Documentation

The document outlines the public functions that create and modify loans in the new token contract and token manager. To begin, add a test balance to your address using this function in the new token contract:

```
function addTestBalance(uint256 amount) public; //Deposits to msg.sender
```

This new balance will then be viewable as a custom token with the following attributes:

Name: beta
Symbol: BETA
Decimals: 18

The testnet contract address of the new token will be printed to your console in all scenarios. In all contracts, token balances are referenced and stored in 10^{-18} tokens. Interest, reward and remaining amount due are referenced and stored in the same manner. To request a loan, use the public function in the token manager:

```
requestLoan(address lender,uint256 amount,uint256 length,uint256
interest,bool willMine,string loanMessage) public returns(uint256);
```

Relevant parameter(s):

Input	Description
address lender	Lender's address.
uint256 amount	Loan principal in 10^-18 tokens.
uint256 length	Number of Ethereum blocks.
uint256 interest	Total interest charged in 10^-18 tokens.
bool willMine	Whether the lender and borrower wish to mine after repayment.
string loanMessage	A message regarding the purpose of the loan.

The function returns the index of the new loan request in the global loan array. Nonsensical loan parameters like zero principal or loan requests above the theoretical

maximum supply cause the request function to revert. To begin a loan period, a lender calls this function in the token manager:

```
function attemptBeginLoanAtIndex(uint256 index) public returns(bool);
```

Relevant parameter(s):

Input	Description
uint256 index	The index of the loan in the global loan array.

The lender must have a non-holding balance of at least the loan principal, which is then automatically moved to the borrower's address. This function returns a boolean value that denotes whether the loan was initiated.

Paying Back A Loan

A borrower can pay back principal and interest in arbitrary amounts. The token manager will track the amount paid back over time. To pay back part or all of a loan, use the token manager function:

```
function payAmountForLoanAtIndex(uint256 amount,uint256 index) public;
```

Relevant parameter(s):

Input	Description
uint256 amount	The amount the borrower is paying back in 10^-18 tokens.
uint256 index	The index of the loan in the global loan array.

If you've paid back both principal and interest, the loan will change its status from active to either completed or holding. The amount being paid back must be greater than zero, and the payment must come from the borrower. If the borrower fails to have a sufficient balance not in holding, this function will revert. In order to mine, the lender must have a non-holding balance equal to or greater than the loan amount. If the borrower pays back a loan late, the lender can still mine and claim all reward.

Claiming Rewards

Before claiming rewards, be sure you've increased the mineable amount of the new token contract. In the final contracts, the converter contract will automatically set

the mineable amount when ELIX is converted to the token. For now, use the following function in the new token contract:

```
function increaseMineableAmount(uint256 amount) public;
```

To distribute rewards for mining, either the lender or borrower use the token manager function:

```
function requestRewardForLoanAtIndex(uint256 index) public returns(bool);
```

Relevant parameter(s):

Input	Description
uint256 index	The index of the loan in the global loan array.

The appropriate rewards are automatically deposited to the lender and borrower addresses. Other relevant aspects of the loan like status are also updated. The holding period must be completed in order to claim a reward. The function returns a boolean value denoting whether rewards were distributed.

The following loan statuses are used within the token manager:

Loan Status	Usage
1	Requested
2	Active
3	Holding
4	Request canceled by borrower
5	Request canceled by lender
6	Borrower paid late
7	Completed
8	Active loan canceled by lender

Loan values are stored as follows:

```
struct loan   {
    address borrower;
    address lender;
    uint256 startBlock;
    uint256 amount;
    uint256 paidBackBlock;
    uint256 status;
    uint256 amountPaidBackSoFar;
    uint256 loanLength;
    uint256 borrowerReward;
    uint256 lenderReward;
    uint256 interest;
    bool willMine;
    bool borrowerPaidLate;
    string message;
}
```

Final Notes

The rewardFactor in this token manager produces approximately 2.2% yearly inflation if 20% of the current supply is being mined at any given time. There is a max cap of tokens that can be mined, currently equal to the amount of tokens produced by the converter. Furthermore, 65% of rewards go to lenders, and 35% to borrowers. These constants are subject to change moving forward, and can be viewed as temporary. The reward feature has been implemented due to popular demand after conducting our recent product survey. We will also be building additional contracts to facilitate crowdfunding, and will provide more information and details as we progress.

To represent loans with multiple installments, you can create several loan objects between two addresses. A frontend UI can organize these installments, or you can pass information about each installment into the descriptions on a blockchain level.

No reward is received by either party in the cases of partial or total default.

“BETA” and “beta” are placeholders.

If you are a developer and are interested in working on ELIX, let us know. We’re looking to expand our team and will likely hire at least one additional developer in the near future.