Jobs

Quick Start Examples

Simple Keeper

To setup a keeper function simply add the following modifier;

```
modifier keep() {
  require(KPR.isKeeper(msg.sender), "::isKeeper: keeper is not registered")
  _;
  KPR.worked(msg.sender);
}
```

The above will make sure the caller is a registered keeper as well as reward them with an amount of KPR equal to their gas spent + premium. Make sure to have credit assigned in the Keep3r system for the relevant job.

Adding Jobs

Jobs can be created directly via governance or by submitting a job proposal to governance automatically via adding liquidity.

Submit a job via governance

Simply create a new proposal via governance to add a new job

```
/**
 * @notice Allows governance to add new job systems
 * @param job address of the contract for which work should be performed
 */
```

```
function addJob(address job) external;
```

Submit a job via adding liquidity

You will need to provide liquidity to one of the approved liquidity pairs (for example KPR-ETH). You put your LP tokens in escrow and receive credit. When the credit is used up, you can simply withdraw the LP tokens. You will receive 100% of the LP tokens back that you deposited.

```
/**
 * @notice Allows liquidity providers to submit jobs
 * @param liquidity the liquidity being added
 * @param job the job to assign credit to
 * @param amount the amount of liquidity tokens to use
 */
function addLiquidityToJob(address liquidity, address job, uint amount) exte
```

Managing Credit

Jobs need credit to be able to pay keepers, this credit can either be paid for directly, or by being a liquidity provider in the system. If you pay directly, this is a direct expense, if you are a liquidity provider, you get all your liquidity back after you are done being a provider.

Add credit to a job via Liquidity

Step 1 is to provide LP tokens as credit. You receive all your LP tokens back when you no longer need to provide credit for a contract.

```
/**
 * @notice Allows liquidity providers to submit jobs
 * @param liquidity the liquidity being added
 * @param job the job to assign credit to
 * @param amount the amount of liquidity tokens to use
 */
function addLiquidityToJob(address liquidity, address job, uint amount) exte
```

Wait `LIQUIDITYBOND` (default 3 days) days.

```
/**
 * @notice Applies the credit provided in addLiquidityToJob to the job
 * @param provider the liquidity provider
 * @param liquidity the pair being added as liquidity
 * @param job the job that is receiving the credit
 */
function applyCreditToJob(address provider, address liquidity, address job)
```

Remove liquidity from a job

```
/**
 * @notice Unbond liquidity for a job
 * @param liquidity the pair being unbound
 * @param job the job being unbound from
 * @param amount the amount of liquidity being removed
 */
function unbondLiquidityFromJob(address liquidity, address job, uint amount)
```

Wait `UNBOND` (default 14 days) days.

```
/**
 * @notice Allows liquidity providers to remove liquidity
 * @param liquidity the pair being unbound
 * @param job the job being unbound from
 */
function removeLiquidityFromJob(address liquidity, address job) external
```

Adding credit directly (non ETH)

```
/**
 * @notice Add credit to a job to be paid out for work
 * @param credit the credit being assigned to the job
 * @param job the job being credited
 * @param amount the amount of credit being added to the job
 */
```

```
function addCredit(address credit, address job, uint amount) external
```

Adding credit directly (ETH)

```
/**
 * @notice Add ETH credit to a job to be paid out for work
 * @param job the job being credited
 */
function addCreditETH(address job) external payable
```

Selecting Keepers

Dependent on your requirements you might allow any keepers, or you want to limit specific keepers, you can filter keepers based on `age` , `bond` , `total earned funds` , or even arbitrary values such as additional bonded tokens.

No access control

Accept all keepers in the system.

```
/**
 * @notice confirms if the current keeper is registered, can be used for ger
 * @param keeper the keeper being investigated
 * @return true/false if the address is a keeper
 */
function isKeeper(address keeper) external returns (bool)
```

Filtered access control

Filter keepers based on bonded amount, earned funds, and age in system.

```
/**
 * @notice confirms if the current keeper is registered and has a minimum bo
 * @param keeper the keeper being investigated
 * @param minBond the minimum requirement for the asset provided in bond
 * @param earned the total funds earned in the keepers lifetime
 * @param age the age of the keeper in the system
 * @return true/false if the address is a keeper and has more than the bond
 */
function isMinKeeper(address keeper, uint minBond, uint earned, uint age) e
```

Additionally you can filter keepers on additional bonds, for example a keeper might need to have `SNX` to be able to participate in the Synthetix ecosystem.

```
/**
 * @notice confirms if the current keeper is registered and has a minimum bo
 * @param keeper the keeper being investigated
 * @param bond the bound asset being evaluated
 * @param minBond the minimum requirement for the asset provided in bond
 * @param earned the total funds earned in the keepers lifetime
 * @param age the age of the keeper in the system
 * @return true/false if the address is a keeper and has more than the bond
 */
function isBondedKeeper(address keeper, address bond, uint minBond, uint ea
```

Paying Keepers

There are three primary payment mechanisms and these are based on the credit provided;

- Pay via liquidity provided tokens (based on `addLiquidityToJob`)
- Pay in direct ETH (based on `addCreditETH`)
- Pay in direct token (based on `addCredit`)

Auto Pay

If you don't want to worry about calculating payment, you can simply let the system calculate the payment itself;

```
/**
 * @notice Implemented by jobs to show that a keeper performed work
 * @param keeper address of the keeper that performed the work
 */
function worked(address keeper) external
```

Pay with KPR

The maximum amount that can be paid out per call is `(gasUsed * fastGasPrice) * 1.1`

```
/**
 * @notice Implemented by jobs to show that a keeper performed work
 * @param keeper address of the keeper that performed the work
 * @param amount the reward that should be allocated
 */
function workReceipt(address keeper, uint amount) external
```

Pay with token

There is no limit on how many tokens can be paid out via this mechanism

```
/**
 * @notice Implemented by jobs to show that a keeper performed work
 * @param credit the asset being awarded to the keeper
 * @param keeper address of the keeper that performed the work
 * @param amount the reward that should be allocated
 */
function receipt(address credit, address keeper, uint amount) external
```

Pay with ETH

There is no limit on how many tokens can be paid out via this mechanism

```
/**
 * @notice Implemented by jobs to show that a keeper performend work
 * @param keeper address of the keeper that performed the work
 * @param amount the amount of ETH sent to the keeper
 */
```

```
function receiptETH(address keeper, uint amount) external
```