dForce USR Scheme

❖ Revision History

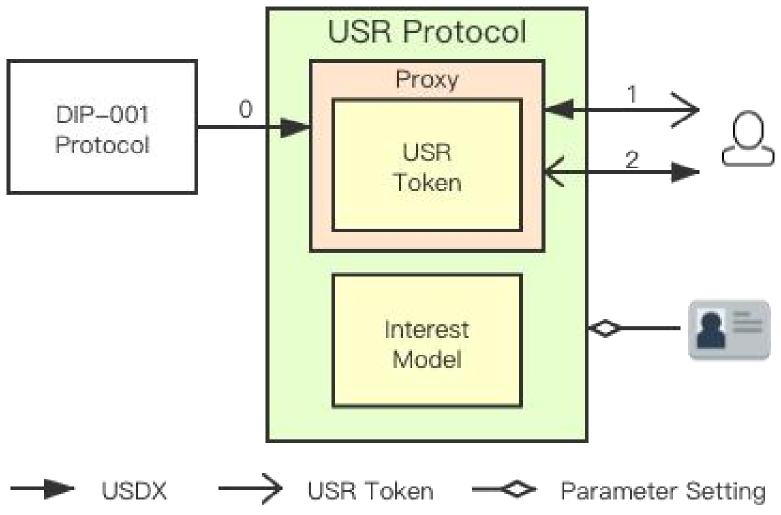
Date	Contents	Author
2020.2.25	v0.1	Horsen
2020.2.26	v0.2	Horsen
2020.2.27	v0.3	Skyge
2020.2.28	v0.4	Skyge
2020.3.4	v0.5	Skyge
2020.3.6	v0.6	Horsen
2020.3.9	v0.7	Skyge
2020.3.13	v0.8	Margaret
2020.3.16	v0.9	Horsen
2020.3.17	V1.0	Margaret

❖ What is USR

USDx is a synthetic stablecoin pegged into a basket of stablecoins. Through the implementation of DIP001, the majority of USDx constituent stablecoins will be unlocked and re-supplied to decentralized lending protocols for yield generation, and a fraction of interest earned will be pooled to fund the USR.

USDx holders can deposit USDx into the contract of USR (USDx Saving Rate) in exchange for USR token. Each USR token represents a stake in the USR pool and a pro rata claim of redeemable USDx. Those users are then able to withdraw capital plus interest from the pool at any time.

❖ Architecture



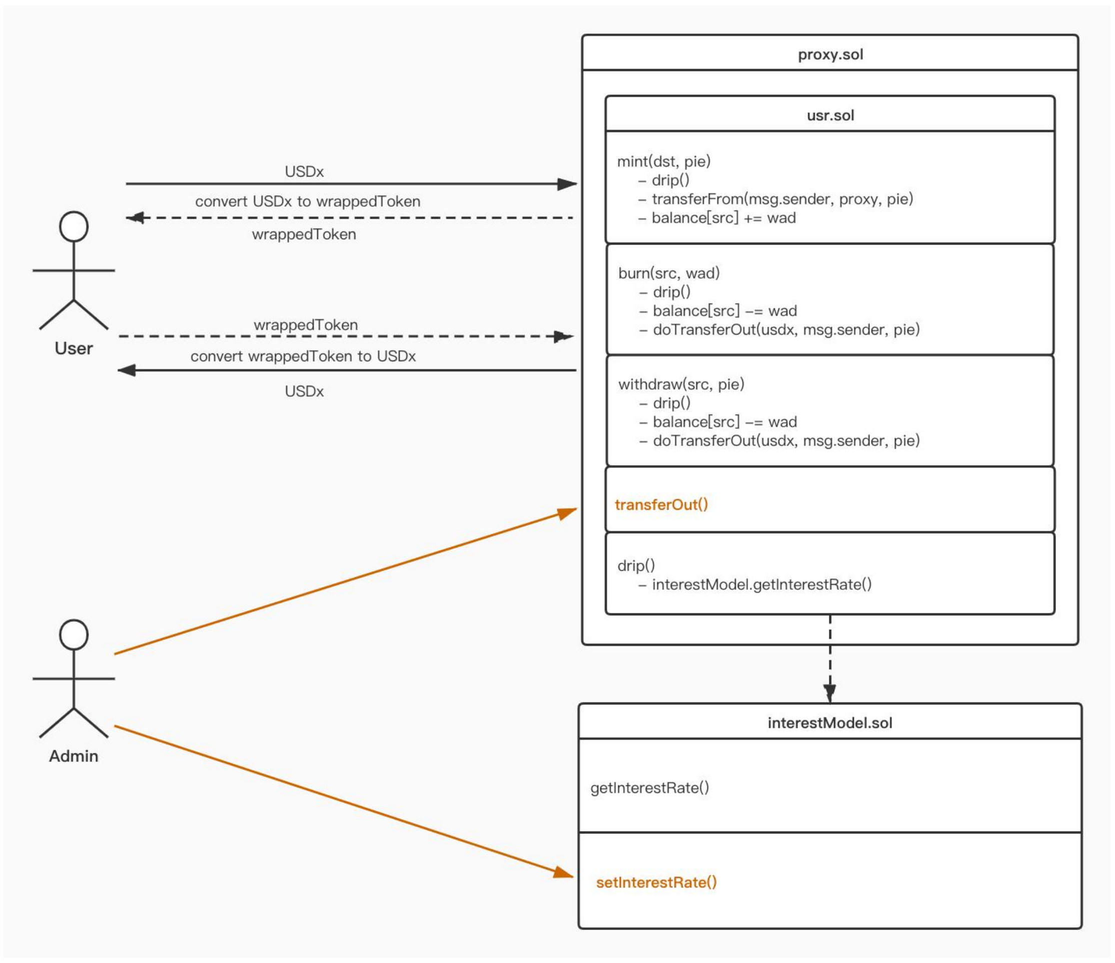
As illustrated in the diagram above, the USR Protocol is composed of a Proxy for USR Token and the Interest Model (both are upgradable). The arrows show the direction of token circulation and exchange.

User deposits USDx into the USR contract in exchange for USR Token based on the real-time price calculated by the Interest Model Contract. Each USR Token represents a pro rata claim of USDx in the pool (for example, 10,000 USDx at the rate of 1.001 is redeemable for 9,990.01 USR Token). At the end of the exchange, USDx will be transferred to the USR Core Proxy.

The implementation of DIP-001 allows the constituent stablecoins of USDx be unlocked and resupplied to yielding protocols for interest generation. Part of accrued interest will be used to mint USDx and be returned to the USR contract.

Vice versa, when a user wants to exist and redeem USDx from the USR contract, it will be calculated based on the real-time price of USDx/USR. USDx will be unlocked from the USR contract and returned to the user, while pro rata USR Token will be burned simultaneously.

❖ Contracts and Its Public Interfaces



1. USR.sol

P.S.:

i. The deployer is the initial owner of the contract;

ii. The USR contract is of ERC20 standard with all basic functions: approve(), balanceOf(), transfer(), transferFrom(), so we will give it a pass here.

1) mint(_dst, _pie) external:

Deposit a specific amount of USDx in return of USR Token on a pro rata basis.

2) burn(_src, _wad) external:

Burn a specific amount of USR Token in return of USDx on a pro rata basis. Please note a small amount of fee will be charged in this relation.

3) withdraw(_src, _pie) external note whenNotPaused:

   Redeem a specific amount of USDx by burning pro rata USR Token.

4) getExchangeRate() public view:

   Calculate the real-time exchange rate from USDx to USR Token.

5) getTotalBalance(_account) external view:

   Calculate the real-time total deposit for a specific account, inclusive of principal and interest earned and exclusive of fees.

6) share() external view:

   Get the maximum amount of USDx that the user can deposit into the contract.

7) equity() external view:

   Calculate the number of USDx in the contract without USR entitlement. Value greater than 0 represents a remainder of USDx in the contract, vice versa, more USDx will need to be supplemented to the contract.

8) drip() public:

   Update real-time price of USDx/USR Token.

9) takeOut(address tokenAddress, address receiver, uint256 amount) external onlyManager whenNotPaused:

   Transfers token from the contract to a specific address.

10) updateInterestModel(address _interestModel) external note onlyOwner:

    Update the address for the Interest Model Contract.

11) setMaxDebtAmount(_newMaxDebtAmount) external onlyOwner：

    Update the total amount of USDx that can be accommodated by the USR contract.

12) updateOriginationFee(uint _newOriginationFee) external onlyOwner

    Update the redemption fee rate.

2. InterestModel.sol

P.S.: The deployer is the initial owner of the contract.

1) setInterestRate(uint _interestRate) external onlyManager;

Set interest rate increasement per second.

2) getInterestRate() external view returns (uint);

   Read the pre-set interest rate increasement per second.

3. Ownable.sol

P.S.:

i. This is a base class contract.
ii. The deployer is the initial owner of the contract.

1) transferOwnership(address pendingOwner_) external onlyOwner:

   Transfer the ownership of the contract to a new account.

2) acceptOwnership() external;

   Accept the ownership transfer.

3) setManager(address account) external onlyOwner

   The contract owner delegates management access to an account.

4) removeManager(address account) external onlyOwner

   The contract owner removes an existing manager.

4. Pausable.sol

P.S.:

iii. This is a base class contract.
iv. The deployer is the initial owner of the contract.

1) pause() public whenNotPaused onlyOwner

   Admin function to pause the contract in an emergency.

2) unpause() public whenPaused onlyOwner

   Admin function to re-activate the contract.