Flexible Loan System

# What is it

This system by its nature would be an on-chain debt ledger.

# Goals

## To lower costs

It aims to move the complex debt issuance on blockchain by leveraging the transparency and the concept of decentralization to attract investors with lower costs.

## To increase debt liquidity

The stake registered by investors can transfer to others any time before the stake gets maturity.

# Who would be the user

Organization can utilize the system on blockchain to record and showcase the debt situation.

Investors can do analysis and register their stakes on chain so to make the debt investments.

# How the system works (Simplified version)

Further details on KT\_ledger:

Generally, the debt principal would be in a number of unsigned integer. To make it further simplified, the conventional concept of separating long-term and short term debt are not considered for fear of complex business logic related to cash flow discount and etc. Also the interest rate is not customized into each creditor log instead it is put as a generic rate, although in reality a more flexible system may require it to be tailored to different fund sources.

It would look nicer if the KT\_ledger could call the entry points of Kt\_token directly. If this is too complicated, to have tz1 call different KT contracts can be an alternative.

Further split on the stake when investors in reality may have the need to transfer partial ownership is not considered here.

Specification focuses on KT\_ledger:

1. Capital in place is a reference value for investors to estimate capital sufficiency.
2. Make KT\_ledger spendable so to have creditors spend tezos as commission fee if the current creditor wants to transfer the ownership to another one.
3. removeCreditor: it is to remove the specified mapped item, as well as modify current capital in place and the total debt amount It returns success or failure of the operation. It is not an entrypoint.
4. checkPoint: it should be entrypoint used by frontend to run in a regular basis to make sure expired creditors are removed in time. More importantly, it is intended to set to trigger token issue function (the proxyTokenTransfer).
5. modifyOwnership: When certain creditor on the mapped list invokes modifyOwnership and sends specified commission fee (in tezos) to KT\_ledger. It would modify the owner of certain debt record [basically it needs to remove the orignal one and insert a new creditor record with the updated start date (which is now) and the same end date as the removed record ]. It returns success or failure of the operation.
6. addCreditor: The function addCredtor would simply add the new creditor as well as the total debt amount. Note for the data point “start date” in the record data structure, which would always be now. It returns success or failaure of the operation
7. proxyTokenTransfer\_ is under the assumption, the KT\_ledger can call entrypoints of Kt\_token; In this case, for example , when removeCreditor is invoked, it would further call proxyTokenTransfer transfer(creditor address,\_\_paymentCalculation)) to transfer tokens

Further details on Kt\_token:

It’s basically the token issuance system on the base of FA1.2; Only the xtz need to be sent and withdrawal when token gets minted or burnt; Due to non applicable on float numbers, the token is 1 million amplified to pegged to USD

Specification focuses on Kt\_token:

1. The mint function would have a QC rule in the front end programming to make sure enough tezos collateral has been transferred to the Kt\_token contract;
2. The burnt function is modfied for simplicity, i.e. to be based on all or nothing principle; The amount withdrawal by triggering the burn function is calculated off-chain; also getBalanceValue would need to be utilized here;
3. Again, for burn function, the withdrawal amount is calculated off chain and send the amount

# Limitation and unsolved issues in general

1. The process flow can be much more smooth if the origination of KT\_ledger can automatically add itself to the list of Kt\_token for token issuance; It’s totally feasible, however as the clock’s ticking, it is not implemented in this version. More manual work at the start is required.
2. The burn funciton can be compromised by front running by manipulating xtz price; The contigency plan would be to apply the average score system by removing 2 highest last prices and 2 lowest last prices in the popular exchanges; And use the rest of the averaged middle prices;
3. Again when to call burn function, a view function and xtz price API would be used, at the same time it leaves space for malicious hacking;
4. Due to limited time, other functions are not developed or polished;

# Frontend Implementation

(based on <https://hackernoon.com/build-your-first-dapp-on-tezos-rwgl3ymb>)

Without the debt owner login, (i.e., Tezbridge connect to other addresses), only transfer ownership and withdrawal (this is the burn function in Kt\_token) would be provided;

If debt owner’s address is connected, collateral (in essential, it’s mint function) would show up.