Set in basis points (0.1% = 10). Percentual reward keeper receives from liquidations.

To finish onboarding a new collateral, a couple of additional configs are needed:

- DssExecLib.setStarstepExponentialDecrease(CLIP_CALC,Price Change Interval (step)
- in seconds,Price Change Multiplier (cut)
- in basis points).
- DssExecLib.setIlkAutoLineParameter(ILK_STR, ilkDebtCeiling**,Target Available Debt (gap)
- ,
- Ceiling Increase Cooldown (ttl)
- ** in Solidity time units).
-

Previous Collateral Onboarding Guide for Smart Contracts Domain Teams Next Collateral Risk Assessment Guide Last updated1 year ago On this page * Useful links * Intro * DssSpellAction * Constants * Functions * CollateralOpts

Was this helpful?