Collateral Onboarding Guide for Smart Contracts Domain Teams

Directory

    - Update DSS Spell
31.
    -
        - License
32.
    - *
33.
    - Spell Testing
34.
    -
        - Make a test for collateral addition
35.
    -
        - CI process
36.
    -
        - Deployment and execution
37.
    - *
38. *
39. Deploy Collateral Type on Mainnet
40.
    - Emergency spells
41. *
42.

1. Introduction

This Collateral Onboarding Smart Contract Guide succinctly captures the best practices of past and present Community smart contract developers and the Maker Foundation's smart contracts team. For developers interested in working on the Smart Contracts Domain Team or independently contributing to MakerDAO, the guide describes how gems are joined and spells are cast, among other things. This guide is intended to evolve as the Maker Community improves on the processes described herein.

The Smart Contract Domain Team oversees the assessment of the collateral's token contract, deployment of its adapter and auction, as well as development of a Spell or Spell snippet, which is responsible for activating the collateral type in the Maker Protocol.

At a high level, the smart contract contributor(s) of a Collateral Onboarding team do the following:

1. Conduct a collateral token smart contract assessment
2. Add the collateral type to Maker Protocol codebase
3. Deploy the collateral type on Kovan
4. Deploy the collateral type on Mainnet
5.

Smart contract domain contributor(s) must have a strong understanding of the smart contracts within the Maker Protocol. Therefore, it is recommended to thoroughly review theMaker Protocol 101 documentation and each smart contract module detailed in theIntroduction to the Maker Protocol .

Also review theMulti-Collateral Dai README.md documentation on GitHub. The conceptual overviews of Collateral, Adapters, Price Feeds, and Liquidations contained therein will be helpful to Collateral Onboarding.

1. Dapptools

Thedapptools framework, listed below, offers a number of tools that will be used to conduct the work described in this guide. Therefore, it must be installed and set-up properly.

Dapptools Framework

- dapp
-
    - ethereum smart contract development framework to assist with building and testing smart contracts
- seth
-
    - command-line interface to interact with ethereum networks
- hevm
-
    - Haskell EVM implementation optimized to run solidity tests
-

Please followthese instructions to install dapptools locally on your machine. TheUsing Kovan section will assist in setting up the environment to run bothseth anddapp .

1. Collateral Token Smart Contract Technical Assessment

Conducting a token contract technical assessment is an important step in evaluating the viability of a new collateral type. The assessment is conducted to verify the token contract's intended behavior and catch vulnerabilities. It also creates a template for easy

comparison between other collateral types and gives insight into the requirements for theGemJoin Adapter.

To conduct a technical assessment, please follow the rubric and short guide published in this forum post . Finish the entire evaluation, but leave the following question blank:Can use existing MCD collateral type adapter? You'll answer it later (see the next section).

Here are some great examples of token contract technical assessments:

- [PAXUSD] ERC20 Token Smart Contract Technical Assessment
- (stablecoin)
- [LEND] ERC20 Token Smart Contract Domain Team Assessment
- (standard ERX20)
- [USDT] ERC20 Token Smart Contract Technical Assessment
- (stablecoin)
- [COMP] ERC20 Token Smart Contract Domain Community Assessment
- (yield farming / governance ERC20)
-

The best way to highlight important details and flag potential vulnerabilities is to describe the implementation and risk in theCollateral Logic Summary andCollateral Risk Summary , respectively. You are encouraged to use a smart contract testing tool to confirm your discussion points, but doing so does not substitute for thoughtful and comprehensive analysis. Place the results of the smart contract tool underSupporting Tools.

Recommended tools:

- https://github.com/ConsenSys/surya
- https://github.com/crytic/slither
- https://github.com/brianmcmichael/slither-scripts
-

1. Add collateral type to Maker Protocol codebase

Adding a new collateral type (ilk) to the Maker Protocol requires the addition of the following contracts, if they don't already exist.

Contracts associated with Collateral Types

GemJoin

TheGemJoin adapter is the interface between the collateral token contract and the core accounting contract (Vat). Its purpose is to abstract away all the particular features of a specific token, hence standardizing the token behavior before being debited in the Maker Protocol. It's similar toWETH , but rather than conforming ETH to an ERC20 standard, aGemJoin adapter conforms XYZ token to Maker's internal balance standard (gem). Each collateral has its ownGemJoin Adapter.

If the collateral token contract being assessed is not similar enough to existing collateral in the Maker Protocol, your team will need to implement a newGemJoin Adapter.

Flipper

TheFlipper contract is where collateral is sent when it is out for auction. Collateral is sourced from Vaults that are liquidated after they become undercollateralized. The sameFlipper implementation is used for all collateral types.

Median and Oracle Security Module (OSM)

TheMedian contract maintains a whitelist of oracle price feeds authorized to post USD price updates for collateral supported in the Maker Protocol.TheOSM contract ensures that new collateral price values are delayed by a specific amount of time. The Oracle Team will handle the deployment of theMedian andOSM , and, once configured, will share its addresses with the Smart Contract team to be called in theDssSpell .

Deployment Scripts

Adding a new collateral type (ilk) to the Maker Protocol requires an upgrade to different repositories in thedss ecosystem, which are responsible for recreating Maker Protocol instances to be rigorously tested before mainnet inclusion. The two repos that are of most interest to you:

- dss-gem-joins
- —This repository holds allGemJoin Adapters
- , apart from the original GemJoinheld in dss
- , which are used with collateral tokens that need extra functionality or constraints.dss-gem-joins
- also holds all collateraltoken contracts
- .
- dss-deploy-scripts
- —This repository contains a set of scripts that deploydss
- to an Ethereum chain of your choosing. It reads theGemJoin
- Adapters, and the token contracts fromdss-gem-joins
- . This contract is used to deploy fresh instances of the Maker Protocol on local testchains and public testnets. For pre-deployed systems, such as the latest version on Kovan and Mainnet, system upgrades are commissioned through Spells and will be described in the Deploy Contracts in section three.

Follow the process below to learn which ones require changes and dependency updates.

Remember: PRs indss-proxy-actions anddss-deploy-scripts should be made after the Greenlight Governance Poll. Additionally, a confirmation of a successful Executive vote (i.e.cast) should be appended as a comment before it is accepted as a new addition and merged intomaster .

Modifying the Maker Codebase

1. Is the token already used in anotherIlk?
2.
    1. Yes (e.g.,ETH-B
3.
    1.): Then there is no need to develop a new adapter. It only requires adding the newilk
4.
    1. todss-deploy-scripts
5.
    1. ; therefore, go directly to step 10d.
6.
    1. No: Review the token contract technical assessment conducted in the first phase of your work.
7. 6.
8. Add a simple version of the token todss-gem-joins
9.
    1. Create a branch in your fork titled TOKEN-X-Collateral (e.g., BAT-A-Collateral)
10.
    1. Ensure the simple version of the token highlights its idiosyncrasies.
11.
    1. It would be best to copy the token contract from an existing mock of another token;BAT
12.
    1. is reccommended. Take care to capture the target token's defining features.
13.
    1. You are going to need this new token later when you write a spell for kovan and updatedss-deploy-scripts
14.
    1. . This is the moment that you want to deploy this token to kovan. You do this now so that if there are any changes you need to make to the token, you don't have to go through the hassle of updating the dependency tree again. Since you deploy everything to kovan, you will need to acquire some kETH (kovan ETH) to deploy contracts and send transaction on kovan. You can ask for some kETH in the #dev channel on the makerdao rocket.chat server:https://chat.makerdao.com/channel/dev
15.
    1. . To deploy this new token, try:
16.
    1.
        1. dapp --use solc:0.5.12 build && dapp create TOKEN (seth --to-uint256 "(seth --to-wei 100000000000 ETH)"), where token is the contract name. (e.g. ForBALwe used :dapp --use solc:0.5.12 build && dapp create BAL (seth --to-uint256 "(seth --to-wei 100000000000 ETH)")`).NOTE:
17.
    1.
        1. this mints 100 billion tokens in kovan.
18.
    1.
        1. Save the kovan address of this new contract for later.
19.
    1.
        1. Go tohttp://kovan.etherscan.io/address/
20.
    1.
        1. #code
21.
    1.
        1. Click Verify and Publish and follow the prompts to verify the contract. We typically use GPLv3 for the license. Make sure you account for the correct compiler version and optimization flags.
22.
    1. 7.
23.
    1. Add the token Solidity / Vyper contract tothis directory
24.
    1. , and commit your work to your branch indss-gem-joins
25.
    1. .
26. 18.
27. Does the token fit with any of theGemJoin Adapters already designed indss-gem-joins?
28.
    1. Yes: Use the most appropriateGemJoin
29.

1. adapter. Followingthis rubric
30.
1. , create a PR of your token addition indss-gem-joins
31.
1. . Go to Step 5.
32.
1. No
33. 6.
34. Implement a new GemJoin Adapter and merge into dss-gem-joins
35.
1. Develop a newGemJoin
36.
1. that addresses the needs of the new collateral token
37.
1. Add a newGemJoinX
38.
1. in ajoin-X.sol
39.
1. file tothis directory
40.
1. .
41.
1. Followingthis rubric
42.
1. , Create a PR of your token addition and new GemJoin adapter indss-gem-joins
43.
1. . Ensure that it gets merged
44. 10.
45. Upgrade dss-proxy-actions that use dss-gem-joins
46.
1. Create a branch in your fork titled TOKEN-X-Collateral (e.g. BAT-A-Collateral) indss-proxy-actions
47.
1. .
48.
1. On your branch, execute the following command in the CLI
49.
1.
    1. dapp upgrade dss-gem-joins && git push
50.
1. 2.
51.
1. Following a simple PR template below, draft a PR of the dependency upgrade:
52.
1.
    1. PR Template
53.
1.
    1.
        1. # Description
54.
1.
    1.
        1. Collateral Onboarding Team Name / Author(s):
55.
1.
    1.
        1. Collateral Type (e.g.ETH-A
56.
1.
    1.
        1.):
57.
1.
    1.
        1. Reason: Updateddss-gem-joins
58.
1.
    1. 6.
59.
1. 8.
60. 15.
61. If we added a newGemJoin, does it have any idiosyncrasies that affect any of the proxy functions indss-proxy-actions?

62.
    1. Yes. Work with the Smart Contract Domain Team to decide:
63.
    1.
        1. What changes are required
64.
    1.
        1. If it’s worth deploying a new proxy actions
65.
    1. 3.
66.
    1. No. Merge the PR that was drafted above
67. 6.
68. Add MedianTOKENUSD in testchain-medians repository:
69.
    1. Create a branch in your fork titled TOKEN-X-Collateral (e.g. BAT-A-Collateral) intestchain-medians
70.
    1. .
71.
    1. Append a new median contract titledMedianTOKENUSD
72.
    1. (e.g.MedianCOMPUSD
73.
    1.) with a newwat
74.
    1. variable to thisfile
75.
    1. .
76.
    1. Following a simple PR template below, create a PR of the file update:
77.
    1.
        1. PR Template
78.
    1.
        1.
            1. **Description**
79.
    1.
        1.
            1. Collateral Onboarding Team Name / Author(s): < name(s) >
80.
    1.
        1.
            1. Collateral Type (e.g.ETH-A
81.
    1.
        1.
            1.): < name >
82.
    1.
        1.
            1. Reason: New MedianTOKENUSD contract
83.
    1.
        1. 6.
84.
    1. 8.
85. 17.
86. Upgrade dependencies in dss-deploy-scripts:
87.
    1. Ensure that all previous PRs have merged
88.
    1. Create a branch in your fork titled TOKEN-X-Collateral (e.g. BAT-A-Collateral) indss-deploy-scripts
89.
    1. .
90.
    1. Execute the following commands
91.
    1.
        1. nix run -f https://github.com/icetan/dapp2nix/tarball/master -c dapp2nix up dss-gem-joins
92.

1.
    1. nix run -f https://github.com/icetan/dapp2nix/tarball/master -c dapp2nix up dss-proxy-actions
93.
    1. nix run -f https://github.com/icetan/dapp2nix/tarball/master -c dapp2nix up testchain-medians
94.
    1. 4.
95.
    1. Verify there are no dependencies imported with different versions. Result of running the following command should be empty value:
96.
    1. nix run -f https://github.com/icetan/dapp2nix/tarball/master -c dapp2nix dup
97.
    1. 2.
98.
    1. Commit your progress to your branch
99. 13.
100. Add a new script to deploy the newilk and update configs in dss-deploy-scripts:
101.
    1. Note:
102.
    1. If you're adding a new version of a collateral type (e.g.ETH-B
103.
    1.), you will not need to make a new deploy ilk script, and you can go straight to the last sub-step, which is to update the environment config files.
104.
    1. In the same branch as before, create a new file calleddeploy-ilk-token
105.
    1. and add it tothis directory
106.
    1. .
107.
    1. With this script,dss-deploy-scripts
108.
    1. will pull the token contract andGemJoin
109.
    1. adapter fromdss-gem-joins
110.
    1. , use the latest auction contract (Flipper
111.
    1.), use an OSM, configure all authorizations, deploy all contracts, and enable the collateral type in a fresh instance of the Maker Protocol.
112.
    1. deploy-ilk-usdc
113.
    1. is a good example to use.
114.
    1.
        1. Replace the token symbol
115.
    1.
        1. Ensure that the last line is referring to the correctGemJoin
116.
    1.
        1. . Remember, your new collateral is going to use a previous adapter or the one you made
117.
    1. 4.
118.
    1. Remember to set the new script to be executable
119.
    1.
        1. chmod +x deploy-ilk-token
120.
    1. 2.
121.
    1. Add the new collateral type in all environment config files inthis directory
122.
    1. .
123.
    1.
        1. Use the historic code snippets for easy addition
124.
    1.

1. For risk parameters, look to the Collateral Onboarding Risk Evaluation posted in the forums. For example BAL was: https://forum.makerdao.com/t/bal-collateral-onboarding-risk-evaluation/4600

125.
1.
1. For the kovan.json
126.
1.
1. file, under gem
127.
1.
1. , you can use the token address we got from the kovan deploy earlier.
128.
1. 6.
129. 29.
130. Make a PR into dss-deploy-scripts
131.
1. At this point, we are still missing the mainnet pip
132.
1. that's needed from the Oracle team. This contract usually undergoes extensive testing before we get it. For now, we can make a PR, but put it in draft form until we get that pip
133.
1. address.
134.
1. Following this simple rubric, submit a PR of the Collateral Type addition in dss-deploy-scripts
135.
1. :
136.
1.
1. PR Template
137.
1.
1.

1. # Description

138.
1.
1.
1. Collateral Onboarding Team Name / Author(s):
139.
1.
1.
1. Collateral Type (e.g. ETH-A
140.
1.
1.
1.):
141.
1.
1.
1. Reason: Added TOKEN-X
142.
1.
1.
1. collateral type
143.
1.
1. 7.
144.
1. 9.
145. 15.
146.

1. Deploy Collateral Type on Kovan

So far, you've worked on updating the Maker Protocol codebase to add support for the new collateral type. The updated deploy scripts are used to deploy the protocol from scratch. They will also be useful for future complete mainnet, testnet, or local test deployments.

The following sections contain the steps required to create and propose transactions to governance that can modify the operational Kovan and Mainnet Maker Protocol deployments to introduce the new collateral type.

Adding a collateral type to the latest protocol instance on Kovan is required for end-to-end testing among other ecosystem participants, such as analytics dashboards and Keepers. The Kovan testnet should be viewed as an experimental sandbox where Collateral Onboarding teams can deploy their contracts in a public domain that closely resembles Ethereum's Mainnet. After the governance poll,

as described in Step ___ of Section __, adding a new collateral type should not be impeded by any external procedure, apart from the Smart Contract Domain Team reviews of PRs made in the previous stage.

What is a spell?

The Maker Protocol contains certain functions used to update parameters, such as collateral types, debt ceilings, rates, etc.

Through the DS-Chief contract, Maker Governance is controlled by MKR holders and is designed to elect an address, thehat , which is then trusted to execute these functions and make changes to the live protocol.

Instead of trusting individuals or multisig wallets, governance always votes on spell contracts, which cannot be modified after deployment and can execute a series of these function calls with predefined inputs. This spell allows both the members of governance as well as the larger ecosystem to precisely understand and align on the changes being introduced to the protocol prior to its execution without having to place trust in any single actor.

All actions performed by MKR governance also go through a timelock contract called the Governance Security Module (GSM, technically known as the PauseProxy) which turns all executions into a two-step process. Spells are also built to work with the first step, which schedules execution for later byplotting aplan and then the actual execution after the delay bycasting the spell, which introduces the desired changes to the live Protocol.

While many different types of spells are routinely cast to introduce changes, you will primarily focus on creating a spell that introduces a new collateral type to the protocol.

A Spell Contract has the following structure:

- Abstract contract versions of various contracts whose functions are to be executed are imported.
- The implementation of the spell is contained in DssSpell.sol and the tests are present in DssSpell.t.sol
- At least oneSpellAction
- contract is defined which contains the main execute function and aDssSpell
- contract contains a description of the spell and orchestrates the execution of one or more spell actions through the GSM.
-

Github Repository Processes

1. All teams must submit their final work to thespells-kovan repo
2. for review.
3. When you start a PR on this repo, a checklist is automatically imported. Please do not make any changes to the checklist and ensure all checks are complete after the file is modified to include the new spell. Also ensure that the code and tests are approved by at least two reviewers from the Maker Foundation smart contracts domain team.
4. After the review, the spell will be deployed and tested on Kovan before beginning the process of mainnet deployment.
5. Ensure that the deployed spell contract is verified on Etherscan.
6. Tests need to be modified to reflect the deployed spell contract address and block timestamp as well.
7. After execution on Kovan, a copy of the spell and the test file must be made in the archive folder within the PR before it gets merged. The date of the spell sol file is the date on which the spell was executed.
8.

Deploy contracts

Take the steps below to prepare the environment before an executive spell is added to the existing DS-Chief and Maker Protocol. Prompts in the form of questions will guide you.

1. Is there an existing collateral token contract in the environment that the community uses?
2.
    1. Yes: Use that one
3.
    1. No: Deploy a new token contract on Kovan (you should have done this in the last section) and supply freshly minted tokens to theFaucet contract
4.
    1. , so that other users can have access for kovan testing.
5.
    1. To send tokens to the faucet:
6.
    1.
        1. Send 50bn tokens (50% of token supply) to the faucet
7.
    1.
        1.
            1. Example forBAL
8.
    1.
        1.
            1. :seth send 0x630D82Cbf82089B09F71f8d3aAaff2EBA6f47B15 'transfer(address,uint256)' 0x57aAeAE905376a4B1899bA81364b4cE2519CBfB3 500000000000000000000000000000
9.
    1.
        1. 3.

10.
    1.
        1. Notify the Smart Contract Domain team that a new token is in the faucet, and let them know how large thegulp
11.
    1.
        1. amount should be.
12.
    1. 7.
13. 12.
14. Deploy the GemJoin adapter
15.
    1. Follow standard practices for contract deployment with the GemJoin adapter found/developed in the previous section
16.
    1. Ensure that theVat
17.
    1. ,ilk
18.
    1. ,gem
19.
    1. , and any other constructor values are set correctly.
20.
    1. Before deployment, confirm above addresses with those in the latest system, as found on thechangelog.makerdao.com
21.
    1. Example deployBAL
22.
    1. with standardGemJoin
23.
    1. :
24.
    1.
        1. Example forBAL
25.
    1.
        1. :dapp --use solc:0.5.12 build && dapp create GemJoin 0xbA987bDB501d131f766fEe8180Da5d81b34b69d9 (seth --to-bytes32 "(seth --from-ascii "BAL-A")") 0x630D82Cbf82089B09F71f8d3aAaff2EBA6f47B15
26.
    1.
        1. Save this address for later
27.
    1.
        1. If this GemJoin has already been used, then you will NOT need to verify the contract on etherscan. However, if you have deployed a brand new GemJoin adapter, please verify it now
28.
    1.
        1.
            1. Go tohttp://kovan.etherscan.io/address/
29.
    1.
        1.
            1. #code
30.
    1.
        1.
            1. Click Verify and Publish and follow the prompts to verify the contract. We typically use GPLv3 for the license. Make sure you account for the correct compiler version and optimization flags.
31.
    1.
        1. 4.
32.
    1. 9.
33. 19.
34. Deploy the Flipper contract
35.
    1. Locate theFlipFab contract
36.
    1. in the changelog.
37.
    1. Ensure that theVat
38.
    1. ,Cat
39.
    1. , andilk
40.
    1. values are set correctly.

41.
    1. Before deployment, confirm above addresses with those in the latest system, as found on thechangelog.makerdao.com
42.
    1. CallnewFlip()
43.
    1. in theFlipFab
44.
    1. contract to deploy a generic Flipper contract. You can find theMCD_FLIP_FAB
45.
    1. address in the changelog along with the argumentsVat
46.
    1. andCat
47.
    1. . You can get the bytes32 of the ilk from the command below, but replaceBAL-A
48.
    1. with your token.
49.
    1.
        1. Example forBAL-A
50.
    1.
        1. :TX=(seth send 0x7c890e1e492FDDA9096353D155eE1B26C1656a62 'newFlip(address,address,bytes32)(address)' 0xbA987bDB501d131f766fEe8180Da5d81b34b69d9 0xdDb5F7A3A5558b9a6a1f3382BD75E2268d1c6958 (seth --to-bytes32 "(seth --from-ascii "BAL-A")") --async) && seth receipt TX logs | jq -r '.[0].address'
51.
    1.
        1. Frustratingly,seth send
52.
    1.
        1. doesn't return the new Flipper address. You can find this address by chasing it down in the transaction hash that was returned. Simply go tohttps://ethtx.info/kovan/
53.
    1.
        1. .
54.
    1.
        1.
            1. BAL-A
55.
    1.
        1.
            1. example:https://ethtx.info/kovan/0x5881ca053134ce6cc76438a3615ab3033f31bc9620a9032f696bd171914dae5a
56.
    1.
        1. 3.
57.
    1. 9.
58. 24.
59. Set Authorizations
60.
    1. Set the wards permissions of both new contracts. In general, it should be calls for each contract, giving authorizations (rely) to other contracts to make calls on authorized functions, and a final release of authorization (deny) of the deployer address. Use the corresponding addresses from the changelog with the variables below.
61.
    1.
        1. seth send "MCD_JOIN_TOKEN_LETTER" 'rely(address)' "MCD_PAUSE_PROXY"
62.
    1.
        1. seth send "MCD_JOIN_TOKEN_LETTER" 'deny(address)' "ETH_FROM"
63.
    1.
        1. seth send "MCD_FLIP_TOKEN_LETTER" 'rely(address)' "MCD_PAUSE_PROXY"
64.
    1.
        1. seth send "MCD_FLIP_TOKEN_LETTER" 'deny(address)' "ETH_FROM"
65.
    1. 5.
66. 7.
67. Receive the OSM contract address from the Oracle Team
68.
    1. The Oracle Team will handle the deployment of theOSM
69.
    1. and will share its address with the Smart Contract team to be called in theDssSpell
70.

1. .
71.
    1. Note: Some stablecoins will use a fixed `DSValue` contract instead of anOSM
72.
    1. .
73. 6.
74.

All actions executed by governance need to go through the GSM and can only be executed after a delay, which is currently set to 12 hours.Mom contracts allow governance to execute certain time-sensitive actions without having to pass through a delay.

TheOSMMom contract allows governance to stop an erroneous oracle price update instantly, and theFlipperMom contract allows governance to instantly stop vaults of a collateral type from being liquidated under unfavorable conditions.

Please re-check the authorization permissions for all newly deployed contracts. After deployment of theGemJoin andFlipper contracts, the deployer address (ETH_FROM) is given authorization, which needs to be revoked with a follow-updeny transaction.

Initialize Spell Action

Setup your spell

1. Create a new branch on the spells-kovan repo namedTOKEN-X
2. .
3. Copy thespell action template
4. in the./template
5. directory to add the new collateral type with values fromhttps://github.com/makerdao/dss-deploy-scripts/tree/master/config
6. , and then replace the contents of thesrc/Kovan-DssSpell.sol
7. contract with it. It is important that the template is exactly followed and no changes are made since it was created after many iterations. Reviews would be conducted against it line by line and they would fail if any changes are introduced to the template.
8.

Update addresses

The first portion of a SpellAction contract stores the contract addresses referred to in the execution process later, and various math units both of which are saved as constants without a public modifier.

Omitting the public modifier removes the default getter function from the deployed contract, which reduces its size and saves gas during deployment, which sometimes could exceed the limits set by Ethereum for all contracts.

Spell actions are executed within the scope of the GSM(Pause Proxy) through a delegatecall.

GSM(Pause Proxy) executes the contents of the spell through a delegatecall, which runs the entire execution within its own contract memory scope. Setting the spellaction variables as constant prevents them from being overwritten within the memory of Pause Proxy and removes any hazards during the execution. Hence, all variables within the Spell Action need to have a constant modifier if they are stored in the contract scope.

Please copy and add the right address values from the latest changelog at this link-https://changelog.makerdao.com/releases/kovan/latest/contracts.json

Each address value is copied from changelog and manually added to the SpellAction.

You can use the address of the deployed collateral token on Kovan and use the MCD_OSM address for PIP. For stablecoin collateral, it could be the address of a deployed DSValue contract as well.

Please note that this manual process might be automated in the near future and the addresses can be retrieved directly within the spell contracts using user-friendly keys when the relevant addresses are stored on-chain and under the control of Maker governance.

Rate Calculation

Governance uses annual rates to discuss changes, Ex: 4%, 5%, etc., while the rate variable used by the Maker Protocol requires a per-second rate. It is your job to correctly translate one into the other.

Please update the bc command-line tool and run the command present in the template following the instructions to calculate the number and store it as a constant to refer to later.

Configure collateral type name

All collateral types within the protocol are assigned a unique identifier which consists of the token ticker symbol, Ex: ETH, WBTC, USDC, and a character denoting the different collateral types available for the same token, Ex: USDC-A, USDC-B.

Please replace the words TOKEN_LETTER in this template to reflect the identifier assigned to the collateral type being on-boarded.

Sanity checks

Sanity checks in the execute function ensure the deployed Join and Flip contracts being used for the collateral type are correctly configured with the same core contract addresses as well.

Collateral Type Configuration

Configuration of the collateral type is stored across multiple contracts and each one would need certain changes.

1. Spotter contract is configured with the Pip oracle contract of the collateral type.
2. Cat contract is configured with the Flip collateral auction contract of the collateral type.
3. Collateral type initialization steps are performed in both the core contract Vat and the rates contract Jug.
4.

Authorizations

Every smart contract within the Maker Protocol maintains a whitelist of other relevant contracts which can execute certain functions guarded by the auth modifier. Some of these permissions need to be modified to give the newly deployed contracts permissions to operate and support the collateral type.

- Vat allows the adapter contract to add or remove internal token balances.
- Cat allows the Flip contract to reduce the on-auction debt number after an auction concludes.
- Flip contract allows Cat to start new collateral auctions.
- Flip contract allows Emergency shutdown contract End to yank(cancel) running auctions.
- Flip contract allows FlipperMom contract, which allows governance to sidestep the GSM and execute certain actions.
-

When you notice a number of exclamation marks, please read the comment following the word Only carefully and either leave the following code line within the Spell or remove it based on the conclusion.

Some collateral types have liquidations disabled and this is done by removing the permission for Flip contract within the FlipperMom contract.

OSM Configuration

Please use the details received from the Oracles Domain Team to update the configuration in this section.

Oracle Security Module(OSM) introduces a delay to the oracle value computed by the Median contract. It stores the delayed and active oracle value as well as the upcoming value which will soon become active.

OSM configuration is skipped for certain collateral types and the following steps should be removed from the spell. This is done when the collateral type is for a stablecoin which does not need an oracle, or if the OSM was already configured for another collateral type.

1. OSMMom is authorized in the OSM to ensure malicious price feed updates can be stopped.
2. OSM is whitelisted within the Median contract.
3. Spotter contract is whitelisted within the OSM.
4. End contract is whitelisted within the OSM.
5. OSMMom contract is updated with the OSM and collateral type details.
6.

A few steps need to be performed off-chain as well to ensure the infrastructure is ready to be used.

You canpoke() the OSM yourself by using the TOKEN_PIP address:seth send TOKEN_PIP 'poke()' . Feel free to do this twice, but you must wait an hour between eachpoke() . Once you've poked, you can run this script to see if there is a price in the OSM:https://gist.github.com/godsflaw/3dd9171237a88caa948fc5879431d0ce

After your second poke you should seecurrentPrice andnextPrice populated. If you don't, you should reach out to the Oracles team to ensure they are indeed providing a price.

Update parameters

This section references this portion of the template:https://github.com/makerdao/spells-kovan/blob/master/template/SpellAction.sol#L83-L102

1. Get the required risk parameters from the Risk Domain Team before updating this section.
2. Please use the math units stored to make the values clear to the readers.
3. Set the global debt ceiling.
4. Set the collateral type debt ceiling.
5. Set the collateral type dust value.
6. Set the collateral auction lot size.
7. Set the liquidation penalty. Ensure that the right number is used instead of the exact percentage number.
8. Set the stability fee using the previously stored rate constant.
9. Set the beg value.
10. Set the bid expiry value.
11. Set the collateral auction expiry value.
12. Set the collateralization ratio value.
13. Execute Poke within Spot to ensure the correct value is updated within Vat.
14.

Update ilk registry

Add the collateral type to the Ilk Registry contract which acts as an on-chain reference for other smart contracts.

Configure the faucet (kovan)

Integrators need to test this new collateral type. They will do this by callinggulp() on the faucet. This section tells the faucet how many tokens should be handed out for agulp() . Oncegulp() is called it cannot be called from the same address again, so be generous. For testing later, this MUST be more than the DAI dust value with collateralization ratio applied. Target 5x the dust value.

Update DSS Spell

1. Update the description of the spell and add the calculated hash value as well.
2. If needed, uncomment and activate the officeHours modifier as well which stops the spell from being executed on a weekend and outside office hours.
3.

License

Please ensure that the AGPL license is applied to your work to give others maximum flexibility to make changes if needed.

Spell Testing

The tests performed are integration tests which perform various checks against the current state of the deployment on chain, and are not unit tests which only test the accuracy of just the spell logic. They verify that the existing state of the chain matches the numbers present, vote and execute the spell, fast forward time to the point at which the spell can be cast, execute the spell and check the new state of the system including all the collateral types along with other items.

This ensures that the code is correct when applied to the current chain state.They are set up to explicitly enumerate all of the collateral config, while testing it is easy to see if there is a change to any parameters, or magnitude changes.

Integration tests also check whether limits for spell deployment and execution also exceed and if they will pass or fail.

Make a test for collateral addition

Using the test template located here:https://github.com/makerdao/spells-kovan/blob/master/template/SpellAction.t.sol

Update the test spell with the correct values:https://github.com/makerdao/spells-kovan/blob/master/src/Kovan-DssSpell.t.sol

CI process

Both spells-mainnet and spells-kovan repositories have a CI process built in which runs the tests for all pushed changes to the PR and reports the result.

Deployment and execution

The kovan spell deployment and execution will be performed by the Maker Foundation smart contracts team after a successful review.

Please look intodapp for detailed information on how it works. Building the project is done with:

dapp build --extract

dapp also does not support all solidity versions across all architectures. Please check thereleases link to understand which solidity versions are supported.

The most common issue is going to be compiling the spell with the older solc version that the dss codebase currently uses. You can either pass a command line flag or create a .dapprc file in the root directory and include DAPP_SOLC_VERSION=0.5.12 (or your desired solidity version) within it to instruct dapp to use that solc version while building the project. Dapp will automatically fetch this solc version and you will not have to install it manually.

The --extract command line flag creates necessary build artifacts like ABI and bin files and places them in the out directory.

Next step is to deploy the contract to the network of your choice. Please configure your.sethrc file with the correct network, gas limit, and signing keys. You can also add the Etherscan API key to the file to .sethrc for automatic etherescan contract verification, ETHERSCAN_API_KEY=XXXXXYYYYYZZZZ.

The command dapp create --verify will then deploy the contract and try verifying it on Etherscan using the provided API key.

You can also flatten the contract along with its imports into a single file using the command hevm flatten --source-file src/Kovan-DssSpell.sol to easily verify it on Etherscan manually.

1. Deploy Collateral Type on Mainnet

After the deployment on Kovan is successful and testing is complete, the team can move on to the mainnet deployment process after the appropriate green light is received from governance.

Changes to mainnet are made using a governance process that follows a precise timeline. The spell details are introduced to the governance community for a vote on a Friday.

Typically a new PR is opened in the spells-mainnet repository on the Monday of the same week. You are expected to submit only the SpellAction contract to the PR and the foundation smart contracts domain team will integrate its execution along with other spell actions into a DSSSpell contract.

After a successful vote and execution, the PR on spells-mainnet repo will also get merged and the spell is archived for future reference.

Emergency spells

Certain emergency spells are currently deployed on the mainnet and can be used immediately in emergency situations.

Please coordinate with the foundation team to check if new collateral type addition requires changes to any emergency spells. For example, a collateral like GUSD is likely to be deployed with liquidations off. This means it needs special treatment in the emergency spells, and the foundation team will need to schedule this work.

Previous Collateral Technology Next Collateral Onboarding Spell Crafting Guide Last updated1 year ago On this page *Directory * 1. Introduction * 2. Dapptools * 3. Collateral Token Smart Contract Technical Assessment* 4. Add collateral type to Maker Protocol codebase * Contracts associated with Collateral Types * Deployment Scripts * Modifying the Maker Codebase * 5. Deploy Collateral Type on Kovan * What is a spell? * Github Repository Processes * Deploy contracts * Initialize Spell Action * Update DSS Spell * Spell Testing * 6. Deploy Collateral Type on Mainnet * Emergency spells

Was this helpful?