Matryx Source Documentation

Release 0.2

Max Howard, Sam Hessenauer

May 09, 2019

API Documentation

This documentation details the features of the Matryx Platform Smart Contract system. You can check out the Matryx Platform source code here.

CHAPTER 1

Entering Matryx

Matryx is a decentralized application for solving difficult scientific problems. The Matryx's commit system allows you to timestamp and value your work on the Ethereum blockchain, providing you with immutable proof of ownership over your content. The Matryx bounty system enables users to place bounties on scientific problems and to award creative solutions, allowing people to build up chains of work and be rewarded for their contributions.

Welcome!

1.1 Approving Platform Transactions

To interact with the platform, you first need to tell `MatryxToken` that you approve of `MatryxPlatform`, and specify a number of `tokens` you want to approve. This is done by calling

```
token.approve(MatryxPlatform.address, tokens)
```

Great! Now that you know how to approve platform transactions, you can create tournaments, make submissions, and interact with other platform users.

CHAPTER 2

The Matryx Commit System

The commit system is a collaborative content and IP tracking tool that allows value to be assigned and distributed across sequences of innovation. The commit system is comprised of individual commits, which are units of work containing an IPFS content hash and a self-determined value that indicates how much each commit is worth. Additionally, commits can contain links to previous commits, allowing users to collaborate with each other and build up entire chains of work.

2.1 Creating your first Commit

In order to create a new commit, you will first need to claim the hash that you are going to use for it on the system. This is done by calling

```
commit.claimCommit(commitHash)
```

Where `commitHash` is a hash that's comprised of your public Ethereum address, an IPFS hash of the content of your commit, and some salt used to encrypt the commit's contents.

The reason you have to claim a commit hash before actually creating the commit is to prevent frontrunning attacks. If a malicious actor were monitoring the system when you try to create the commit, they could potentially steal and take ownership of your commit content if they send the same transaction as you and their transaction gets mined first. The `claimCommit` function exists to prevent this kind of attack. Once you have claimed a commit hash, the system knows that this particular hash belongs to you. So, when you send the transaction to create the commit and actually reveal the commit's content, no malicious actor can frontrun the transaction and steal your work.

That said, you can create your commit by calling

```
commit.createCommit(parentHash, isFork, salt, content, value)
```

`Salt` and `content` are the same hashes that you used to claim your commit. `Value` is a user-specified amount that indicates how much you think your commit is worth. The first two fields, `parentHash` and `isFork` are used whenever you want to work off of someone else's commit. We will now explain how this collaboration process takes place.

2.2 Groups and Collaboration

Commit groups are used to allow people to team up and work freely off of each other's commits. In order to make a commit off of a parent commit, you must either be in the parent commit's group, or fork off of the commit and start working with a new group.

To add a new user (or multiple users) to your group, you can call these two functions, respectively:

```
commit.addGroupMember(commitHash, user)
commit.addGroupMembers(commitHash, users)
```

Where `commitHash` is the hash of the commit in the chain that the new user wants to contribute to.

Warning: Group members cannot be removed from the group after you have added them. You should only add to your group users that you trust.

Any group member can add you to their group. Once you become a part of the group, you will be able to make new commits along the same commit chain without having to pay for each of the commits made by your group members. To create a commit off of another commit by someone in your group, you call the `createCommit` function and pass the hash of their commit as your `parentHash`. Since you are working in the same group, `isFork` should be `false`.

2.3 Forking Commits

The process of forking off of a commit is similar to that of creating a new commit. First, you need to claim a commit hash using the same claim function:

```
commit.claimCommit(commitHash)
```

Then, you create your commit:

```
commit.createCommit(parentHash, isFork, salt, content, value)
```

Now, `parentHash` is the hash of the commit you want to fork, and `isFork` should be `true`.

When you fork off of a commit, you are buying the right to use the entire previous chain of work for your own line of work that you are initiating at this point. The new commit that you create will represent a new starting point, and you will be the only user in the group (until you decide to add more people). After the fork, you will be able to work off of your commit freely, as you now have the right to use the previous line of work before the fork and the work that you and your group members add from that point on.

To fork off of a commit, you have to compensate the owners of commits in the chain for their contributions. The cost of a fork is the sum of the value of the commits in the line of work up until that point. Therefore, before you fork, you should make sure that you have approved at least that many MTX tokens.

Note: Anyone can fork a commit. You do not have to be in a commit's group in order to fork the commit.

CHAPTER 3

The Matryx Bounty System

The Matryx Bounty system enables and incentivizes decentralized scientific collaboration in the form of tournaments and submissions, where all users participating in a tournament receive credit for their contributions, and the tournament bounty is rightfully distributed among the chosen winners.

3.1 Tournament and Round States

Each tournament in Matryx is subdivided into rounds, which are indexed incrementally. In each round, tournament participants make submissions to the tournament, and at the end of each round one or multiple submissions are selected by the tournament owner to be the winners of the round. The round winners then receive the allocated round reward.

A tournament can be in one of five possible states: `NotYetOpen`, `OnHold`, `Open`, `Closed`, or `Abandoned`.

If a tournament is `NotYetOpen`, then its first round has not started yet. If it is `OnHold`, that means that the tournament has already started but the next upcoming round has yet to begin. `Open` tournaments are the ones that are currently active: submissions are being made or reviewed. `Closed` tournaments are no longer active; the tournament owner has decided to end the tournament, and all of the tournament's bounty has been distributed among the various rounds' winners. Lastly, a tournament becomes `Abandoned` if a round ends without receiving any submissions, or if the tournament owner fails to select winners before the end of the round's review period.

You can check the state of a tournament at any time by calling

```
tournament.getState()
```

A round can be in one of seven possible states: `NotYetOpen`, `Unfunded`, `Open`, `InReview`, `HasWinners`, `Closed`, or `Abandoned`.

Rounds that have not started yet are `NotYetOpen`. A round is `Unfunded` if it has already started but the tournament owner has not added any MTX to its bounty yet. `Open` rounds are currently active; you can enter the round and make new submissions. If a round is `InReview`, you can no longer make any more submissions (you're going to have to wait until the next round!). This is the time when the tournament owner reviews all the submissions made to the round and selects the winners. A round is `Closed` after the `InReview` period ends. Lastly, a round becomes `Abandoned` if it reaches the end of its `Open` state without receiving any submissions.

You can check the state of any round at any time by calling

```
tournament.getRoundState(roundIndex)
```

3.2 Entering and Exiting Tournaments

To enter a tournament that you'd like to participate in, you can make the following call:

```
tournament.enter()
```

Whenever you enter a tournament, you will have to pay the tournament entry fee designated by the tournament creator, so you will need to first approve at least that many MTX tokens. To check what a tournament's entry fee is before entering, you can call

```
tournament.getDetails()
```

You can choose to exit an ongoing tournament at any time with the following call:

```
tournament.exit()
```

When you exit the tournament, the entry fee that you paid when you first entered will be returned to you automatically.

Note: If you later decide to enter the tournament again, you will have to pay the tournament entry fee again before making any submissions.

If a tournament you are currently participating in happens to become `Abandoned`, you can collect your share of the remaining tournament bounty, as well as your original entry fee, with the following call:

```
tournament.withdrawFromAbandoned()
```

3.3 Making your first Submission

To create a submission, you must first enter the tournament that you want to participate in by calling

```
tournament.enter()
```

You can then create a submission in two ways: You can create a new commit and submit it to the tournament, or you can submit one of your previous commits.

To create a new commit, you first need to claim the commit hash:

```
commit.claimCommit(commitHash)
```

Then, you can create the commit and submit it to a tournament using the following function:

```
commit.createSubmission(tournament, content, parentHash, isFork, salt, commitContent,␣
↪value)
```

Where `tournament` is the address of the tournament, `content` is the Submission content, and `parentHash`, `isFork`, `salt`, `commitContent`, and `value` are the data that corresponds to the commit you are creating.

Alternatively, if you want to submit a commit that already exists, you can simply call:

```
tournament.createSubmission(content, commitHash)
```

Where `content` is the content of your submission and `commitHash` is the hash of your commit.

3.4 Checking Commit Balances

If a commit receives some amount of MTX, the appropriate amount will be allocated to the commit on the Matryx platform. To check the current allocated balance of any commit, you can call

```
commit.getCommitBalance(commitHash)
```

3.5 Collecting your Reward

When a commit in your line of work receives a reward from winning a tournament, you can withdraw your share of the reward by calling:

```
commit.withdrawAvailableReward(commitHash)
```

To check the reward that any user is entitled to for any particular commit, you can call

```
commit.getAvailableRewardForUser(userAddress)
```

After you withdraw your reward, your available reward for that particular commit goes down to 0.

The share of the reward that is allocated to each commit owner in a commit chain is proportional to the total value of the commits that they created. Therefore, when a commit wins a tournament, everyone who contributed a piece of work used by the winning commit receives compensation for their contributions.

3.6 Creating your own Tournament

To create a tournament, you can call the `createTournament` function on the platform as follows:

```
platform.createTournament(tournamentDetails, roundDetails)
```

Where `tournamentDetails` and `roundDetails` are:

```
struct TournamentDetails
{
    string content;
    uint256 bounty;
    uint256 entryFee;
}

struct RoundDetails
{
    uint256 start;
    uint256 duration;
    uint256 review;
    uint256 bounty;
}
```

These structs contain information about the tournament that you are about to create and the first round that will kick off when the tournament begins. You can add more funds to the tournament bounty at any point, but you cannot remove funds from it after you make the `createTournament` call, so choose your initial bounty wisely!

Similarly, you cannot remove funds from the share of the tournament bounty you assign to the first round, and you won't be able to edit the round details after the round has started. Be sure to enter a reasonable amount of time (in seconds) for the round's `start` time and `duration`, as well as its `review` period. You'll need some time to look over the submissions and choose your round winners before the review period ends!

Note: The tournament and round bounty will be visible to any users looking to enter your tournament, as well as the tournament and round details.

3.7 Updating Tournament Details

To edit the data of your tournament, you can call the `updateDetails` function as follows:

```
tournament.updateDetails(tournamentDetails)
```

Where `tournamentDetails` is the same data struct used to create the tournament originally. The `bounty` field, however, will not change when you try to modify the tournament's data.

3.8 Adding to a Tournament Bounty

Suppose you wanted to add MTX to a tournament's bounty. You can call the `addToBounty` function as follows:

```
tournament.addToBounty(amount)
```

This function transfers MTX to the specified tournament. To allocate these new funds to the current round, you can call the `transferToRound` function:

```
tournament.transferToRound(amount)
```

The added MTX will now also be distributed to this round's winners when it is time to reward their submissions.

Warning: Remember that you cannot remove funds from a tournament's bounty after you've added them or remove funds from a round after it has already started.

3.9 Choosing Tournament Winners

To get all the submissions made to a round, you can call

```
tournament.getRoundInfo(roundIndex)
```

Round info contains all the hashes of all the submissions made to the round. To view the contents of each submission, you can call

```
platform.getSubmission(submissionHash)
```

To choose your round winners, you can call `selectWinners` on the tournament as follows:

```
tournament.selectWinners(winnersData, roundDetails)
```

Where `winnersData` is:

```
struct WinnersData
{
    bytes32[] submissions;
    uint256[] distribution;
    uint256 action;
}
```

Here, `action` represents an enumerated value from the following enum:

```
enum SelectWinnerAction { DoNothing, StartNextRound, CloseTournament }
```

and `RoundDetails` are:

```
struct RoundDetails
{
    uint256 start;
    uint256 duration;
    uint256 review;
    uint256 bounty;
}
```

In `winnersData`, you can specify which submissions get rewarded and how much MTX is assigned to each one. The first parameter contains all the winning submissions' hashes, and the second contains the reward each one will get, respectively, expressed as a percentage or a proportion of the total round bounty.

When selecting round winners, you have three options for how to proceed with the tournament: you can choose to wait until the end of the review period for the next round to start, to start the next round immediately after selecting the winners, or to close the tournament. The action you choose (0, 1 or 2, representing `SelectWinnerAction.DoNothing`, `SelectWinnerAction.StartNextRound` and `SelectWinnerAction.CloseTournament`, respectively) is passed as the third parameter of `winnersData` and indicates how you would like to proceed.

If you choose to wait until the end of the review period (`DoNothing`), the next round will automatically be created as an identical copy of the last round, and it will begin once the current review period ends. If you choose to start the next round immediately when you select the winners (`StartNextRound`), the next round will be initialized with the round data that you provide and will begin immediately. If you choose to close the Tournament (`CloseTournament`), the remaining bounty unallocated to any round will be allocated to the current round and used to award `winnersData.submissions`, and the Tournament will end.

Warning: Once you close the tournament, you can't open it up again. Any remaining funds that might still be in the tournament's balance will be evenly distributed among the last round's winners when you decide to close the tournament.

Warning: If the round's review period ends and you still have not chosen any winners, the tournament will be considered Abandoned, and any remaining funds in the tournament's balance will be uniformly allocated to all of the round's participants for them to withdraw.

CHAPTER 4

Matryx Marketplace

Stay tuned!

CHAPTER 5

Frequently Asked Questions

Stay tuned!

CHAPTER 6

Versions

Stay tuned!