Urbit Constitution JS Library

Code Review

December 16th 2018

Version 2.0.0

Prepared by

Bloctrax

Table of Contents

Introduction

This document includes the results of the code review for Urbit's Constitution JS library, as found in the section titled 'Source Code'. The code review was performed by the Bloctrax team from November 28th 2018 to December 16th 2018.

The purpose of this engagement is to review Urbit's Constitution JS library source code and provide feedback on the design, architecture, and quality of the source code.

Overall Assessment

Our overall assessment of the Urbit Constitution JS library is that this version is substantially better than the version we previously reviewed and appears not to suffer from the extremely poor code quality that the previous version did.

Despite that, this version still has a number of concerning outstanding issues. Most importantly, the urbit-azimuth library has undergone significant revisions since the version that the Constitution JS library is pinned to. These changes range from relatively simple things to be fixed, such as renaming Constitution to Ecliptic and Ships to Azimuth, to more concerning issues, such as missing entire fields and methods that the Constitution JS library relies upon in the latest version of the contracts.

These issues are further compounded by the overall low code coverage of the library. Our assessment shows that the library only has ~61% statement coverage (and lower branch coverage). This is unreasonably low, especially for a project written in JavaScript where small syntactic issues (and even typos) will not manifest themselves until runtime.

The importance of improving the test coverage of this project is further emphasized by the fact that we caught several different issues where code will either throw or not work at runtime. These issues would have been flagged and identified by even the most trivial missing unit tests. We believe that investing significant time to improve the quality of the

project's test suite is likely to uncover further issues that we did not catch during the course of our review.

Specification

Our understanding of the specification was based on the following sources:

- Our understanding of the desired behavior based on our previous review of the Urbit Constitution Solidity code.
- Discussions with the Urbit team.

Source Code

The following source code was reviewed:

Repository	Commit
constitution-js	819102e717b7c0166a6967a353c02e03cb70965f

Note: This document contains a review only of the JavaScript code contained in the code repository listed above. The review does not include any of the dependencies being used.

Severity Level Reference

Level	Description
High	The issue poses existential risk to the project, and the issue identified could lead to massive financial or reputational repercussions.
Medium	The potential risk is large, but there is some ambiguity surrounding whether or not the issue would practically manifest.
Low	The risk is small, unlikely, or not relevant to the project in a meaningful way.
Code Quality	The issue identified does not pose any obvious risk, but fixing it would improve overall code quality, conform to recommended best practices, and perhaps lead to fewer development issues in the future.

Issues Descriptions and Recommendations

Mismatch between latest Urbit constitution contracts, and Constitution JS library

MEDIUM

The Constitution JS library is pinned to an older version of the Urbit Constitution contracts that does not match the current state / branding. For example, the code references the `Constitution` contract instead of `Ecliptic` and `Ships` instead of `Azimuth`.

Mismatch between external and internal contract implementations

MEDIUM

The public `conditonalSR.js` file tries to expose a function called `getStartTime` from the `internal/conditionalSR.js` file, but that function does not exist.

Unusable code

MEDIUM

The `initContracts` and `initContractsPartial` functions in `contracts.js` do not initialize the `conditionalSR` and `linearSR` contracts so those won't be usable even though they both have internal and public implementations (`conditionalSR.js`, `linearSR.js` and `internal/conditionalSR.js, internal/linearSR.js` respectively).

In addition, there is an `internal/pool.js` implementation, but no external implementation so that contract will not be usable as well. It's also not initialized by the `initContracts` and `initContractsPartial` functions either.

Broken and unrunnable code

MEDIUM

There are several portions of the Urbit constitution JS library that are broken and/or will not run as expected.

1. `internal/ships.js` has a function called `getSpawnProxy` where the `contracts` argument is misspelled as `contarct`. This will cause a runtime error.
2. `ships.js` and `internal/ships.js` both expose a method called `getOwnedShipAtIndex`, but that method does not exist on any of the actual solidity contracts.
3. Both the internal implementations of `conditionalSR` and `linearSR` have a method called `getApprovedTransfer` that relies on a solidity method / field called `transfers`, but neither of the solidity contracts has that method. Any code dependent on these functions will be broken.
4. The `csrCanWithdraw` function is not tested at all and is likely broken. It does a comparison in the form of `com.withdrawn >= lim`, but `com.withdrawn` is an array, not a number. Similarly, it has a conditional check in the form of `if (com.forfeit && rem.length <= com.forfeited)`, but there is no forfeit field and the forfeited field is an array not a number.
5. The internal implementation of `conditionalSR` calls the `withdrawLimit` method with one argument (`address`) but the solidity contract accepts two arguments (`address` and `batch`).
6. The internal implementation of `conditionalSR` calls the `withdraw` method with zero arguments, but the solidity contract accepts two arguments (`batch` and `address`).
7. The internal implementation of `conditionalSR` calls the `withdrawTo` method, but that function does not exist on the solidity contract.
8. The internal implementation of `constitution` calls the `transferShip` method on the `Ecliptic` contract, but that function does not exist on the solidity contract.

9. The internal implementation of `constitution` calls the `startConstitutionPoll` method on the `Ecliptic` contract, but that function does not exist on the solidity contract.
10. The internal implementation of `constitution` calls the `castConstitutionVote` method on the `Ecliptic` contract, but that function does not exist on the solidity contract.
11. The internal implementation of `constitution` calls the `updateConstitutionPoll` method on the `Ecliptic` contract, but that function does not exist on the solidity contract.
12. The internal implementation of `linearSR` calls the `transfers` method on the `LinearStarRelease` contract, but that function does not exist on the solidity contract.
13. The internal implementation of `linearSR` calls the `withdrawTo` method on the `LinearStarRelease` contract, but that function does not exist on the solidity contract.
14. The internal implementation of `polls` calls the `constitutionPolls` method on the `polls` contract, but that function does not exist on the solidity contract.
15. The internal implementation of `polls` calls the `constitutionHasAchievedMajority` method on the `polls` contract, but that function does not exist on the solidity contract.
16. The internal implementation of `polls` calls the `hasVotedOnConstitutionPoll` method on the `polls` contract, but that function does not exist on the solidity contract.
17. The internal `pool` contract does not seem to line up with any of the solidity contracts in the latest version of the Urbit Azimuth library.
18. The internal implementation of `ships` calls the `ships` function on the `ships` contract, but that function does not exist on the `Azimuth` solidity contract.
19. The internal implementation of `ships` calls the `getOwnedShipsByAddress` function on the `ships` contract, but that function does not exist on the `Azimuth` solidity contract.
20. The internal implementation of `ships` calls the `getOwnedShipCount` function on the `ships` contract, but that function does not exist on the `Azimuth` solidity contract.

21. The internal implementation of `ships` calls the `getOwnedShipAtIndex` function on the `ships` contract, but that function does not exist on the `Azimuth` solidity contract.
22. The internal implementation of `ships` calls the `hasBeenBooted` function on the `ships` contract, but that function does not exist on the `Azimuth` solidity contract.

Remaining FIXMEs in the code

LOW

There are two remaining FIXMEs in the code. One in the `canStartConstitutionPoll` function in `check.js`, and one in the `tests.js` file in a commented out test, which should also be addressed.

Incorrect documentation

LOW

Several portions of the documentation (either code comments, README, or otherwise) are incorrect.

1. The documentation in `polls.js` states that `hasVotedOnDocumentPoll`, the proposal argument, should be the address of the proposal, but actually it should be the hash of the proposal (`bytes32` in solidity).
2. The documentation in `ships.js` states that `isManagementProxy` accepts an `owner address` and a `manager address`, but actually the underlying Azimuth contract accepts `owner` as a `pointID` and `manager` as an `address`.
3. The documentation in `ships.js` states that `isVotingProxy` accepts an `owner address` and a `manager address`, but actually the underlying Azimuth contract accepts `owner` as a `pointID` and `manager` as an `address`.
4. The documentation in `ships.js` states that `getKeys` will return an object with the ships key configuration, which is true if the an object is passed as the `ship` parameter, but if `ship` is passed as a `pointID` then the function will return an `array` inside a `promise`.
5. The documentation in `ships.js` states that `getSpawned` will return a `bool` indicating whether the ship has been spawned, but the implementation of

`getSpawned` in the `Azimuth` contract returns an `array` of `points` created under the provided `point`.

6. The documentation states that all the functions in `ships.js` are supposed to return `promises`, regardless of whether they call out to the network or not, but many of the functions are non `async` functions that just return an unpromisified result in the case where they are passed a `ship` object as an argument.
7. All the documentation strings in the `conditonalSR.js` file state that the functions return concrete types when in fact they return concrete types wrapped in `promises`.

isParent function should handle zero galaxy

LOW

The `isParent` function in `check.js` should check if `ship >= 0` instead of `ship > 0` because 0 is a valid galaxy.

pollIsActive function should add additional check

LOW

The `pollIsActive` function in `check.js` should check that now is larger than or equal to the start of the poll.

Suggestion: additional assertions and guard statements

CODE QUALITY

Several of the functions in the Urbit Constitution JS library would benefit from additional assertions and guard statements.

1. The `initContracts` function should throw if any of the contract keys are missing.
2. The `pollIsActive` function in `check.js` should check that now is larger than or equal to the start of the poll.
3. The `canCreateGalaxy` function should check that `shipID` is a valid galaxy.
4. The `canSpawn` function should check that the `target` is a valid ship.

Code duplication

CODE QUALITY

The exact same `tx` function is repeated in several different files in the `internal` folder. It would be better to define this function one time and reuse it.

Appendix

Exhibit A - Disclaimer

The scope of this report and review is limited to a review of only the code presented by the Urbit team and only the source code Bloctrax notes as being within the scope of Bloctrax's review within this report. Specifically, this report does not constitute investment advice and is not intended to be relied upon as investment advice. The report is not an endorsement of this project or team, nor is it a guarantee as to the absolute quality or security of this project.

Bloctrax makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Bloctrax specifically disclaims all implied warranties or merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Bloctrax will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand against company by any other party. If no event will Bloctrax be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Bloctrax has been advised of the possibility of such damages.

Bloctrax assumes no responsibility for the use of software, whether created by Urbit, or any third party and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.