

CERTIK AUDIT REPORT
FOR TELLOR



Request Date: 2019-05-28
Revision Date: 2019-08-06
Platform Name: Ethereum



Contents

Disclaimer	1
About CertiK	2
Exective Summary	3
Vulnerability Classification	3
Testing Summary	4
Audit Score	4
Type of Issues	4
Vulnerability Details	5
Manual Review Notes	6
Static Analysis Results	13
Formal Verification Results	14
How to read	14
Source Code with CertiK Labels	89

Disclaimer

This Report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and Tellor(the “Company”), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the “Agreement”). This Report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This Report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK’s prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, has developed a proprietary Formal Verification technology to apply rigorous and complete mathematical reasoning against code. This process ensures algorithms, protocols, and business functionalities are secured and working as intended across all platforms.

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis, and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 6.2B in assets.

For more information: <https://certik.org/>

Executive Summary

This report has been prepared as the product of the Smart Contract Audit request by Tellor. This audit was conducted to discover issues and vulnerabilities in the source code of Tellor's Smart Contracts. Utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review, a comprehensive examination has been performed. The auditing process pays special attention to the following considerations.

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessment of the codebase for best practice and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line by line manual review of the entire codebase by industry experts.

Vulnerability Classification

For every issue found, CertiK categorizes them into 3 buckets based on its risk level:

Critical

The code implementation does not match the specification, or it could result in loss of funds for contract owner or users.

Medium

The code implementation does not match the specification at certain conditions, or it could affect the security standard by lost of access control.

Low

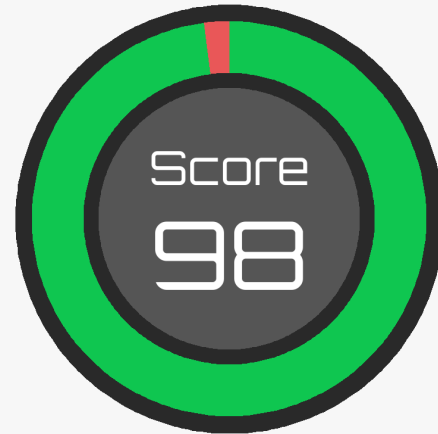
The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerabilities, but no concern found yet.

Testing Summary

PASS

CERTIK believes this
smart contract passes security
qualifications to be listed on
digital asset exchanges.

Aug 06, 2019



Type of Issues

CertiK smart label engine applied 100% covered formal verification labels on the source code, and scanned the code using our proprietary static analysis and formal verification engine to detect the follow type of issues.

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by minors to some degree.	0	SWC-116
Insecure Compiler Version	Using an fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	SWC-102 SWC-103
Insecure Randomness	Block attributes are insecure to generate random numbers, as they can be influenced by minors to some degree.	0	SWC-120

"tx.origin" for authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115
Delegatecall to Untrusted Callee	Calling into untrusted contracts is very dangerous, the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default. A malicious user is able to make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used as best practice.	0	SWC-111
Unused variables	Unused variables reduce code quality	0	

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Manual Review Notes

Tellor Manual Review

Review Details

Source Code SHA-256 Checksum

- **SafeMath.sol**
64253d59b1eb369ade333d1b7baa8c1f758955c6b1474a8407e974a9b4e72a8f
- **TellorDispute.sol**
705dff066bad4d1c0b072fe2950abaf56b40426cbac401c7f945c5459a6ca89
- **TellorGettersLibrary.sol**
7cd859cdd2a1635e6fe5aa3d79df41f3104531f4164a4d0aedf947458a18e9cb
- **TellorLibrary.sol**
8826af2e2d5e3e2e1394f06e0cdcaf13751dd98b58177cf49b38d0415b688f6a
- **TellorStake.sol**
61062be22a45e965b6dbfb4bbb5200d8ec12c71a6996fdd2354f7df4077015c7
- **TellorStorage.sol**
75e5132feee6116adf2ec2253d7d050e4521a8e0d9c436060b4a0431fb28e386
- **TellorTransfer.sol**
da43a8510b6b184c283bb9ed904909ab937e09b9761f0e4c65d04eeb93636f94
- **Utilities.sol**
d287c00e4d3b8e60319a7cdc68a66fba253d2024aab9bc9bd609703cf5ba795c
- **Migrations.sol**
8f6c5cd63921e1fda822ae49d8b46d6d2ddea8a40e07f531e6f2bfcd88d2a102
- **Tellor.sol**
6b2fedfe4b42e9e3a545e28244c7417fbd65d0c419936eadcbe5a831135c9ce3
- **TellorGetters.sol**
8e34e605a9d3db9c9a95e7004bb30de557c0d68f6b1505dcfc8b06287f181152
- **TellorMaster.sol**
d93056c8b8c544bd1aac02f2480f81f13f71ada7e52d6d963dd3ebb133135b64
- **UtilitiesTests.sol**
2c500ee0817614379be550b431a15478ea2796fac91b9e8119d4314994bd7bc9
- **Optimistic.sol**
21b2aba8e451f01c79c4c9a325a0ecf4ce5bcb3ae17cdf8c0a911ac3f59afa71
- **Reader.sol**
d9ccfc2b09e20103f88c227add8b28e54b4193bcb978943b320dfe67a7dfab92

- **UserContract.sol**

1f0ae6d9e5e9e4071c0508cfe75da0c83f750e2129ca04dc21edcffe7a863d49

- **UsingTellor.sol**

67399c64448241e5153e62ad69895aac8d4e7d9a6ff01c0b164c42df0eef3c3a

Summary

CertiK was chosen by Tellor to audit the design and implementation of its soon to be released oracle and governance smart contract. To ensure comprehensive protection, the source code has been analyzed by the proprietary CertiK formal verification engine and manually reviewed by our smart contract experts and engineers. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with the best practices in the space.

The Tellor team has demonstrated their professional and knowledgeable understanding of the project, by having 1) a production ready repository with high-quality source code; 2) unit tests covering the majority of its business scenarios; 3) accessible, clean, and accurate readme documents for intentions, functionalities, and responsibilities of the smart contracts.

The Tellor team responded to feedbacks from CertiK team timely and demonstrated very high productivity, those factors make the review process very smooth.

Overall the CertiK team found the smart contracts to follow good security practices. With the final update of source code and delivery of the audit report, we conclude that the contracts are structurally sound and not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend to seek multiple opinions, keep improving the codebase, and more test coverage and sandbox deployments before the mainnet release.

Documentation

CertiK used the following sources of truth about how Tellor smart contracts should work:

1. [Tellor Website](#)
2. [Tellor Whitepaper](#)
3. [Tellor Contracts Source Code](#)
4. [Tellor Contracts Test Cases](#)

All listed sources act as a specification. If we discovered inconsistencies within the actual code behavior, we consulted with the Tellor team for further discussion and confirmation.

Discussion

Items in this section are low impact to the overall aspects of the smart contracts, thus will let client to decide whether to have those reflected in the final deployed version of source codes.

- **INFO** For data structure design, what were the design considerations for introducing `uintVars`, `addressVars`, and `apiUintVars` mappings instead of creating these variables directly as `struct` members? A more flattened data structure design might improve readability of the contract code.
 - (Tellor) Mainly for upgradeability (gives us an unlimited number of future variables), but it also cuts down deployment size limits (bytecode too large). We decided to keep this structure since it is well documented and allows us the upgradability, we need to the `TellorStorage` since the structure we adopted makes `TellorStorage` static/unupgradable.
- **INFO** There are too many `keccak256` calls, which results in low code readability and potential more consumption of gas.
 - (Tellor) We decided to keep for upgradeability.
- **INFO** In the whitepaper `RequestQ` size is stated to be 50, nevertheless in the code `Request.requestQ` is defined as `uint[51]`. Is there any reason to the increase of the size by 1 (e.g. prefer index to be starting from 1 instead of 0) ?
 - (Tellor) We wanted the index to start at 1 to avoid confusion as the mapping that points to their position in the array has 0 as the null value.

Review Notes

Review: 2rd Round (07/2019)

- **MAJOR** Potential overflow issue in `TellorDispute.vote` method:
 - Line 120: Potential overflow if `disp.tally + voteWeight` is too large.
 - Line 123: Potential underflow if `disp.tally - voteWeight` is too small.
 - * (Tellor) Fixed. Introduced `SafeMath` for int type and applied it to the above calculations.
- **MINOR** We could use a `pending/claim` ownership transfer pattern for `TellorLibrary.transferOwnership`. Basically what we need to do is adding a `pendingOwner` field, then first current owner would propose a `pendingOwner` by calling `proposeOwnership` function, followed by `pendingOwner` to call `claimOwnership` function to finish the ownership transfer; this help eliminate cases like a small typo of the new owner address cause some severe impact

- (Tellor) Fixed. Applied the pending/claim pattern.
- **DISCUSSION** Can anyone propose a fork? Who should be responsible for creating the new tellor contract?
 - (Tellor) Yes. But they need 20% quorum to be able to execute upon tally. Anyone can propose an address for forking.
- **DISCUSSION** Should `TellorLibrary.addTip` and `TellorLibrary.requestData` require tip to be larger than 0; especially for `addTip` it doesn't make much sense to add zero tip.
 - (Tellor) Since we initiate the contract with zero token supply we need to be able to call the 1st data request with zero tokens.
- **INFO** `Utilities.getMin()`, `Utilities.getMax()`: There is a warning on the official solidity documentation about the usage of the first slot of an array: "Statically-sized memory arrays do not have a length field, but it might be added later to allow better convertibility between statically- and dynamically-sized arrays, so please do not rely on that." Given this consideration, we would now suggest switching `getMin()`, `getMax()` functions back to the non-assembly version for long-term compatibility.
 - (Tellor) Fixed. Switched Utilities functions back to non-assembly version.
- **INFO** For complex function such as `TellorLibrary.submitMiningSolution` we can declare local variables instead of calling statements such as `self.uintVars[keccak256("currentRequestId")]` to both improve readability and save gas cost.
 - (Tellor) We have added two local variables: `_topId` (for `self.uintVars[keccak256("currentRequestId")]`) and `_timeofLastNewValue` (for `now - (now \% 1 minutes)`).
- **INFO** We didn't see any unit test cases for the assembly code in `Utilities.sol`. Would you like to to cover this part with some edge test cases?
 - (Tellor) We have added tests for the `getMin` and `getMax` in the `furtherTests` tests.

Review: 1st Round (06/2019)

According to our understanding that `TellorLibrary` and `TellorGettersLibrary` are designed to share the exact same `TellorStorageStruct`. While `TellorLibrary` includes a set of functions that can be upgradable with Tellor contract, `TellorGettersLibrary` includes a set of functions that cannot be upgraded.

- **MINOR** `TellorGettersLibrary.tellorPostConstructor()`: In the for loop initializing `requestQ` item values to 0, the 49 is used as the size of the array which isn't the same as the defined size of `requestQ`; also the for loop is not actually needed since the initial value for `requestQ` items are already set to 0 by solidity.

- (Teller) We removed it. We still left the array at 51 since we still want the requests to begin at 1.
- **MINOR** `TellerGettersLibrary.getTimestampByRequestIDandIndex()`: This will cause failing asserting when the `_index` is out of bound. Recommend to check by `require()` first.
 - (Teller) Fixed.
- **DISCUSSION** Are there any specific reasons why the `TellerStorageStruct` methods are not upgradable?
 - (Teller) We decided to leave the `TellerGettersLibrary` in the `TellerMaster.sol` and un-upgradable for two reasons, 1) `TellerStorage` is not upgradable and `TellerGettersLibrary/TellerGetters` provides access to all of the variables in the `TellerStorage` and 2) it is much straight forward to access these without having to use a delegate call.
- **DISCUSSION** What is the purpose of introducing `onDeckRequest` in `TellerLibrary.submitMiningSolution`, instead of just promoting the request with the highest tips from `requestQ` directly?
 - (Teller) We used the `max` function to find the request with the highest tip/-payout and return that instead of using variables. This is removed in the latest version to simplify the contract.
- **INFO** Since the `TellerStorageStruct` is a large struct that contains dozens of variables, this makes readability a bit challenging. Is it possible to consider organizing data in a “domain-driven” way by splitting related data and logic into smaller business domain based libraries, such as `Dispute/Request/Mining/Transfer/Management`, etc, and eventually wiring them together in the `Teller/TellerMaster` contract?
 - (Teller) Thanks for helping us split into the different libraries. We’ve adopted the structure you suggested. The main difference is that we call `TellerStake.Init()` function via the `TellerMaster` to initialize the first 6 miners.
- **INFO** Recommend to use “concerns” pattern to address the gas consumption too high issue. By changing visibility for different library methods and reorganizing them into separate library files, we’ll be able to lower down the total gas cost to a ideal threshold.
 - (Teller) Reorganized in the above method and gas too high issue was resolved. Thanks for helping us split into the different libraries.
- **INFO** In the whitepaper `RequestQ` size is stated to be 50, nevertheless in the code `Request.requestQ` is defined as `uint[51]`. Is there any reason to the increase of the size by 1 (e.g. prefer index to be starting from 1 instead of 0) ?
 - (Teller) We wanted the index to start at 1 to avoid confusion as the mapping that points to their position in the array has 0 as the null value.

- **INFO** For data structure design, what were the design considerations for introducing `uintVars`, `addressVars`, and `apiUintVars` mappings instead of creating these variables directly as `struct` members? A more flattened data structure design might improve readability of the contract code.
 - (Tellor) Mainly for upgradeability (gives us an unlimited number of future variables), but it also cuts down deployment size limits (bytecode too large). We decided to keep this structure since it is well documented and allows us the upgradability, we need to the `TellorStorage` since the structure we adopted makes `TellorStorage` static/unupgradable.
- **INFO** There are too many `keccak256` calls, which results in low code readability and potential more consumption of gas.
 - (Tellor) We decided to keep for upgradeability.
- **INFO** Recommend to convert `Utilities.getMin` and `Utilities.getMax` to assembly version to save gas consumption.
 - (Tellor) We have converted these two functions to assembly version.
- **INFO** Recommend to create modifiers to improve readability for miner status, etc.
 - (Tellor) We were really trying to reduce bytecode size at deployment (and gas costs for users), but we agree that it would make it more readable.
- **INFO** `TellorLibrary.tellorPostConstructor()` is not called explicitly, should we include it in `Tellor.constructor`?
 - (Tellor) Fixed. This function is now `TellorStake.Init()` and it is called by the `TellorMaster`.
- **INFO** Is it possible to refactor the `submitMiningSolution` method into smaller internal functions each handling a single responsibility?
 - (Tellor) With the reduction in deployment size gained through your recommendation we implemented a new function: `new block` and made the `submitMiningSolution` smaller. The contract with the largest deployment size is 6M.
- **INFO** `TellorLibrary`: Recommend to remove unused import library `Utilities`
 - (Tellor) Fixed.
- **INFO** `TellorLibrary.approve()`: Recommend to check whether the `_user` address is a non zero address.
 - (Tellor) Done.

- **INFO** `TellorMaster`: Recommend to remove unused import library `TellorLibrary` and `TellorGettersLibrary`.
 - (Tellor) Done.

Best Practice

Solidity Protocol

- ✓ Use stable solidity version
- ✓ Handle possible errors properly when making external calls
- ✓ Provide error message along with `require()`
- ✓ Use modifiers properly
- ✓ Use events to monitor contract activities
- ✓ Refer and use libraries properly
- ✓ No compiler warnings

Privilege Control

- ✓ Restrict access to sensitive functions

Documentation

- ✓ Provide project readme and execution guidance
- ✓ Provide inline comment for function intention
- ✓ Provide instruction to initialize and execute the test files

Testing

- ✓ Provide test scripts and coverage for potential scenarios

With the final update of source code and delivery of the audit report, CertiK is able to conclude that the Tellor contracts are not vulnerable to any classically known anti-patterns or security issues.

While this CertiK review is a strong and positive indication, the audit report itself is not necessarily a guarantee of correctness or trustworthiness. CertiK always recommends seeking multiple opinions, test coverage, sandbox deployments before any mainnet release.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File Tellor.sol

```
1 pragma solidity ^0.5.0;
```

 Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File TellorGetters.sol

```
1 pragma solidity ^0.5.0;
```

 Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File TellorTransfer.sol

```
1 pragma solidity ^0.5.0;
```

 Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File SafeMath.sol

```
1 pragma solidity ^0.5.0;
```

 Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File TellorGettersLibrary.sol

```
1 pragma solidity ^0.5.0;
```



 Only these compiler versions are safe to compile your code: 0.5.9

Formal Verification Results

How to read

Detail for Request 1

transferFrom to same address


Verification date	 20, Oct 2018
Verification timespan	 395.38 ms

CERTIK label location	Line 30-34 in File howtoread.sol
-----------------------	----------------------------------

CERTIK label	30	/*@CTK FAIL "transferFrom to same address"
	31	@tag assume_completion
	32	@pre from == to
	33	@post __post.allowed[from][msg.sender] ==
	34	*/

Raw code location	Line 35-41 in File howtoread.sol
-------------------	----------------------------------

Raw code	35	function transferFrom(address from, address to
) {
	36	balances[from] = balances[from].sub(tokens
	37	allowed[from][msg.sender] = allowed[from][
	38	balances[to] = balances[to].add(tokens);
	39	emit Transfer(from, to, tokens);
	40	return true;
	41	}

Counterexample	 This code violates the specification	
Initial environment	1	Counter Example:
	2	Before Execution:
	3	Input = {
	4	from = 0x0
	5	to = 0x0
	6	tokens = 0x6c
	7	}
	8	This = 0
	52	}
	53	balance: 0x0
	54	}
	55	}
Post environment	57	After Execution:
	58	Input = {
	59	from = 0x0
	60	to = 0x0
	61	tokens = 0x6c

Formal Verification Request 1

approve correctness

📅 06, Aug 2019

🕒 46.78 ms

Line 164-171 in File Tellor.sol

```
164 /*@CTK "approve correctness"
165     @tag assume_completion
166     @post _spender != 0x0
167     @post __post.tellor.allowed[msg.sender][_spender] == _amount
168     @post !(__has_overflow)
169     @post !(__has_buf_overflow)
170     @post !(__has_assertion_failure)
171 */
```

Line 172-174 in File Tellor.sol

```
172 function approve(address _spender, uint _amount) external returns (bool) {
173     return tellor.approve(_spender,_amount);
174 }
```

✅ The code meets the specification.

Formal Verification Request 2

didMine correctness

📅 06, Aug 2019

🕒 21.91 ms

Line 67-72 in File TellorGetters.sol

```
67 /*@CTK "didMine correctness"
68     @post __return == tellor.minersByChallenge[_challenge][_miner]
69     @post !(__has_overflow)
70     @post !(__has_assertion_failure)
71     @post !(__has_assertion_failure)
72 */
```

Line 73-75 in File TellorGetters.sol

```
73 function didMine(bytes32 _challenge, address _miner) external view returns(bool){
74     return tellor.didMine(_challenge,_miner);
75 }
```

✅ The code meets the specification.

Formal Verification Request 3

didVote correctness

📅 06, Aug 2019

🕒 22.15 ms

Line 84-89 in File TellorGetters.sol

```
84  /*@CTK "didVote correctness"
85    @post __return == tellor.disputesById[_disputeId].voted[_address]
86    @post !(__has_overflow)
87    @post !(__has_assertion_failure)
88    @post !(__has_assertion_failure)
89    */
```

Line 90-92 in File TellorGetters.sol

```
90  function didVote(uint _disputeId, address _address) external view returns(bool){
91    return tellor.didVote(_disputeId,_address);
92  }
```

✓ The code meets the specification.

Formal Verification Request 4

getAddressVars correctness

📅 06, Aug 2019

🕒 21.03 ms

Line 102-107 in File TellorGetters.sol

```
102 /*@CTK "getAddressVars correctness"
103    @post __return == tellor.addressVars[_data]
104    @post !(__has_overflow)
105    @post !(__has_assertion_failure)
106    @post !(__has_assertion_failure)
107    */
```

Line 108-110 in File TellorGetters.sol

```
108 function getAddressVars(bytes32 _data) view external returns(address){
109    return tellor.getAddressVars(_data);
110 }
```

✓ The code meets the specification.

Formal Verification Request 5

getDisputeIdByDisputeHash correctness

📅 06, Aug 2019

🕒 21.92 ms

Line 152-157 in File TellorGetters.sol

```
152 /*@CTK "getDisputeIdByDisputeHash correctness"
153    @post __return == tellor.disputeIdByDisputeHash[_hash]
154    @post !(__has_overflow)
155    @post !(__has_assertion_failure)
156    @post !(__has_assertion_failure)
157    */
```

Line 158-160 in File TellorGetters.sol

```
158     function getDisputeIdByDisputeHash(bytes32 _hash) external view returns(uint){
159         return tellor.getDisputeIdByDisputeHash(_hash);
160     }
```

✓ The code meets the specification.

Formal Verification Request 6

getDisputeUintVars correctness

06, Aug 2019

22.09 ms

Line 171-176 in File TellorGetters.sol

```
171     /*@CTK "getDisputeUintVars correctness"
172         @post __return == tellor.disputesById[_disputeId].disputeUintVars[_data]
173         @post !(__has_overflow)
174         @post !(__has_assertion_failure)
175         @post !(__has_assertion_failure)
176     */
```

Line 177-179 in File TellorGetters.sol

```
177     function getDisputeUintVars(uint _disputeId, bytes32 _data) external view returns(
178         uint){
179         return tellor.getDisputeUintVars(_disputeId, _data);
180     }
```

✓ The code meets the specification.

Formal Verification Request 7

getMinedBlockNum correctness

06, Aug 2019

21.85 ms

Line 208-213 in File TellorGetters.sol

```
208     /*@CTK "getMinedBlockNum correctness"
209         @post __return == tellor.requestDetails[_requestId].minedBlockNum[_timestamp]
210         @post !(__has_overflow)
211         @post !(__has_assertion_failure)
212         @post !(__has_assertion_failure)
213     */
```

Line 214-216 in File TellorGetters.sol

```
214     function getMinedBlockNum(uint _requestId, uint _timestamp) external view returns(
215         uint){
216         return tellor.getMinedBlockNum(_requestId, _timestamp);
217     }
```

✓ The code meets the specification.

Formal Verification Request 8

getMinersByRequestIdAndTimestamp correctness

📅 06, Aug 2019

🕒 22.42 ms

Line 225-230 in File TellorGetters.sol

```
225  /*@CTK "getMinersByRequestIdAndTimestamp correctness"
226      @post __return == tellor.requestDetails[_requestId].minersByValue[_timestamp]
227      @post !(__has_overflow)
228      @post !(__has_assertion_failure)
229      @post !(__has_assertion_failure)
230  */
```

Line 231-233 in File TellorGetters.sol

```
231  function getMinersByRequestIdAndTimestamp(uint _requestId, uint _timestamp)
      external view returns(address[5] memory){
232      return tellor.getMinersByRequestIdAndTimestamp(_requestId,_timestamp);
233  }
```

✅ The code meets the specification.

Formal Verification Request 9

getName correctness

📅 06, Aug 2019

🕒 25.68 ms

Line 240-245 in File TellorGetters.sol

```
240  /*@CTK "getName correctness"
241      @post __return == "Tellor Tributes"
242      @post !(__has_overflow)
243      @post !(__has_assertion_failure)
244      @post !(__has_assertion_failure)
245  */
```

Line 246-248 in File TellorGetters.sol

```
246  function getName() external view returns(string memory){
247      return tellor.getName();
248  }
```

✅ The code meets the specification.

Formal Verification Request 10

getNewValueCountybyRequestId correctness

📅 06, Aug 2019

🕒 21.18 ms

Line 258-263 in File TellorGetters.sol

```
258 /*@CTK "getNewValueCountbyRequestId correctness"
259     @post __return == tellor.requestDetails[_requestId].requestTimestamps.length
260     @post !(__has_overflow)
261     @post !(__has_assertion_failure)
262     @post !(__has_assertion_failure)
263 */
```

Line 264-266 in File TellorGetters.sol

```
264 function getNewValueCountbyRequestId(uint _requestId) external view returns(uint){
265     return tellor.getNewValueCountbyRequestId(_requestId);
266 }
```

✓ The code meets the specification.

Formal Verification Request 11

getRequestIdByRequestQIndex correctness

📅 06, Aug 2019

🕒 38.08 ms

Line 274-280 in File TellorGetters.sol

```
274 /*@CTK "getRequestIdByRequestQIndex correctness"
275     @post _index > 50 -> __reverted == true
276     @post _index <= 50 -> __return == tellor.requestIdByRequestQIndex[_index]
277     @post !(__has_overflow)
278     @post !(__has_assertion_failure)
279     @post !(__has_assertion_failure)
280 */
```

Line 281-283 in File TellorGetters.sol

```
281 function getRequestIdByRequestQIndex(uint _index) external view returns(uint){
282     return tellor.getRequestIdByRequestQIndex(_index);
283 }
```

✓ The code meets the specification.

Formal Verification Request 12

getRequestIdByTimestamp correctness

📅 06, Aug 2019

🕒 20.52 ms

Line 291-296 in File TellorGetters.sol

```
291 /*@CTK "getRequestIdByTimestamp correctness"
292     @post __return == tellor.requestIdByTimestamp[_timestamp]
293     @post !(__has_overflow)
294     @post !(__has_assertion_failure)
295     @post !(__has_assertion_failure)
296 */
```

Line 297-299 in File TellorGetters.sol

```
297     function getRequestIdByTimestamp(uint _timestamp) external view returns(uint){
298         return tellor.getRequestIdByTimestamp(_timestamp);
299     }
```

✓ The code meets the specification.

Formal Verification Request 13

getRequestIdByQueryHash correctness

📅 06, Aug 2019

🕒 21.1 ms

Line 306-311 in File TellorGetters.sol

```
306     /*@CTK "getRequestIdByQueryHash correctness"
307         @post __return == tellor.requestIdByQueryHash[_request]
308         @post !(__has_overflow)
309         @post !(__has_assertion_failure)
310         @post !(__has_assertion_failure)
311     */
```

Line 312-314 in File TellorGetters.sol

```
312     function getRequestIdByQueryHash(bytes32 _request) external view returns(uint){
313         return tellor.getRequestIdByQueryHash(_request);
314     }
```

✓ The code meets the specification.

Formal Verification Request 14

getRequestQ correctness

📅 06, Aug 2019

🕒 21.25 ms

Line 321-326 in File TellorGetters.sol

```
321     /*@CTK "getRequestQ correctness"
322         @post __return == tellor.requestQ
323         @post !(__has_overflow)
324         @post !(__has_assertion_failure)
325         @post !(__has_assertion_failure)
326     */
```

Line 327-329 in File TellorGetters.sol

```
327     function getRequestQ() view public returns(uint[51] memory){
328         return tellor.getRequestQ();
329     }
```

✓ The code meets the specification.

Formal Verification Request 15

getRequestUintVars correctness

📅 06, Aug 2019

🕒 22.12 ms

Line 341-346 in File TellorGetters.sol

```
341  /*@CTK "getRequestUintVars correctness"
342    @post __return == tellor.requestDetails[_requestId].apiUintVars[_data]
343    @post !(__has_overflow)
344    @post !(__has_assertion_failure)
345    @post !(__has_assertion_failure)
346  */
```

Line 347-349 in File TellorGetters.sol

```
347  function getRequestUintVars(uint _requestId, bytes32 _data) external view returns(
348    uint){
349    return tellor.getRequestUintVars(_requestId, _data);
350  }
```

✅ The code meets the specification.

Formal Verification Request 16

getStakerInfo correctness

📅 06, Aug 2019

🕒 24.63 ms

Line 373-379 in File TellorGetters.sol

```
373  /*@CTK "getStakerInfo correctness"
374    @post __return == tellor.stakerDetails[_staker].currentStatus
375    @post __return1 == tellor.stakerDetails[_staker].startDate
376    @post !(__has_overflow)
377    @post !(__has_assertion_failure)
378    @post !(__has_assertion_failure)
379  */
```

Line 380-382 in File TellorGetters.sol

```
380  function getStakerInfo(address _staker) external view returns(uint, uint){
381    return tellor.getStakerInfo(_staker);
382  }
```

✅ The code meets the specification.

Formal Verification Request 17

getSubmissionsByTimestamp correctness

📅 06, Aug 2019

🕒 22.8 ms

Line 390-395 in File TellorGetters.sol

```
390  /*@CTK "getSubmissionsByTimestamp correctness"
391    @post __return == tellor.requestDetails[_requestId].valuesByTimestamp[_timestamp]
392    ]
393    @post !(__has_overflow)
394    @post !(__has_assertion_failure)
395    @post !(__has_assertion_failure)
396    */
```

Line 396-398 in File TellorGetters.sol

```
396  function getSubmissionsByTimestamp(uint _requestId, uint _timestamp) external view
      returns(uint[5] memory){
397    return tellor.getSubmissionsByTimestamp(_requestId,_timestamp);
398  }
```

✓ The code meets the specification.

Formal Verification Request 18

getSymbol correctness

📅 06, Aug 2019

🕒 21.52 ms

Line 404-409 in File TellorGetters.sol

```
404  /*@CTK "getSymbol correctness"
405    @post __return == "TT"
406    @post !(__has_overflow)
407    @post !(__has_assertion_failure)
408    @post !(__has_assertion_failure)
409    */
```

Line 410-412 in File TellorGetters.sol

```
410  function getSymbol() external view returns(string memory){
411    return tellor.getSymbol();
412  }
```

✓ The code meets the specification.

Formal Verification Request 19

Buffer overflow / array index out of bound would never happen.

📅 06, Aug 2019

🕒 398.39 ms

Line 420 in File TellorGetters.sol

```
420  //@CTK FAIL NO_BUF_OVERFLOW
```

Line 426-428 in File TellorGetters.sol

```
426  function getTimestampbyRequestIDandIndex(uint _requestID, uint _index) external
      view returns(uint){
427    return tellor.getTimestampbyRequestIDandIndex(_requestID,_index);
428  }
```


✗ This code violates the specification.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     _index = 128
5     _requestID = 0
6   }
7   This = 0
8   Internal = {
9     __has_assertion_failure = false
10    __has_buf_overflow = false
11    __has_overflow = false
12    __has_returned = false
13    __reverted = false
14    msg = {
15      "gas": 0,
16      "sender": 0,
17      "value": 0
18    }
19  }
20  Other = {
21    __return = 0
22    block = {
23      "number": 0,
24      "timestamp": 0
25    }
26  }
27  Address_Map = [
28    {
29      "key": "ALL_OTHERS",
30      "value": {
31        "contract_name": "TellorGetters",
32        "balance": 0,
33        "contract": {
34          "teller": {
35            "currentChallenge": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
36            "requestQ": [
37              {
38                "key": 0,
39                "value": 64
40              },
41              {
42                "key": "ALL_OTHERS",
43                "value": 0
44              }
45            ],
46            "newValueTimestamps": [],
47            "currentMiners": [
48              {
49                "value": 64,
50                "miner": 64
51              },
52              {
53                "value": 64,
54                "miner": 64
55              },
56              {
57                "value": 64,

```

[illegible]

page 25

[illegible]

```
217     "stakerDetails": [  
218         {  
219             "key": "ALL_OTHERS",  
220             "value": {  
221                 "currentStatus": 64,  
222                 "startDate": 64  
223             }  
224         }  
225     ],  
226     "requestDetails": [  
227         {  
228             "key": "ALL_OTHERS",  
229             "value": {  
230                 "queryString": "",  
231                 "dataSymbol": "",  
232                 "queryHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",  
233                 "requestTimestamps": [  
234                     0  
235                 ],  
236                 "apiUintVars": [  
237                     {  
238                         "key": "ALL_OTHERS",  
239                         "value": 0  
240                     }  
241                 ],  
242                 "minedBlockNum": [  
243                     {  
244                         "key": 0,  
245                         "value": 1  
246                     },  
247                     {  
248                         "key": "ALL_OTHERS",  
249                         "value": 0  
250                     }  
251                 ],  
252                 "finalValues": [  
253                     {  
254                         "key": 2,  
255                         "value": 0  
256                     },  
257                     {  
258                         "key": 1,  
259                         "value": 32  
260                     },  
261                     {  
262                         "key": 4,  
263                         "value": 4  
264                     },  
265                     {  
266                         "key": "ALL_OTHERS",  
267                         "value": 128  
268                     }  
269                 ],  
270                 "inDispute": [  
271                     {  
272                         "key": "ALL_OTHERS",  
273                         "value": false  
274                     }  
275                 ]  
276             }  
277         }  
278     ]  
279 }
```

[illegible]

[illegible]

```

369 After Execution:
370     Input = {
371         _index = 128
372         _requestID = 0
373     }
374     This = 0
375     Internal = {
376         __has_assertion_failure = false

```

```

377     __has_buf_overflow = true
378     __has_overflow = false
379     __has_returned = true
380     __reverted = false
381     msg = {
382         "gas": 0,
383         "sender": 0,
384         "value": 0
385     }
386 }
387 Other = {
388     __return = 1
389     block = {
390         "number": 0,
391         "timestamp": 0
392     }
393 }
394 Address_Map = [
395     {
396         "key": 0,
397         "value": {
398             "contract_name": "TellorGetters",
399             "balance": 0,
400             "contract": {
401                 "tellor": {
402                     "currentChallenge": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
403                     "requestQ": [
404                         {
405                             "key": 0,
406                             "value": 64
407                         },
408                         {
409                             "key": "ALL_OTHERS",
410                             "value": 0
411                         }
412                     ],
413                     "newValueTimestamps": [],
414                     "currentMiners": [
415                         {
416                             "value": 64,
417                             "miner": 64
418                         },
419                         {
420                             "value": 64,
421                             "miner": 64
422                         },
423                         {
424                             "value": 64,
425                             "miner": 64
426                         },
427                         {
428                             "value": 64,
429                             "miner": 64
430                         },
431                         {
432                             "value": 64,
433                             "miner": 64
434                     ]

```


[illegible]

[illegible]

```

536         {
537             "key": "ALL_OTHERS",
538             "value": 0
539         }
540     ],
541     "voted": [
542         {
543             "key": "ALL_OTHERS",
544             "value": false
545         }
546     ]
547 }
548 }
549 ],
550 "balances": [
551     {
552         "key": "ALL_OTHERS",
553         "value": [
554             {
555                 "key": "ALL_OTHERS",
556                 "value": {
557                     "fromBlock": 64,
558                     "value": 64
559                 }
560             }
561         ]
562     }
563 ],
564 "allowed": [
565     {
566         "key": 8,
567         "value": [
568             {
569                 "key": "ALL_OTHERS",
570                 "value": 1
571             }
572         ]
573     },
574     {
575         "key": "ALL_OTHERS",
576         "value": [
577             {
578                 "key": "ALL_OTHERS",
579                 "value": 0
580             }
581         ]
582     }
583 ],
584 "stakerDetails": [
585     {
586         "key": "ALL_OTHERS",
587         "value": {
588             "currentStatus": 64,
589             "startDate": 64
590         }
591     }
592 ],
593 "requestDetails": [

```

```

594 {
595   "key": "ALL_OTHERS",
596   "value": {
597     "queryString": "",
598     "dataSymbol": "",
599     "queryHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
600     "requestTimestamps": [
601       0
602     ],
603     "apiUintVars": [
604       {
605         "key": "ALL_OTHERS",
606         "value": 0
607       }
608     ],
609     "minedBlockNum": [
610       {
611         "key": 0,
612         "value": 1
613       },
614       {
615         "key": "ALL_OTHERS",
616         "value": 0
617       }
618     ],
619     "finalValues": [
620       {
621         "key": 2,
622         "value": 0
623       },
624       {
625         "key": 1,
626         "value": 32
627       },
628       {
629         "key": 4,
630         "value": 4
631       },
632       {
633         "key": "ALL_OTHERS",
634         "value": 128
635       }
636     ],
637     "inDispute": [
638       {
639         "key": "ALL_OTHERS",
640         "value": false
641       }
642     ],
643     "minersByValue": [
644       {
645         "key": 16,
646         "value": [
647           16,
648           16,
649           16,
650           16,
651           16

```

page 35

[illegible]

[illegible]

[illegible]

[illegible]

```
913     {
914         "key": "ALL_OTHERS",
915         "value": [
916             {
917                 "key": "ALL_OTHERS",
918                 "value": 0
919             }
920         ]
921     }
922 ],
923 "stakerDetails": [
924     {
925         "key": "ALL_OTHERS",
926         "value": {
927             "currentStatus": 64,
928             "startDate": 64
929         }
930     }
931 ],
932 "requestDetails": [
933     {
934         "key": "ALL_OTHERS",
935         "value": {
936             "queryString": "",
937             "dataSymbol": "",
938             "queryHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
939             "requestTimestamps": [
940                 0
941             ],
942             "apiUintVars": [
943                 {
944                     "key": "ALL_OTHERS",
945                     "value": 0
946                 }
947             ],
948             "minedBlockNum": [
949                 {
950                     "key": 0,
951                     "value": 1
952                 },
953                 {
954                     "key": "ALL_OTHERS",
955                     "value": 0
956                 }
957             ],
958             "finalValues": [
959                 {
960                     "key": 2,
961                     "value": 0
962                 },
963                 {
964                     "key": 1,
965                     "value": 32
966                 },
967                 {
968                     "key": 4,
969                     "value": 4
970                 },
971             ],
```


```
971         {
972             "key": "ALL_OTHERS",
973             "value": 128
974         }
975     ],
976     "inDispute": [
977         {
978             "key": "ALL_OTHERS",
979             "value": false
980         }
981     ],
982     "minersByValue": [
983         {
984             "key": 16,
985             "value": [
986                 16,
987                 16,
988                 16,
989                 16,
990                 16
991             ]
992         },
993         {
994             "key": "ALL_OTHERS",
995             "value": [
996                 0,
997                 0,
998                 0,
999                 0,
1000                0
1001            ]
1002        }
1003    ],
1004     "valuesByTimestamp": [
1005         {
1006             "key": 8,
1007             "value": [
1008                 0,
1009                 0,
1010                 0,
1011                 0,
1012                 0
1013             ]
1014         },
1015         {
1016             "key": "ALL_OTHERS",
1017             "value": [
1018                 16,
1019                 16,
1020                 16,
1021                 16,
1022                 16
1023             ]
1024         }
1025     ]
1026 }
1027 ]
1028 ,
```

page 42

1073]

Formal Verification Request 20

getTimestampbyRequestIDandIndex correctness

 06, Aug 2019 0.47 ms

Line 421-425 in File TellorGetters.sol

```
421 /*@CTK "getTimestampbyRequestIDandIndex correctness"
422    @post __return == tellor.requestDetails[_requestID].requestTimestamps[_index]
423    @post !(__has_overflow)
424    @post !(__has_assertion_failure)
425 */
```


Line 426-428 in File TellorGetters.sol

```
426 function getTimestampbyRequestIDandIndex(uint _requestID, uint _index) external
    view returns(uint){
427     return tellor.getTimestampbyRequestIDandIndex(_requestID,_index);
428 }
```

 The code meets the specification.

Formal Verification Request 21

getUintVar correctness

 06, Aug 2019 20.83 ms

Line 440-445 in File TellorGetters.sol

```
440 /*@CTK "getUintVar correctness"
441    @post __return == tellor.uintVars[_data]
442    @post !(__has_overflow)
443    @post !(__has_buf_overflow)
444    @post !(__has_assertion_failure)
445 */
```


Line 446-448 in File TellorGetters.sol

```
446 function getUintVar(bytes32 _data) view public returns(uint){
447     return tellor.getUintVar(_data);
448 }
```

 The code meets the specification.

Formal Verification Request 22

isInDispute correctness

 06, Aug 2019 22.47 ms

Line 466-471 in File TellorGetters.sol

```
466  /*@CTK "isInDispute correctness"
467    @post __return == tellor.requestDetails[_requestId].inDispute[_timestamp]
468    @post !(__has_overflow)
469    @post !(__has_buf_overflow)
470    @post !(__has_assertion_failure)
471  */
```

Line 472-474 in File TellorGetters.sol

```
472  function isInDispute(uint _requestId, uint _timestamp) external view returns(bool)
473  {
474    return tellor.isInDispute(_requestId,_timestamp);
}
```

✓ The code meets the specification.

Formal Verification Request 23

retrieveData correctness

📅 06, Aug 2019

🕒 22.11 ms

Line 483-488 in File TellorGetters.sol

```
483  /*@CTK "retrieveData correctness"
484    @post __return == tellor.requestDetails[_requestId].finalValues[_timestamp]
485    @post !(__has_overflow)
486    @post !(__has_buf_overflow)
487    @post !(__has_assertion_failure)
488  */
```

Line 489-491 in File TellorGetters.sol

```
489  function retrieveData(uint _requestId, uint _timestamp) external view returns (
490    uint) {
491    return tellor.retrieveData(_requestId,_timestamp);
}
```

✓ The code meets the specification.

Formal Verification Request 24

approve correctness

📅 06, Aug 2019

🕒 16.61 ms

Line 54-61 in File TellorTransfer.sol

```
54  /*@CTK "approve correctness"
55    @tag assume_completion
56    @post _spender != 0x0
57    @post self__post.allowed[msg.sender][_spender] == _amount
```

```
58     @post !(__has_overflow)
59     @post !(__has_buf_overflow)
60     @post !(__has_assertion_failure)
61     */
```

Line 62-70 in File TellorTransfer.sol

```
62     function approve(TellorStorage.TellorStorageStruct storage self, address _spender,
63         uint _amount) public returns (bool) {
64         /*@IGNORE
65         require(allowedToTrade(self,msg.sender,_amount));
66         @IGNORE*/
67         require(_spender != address(0));
68         self.allowed[msg.sender][_spender] = _amount;
69         emit Approval(msg.sender, _spender, _amount);
70         return true;
71     }
```

✓ The code meets the specification.

Formal Verification Request 25

allowance correctness

📅 06, Aug 2019

🕒 4.45 ms

Line 78-83 in File TellorTransfer.sol

```
78     /*@CTK "allowance correctness"
79     @post __return == self.allowed[_user][_spender]
80     @post !(__has_overflow)
81     @post !(__has_buf_overflow)
82     @post !(__has_assertion_failure)
83     */
```

Line 84-86 in File TellorTransfer.sol

```
84     function allowance(TellorStorage.TellorStorageStruct storage self, address _user,
85         address _spender) public view returns (uint) {
86         return self.allowed[_user][_spender];
87     }
```

✓ The code meets the specification.

Formal Verification Request 26

If method completes, integer overflow would not happen.

📅 06, Aug 2019

🕒 7.84 ms

Line 124 in File TellorTransfer.sol

```
124     /*@CTK NO_OVERFLOW
```

Line 131-140 in File TellorTransfer.sol

```
131     function balanceOfAt(TellorStorage.TellorStorageStruct storage self,address _user,  
132         uint _blockNumber) public view returns (uint) {  
133         if ((self.balances[_user].length == 0) || (self.balances[_user][0].fromBlock >  
134             _blockNumber)) {  
135             return 0;  
136         }  
137     else {  
138         /*@IGNORE  
139         return getBalanceAt(self.balances[_user], _blockNumber);  
140         @IGNORE*/  
141     }
```

✓ The code meets the specification.

Formal Verification Request 27

Buffer overflow / array index out of bound would never happen.

📅 06, Aug 2019

🕒 0.75 ms

Line 125 in File TellorTransfer.sol

```
125     //@CTK_NO_BUF_OVERFLOW
```

Line 131-140 in File TellorTransfer.sol

```
131     function balanceOfAt(TellorStorage.TellorStorageStruct storage self,address _user,  
132         uint _blockNumber) public view returns (uint) {  
133         if ((self.balances[_user].length == 0) || (self.balances[_user][0].fromBlock >  
134             _blockNumber)) {  
135             return 0;  
136         }  
137     else {  
138         /*@IGNORE  
139         return getBalanceAt(self.balances[_user], _blockNumber);  
140         @IGNORE*/  
141     }
```

✓ The code meets the specification.

Formal Verification Request 28

Method will not encounter an assertion failure.

📅 06, Aug 2019

🕒 0.32 ms

Line 126 in File TellorTransfer.sol

```
126     //@CTK_NO_ASF
```

Line 131-140 in File TellorTransfer.sol


```

131     function balanceOfAt(TellorStorage.TellorStorageStruct storage self,address _user,
132         uint _blockNumber) public view returns (uint) {
133         if ((self.balances[_user].length == 0) || (self.balances[_user][0].fromBlock >
134             _blockNumber)) {
135             return 0;
136         }
137         else {
138             /*@IGNORE
139             return getBalanceAt(self.balances[_user], _blockNumber);
140             @IGNORE*/
141         }
142     }

```

✓ The code meets the specification.

Formal Verification Request 29

balanceOfAt

📅 06, Aug 2019

🕒 1.34 ms

Line 127-130 in File TellorTransfer.sol

```

127     /*@CTK balanceOfAt
128         @post self.balances[_user].length == 0 -> __return == 0
129         @post self.balances[_user][0].fromBlock > _blockNumber -> __return == 0
130     */

```

Line 131-140 in File TellorTransfer.sol

```

131     function balanceOfAt(TellorStorage.TellorStorageStruct storage self,address _user,
132         uint _blockNumber) public view returns (uint) {
133         if ((self.balances[_user].length == 0) || (self.balances[_user][0].fromBlock >
134             _blockNumber)) {
135             return 0;
136         }
137         else {
138             /*@IGNORE
139             return getBalanceAt(self.balances[_user], _blockNumber);
140             @IGNORE*/
141         }
142     }

```

✓ The code meets the specification.

Formal Verification Request 30

If method completes, integer overflow would not happen.

📅 06, Aug 2019

🕒 28.23 ms

Line 149 in File TellorTransfer.sol

```

149     /*@CTK NO_OVERFLOW

```

Line 166-188 in File TellerTransfer.sol

```

166     function getBalanceAt(TellerStorage.Checkpoint[] storage checkpoints, uint _block)
167         view public returns (uint) {
168         if (checkpoints.length == 0) return 0;
169         if (_block >= checkpoints[checkpoints.length-1].fromBlock)
170             return checkpoints[checkpoints.length-1].value;
171         if (_block < checkpoints[0].fromBlock) return 0;
172         // Binary search of the value in the array
173         uint min = 0;
174         uint max = checkpoints.length-1;
175         /*@CTK getValueAt_forLoop
176         //  @var TellerStorage.Checkpoint[] checkpoints
177         @inv max > min || max <= min
178         @post max <= min
179         */
180         while (max > min) {
181             uint mid = (max + min + 1) / 2;
182             if (checkpoints[mid].fromBlock <= _block) {
183                 min = mid;
184             } else {
185                 max = mid-1;
186             }
187         }
188         return checkpoints[min].value;
189     }

```

✓ The code meets the specification.

Formal Verification Request 31

Buffer overflow / array index out of bound would never happen.

📅 06, Aug 2019

🕒 13.65 ms

Line 150 in File TellerTransfer.sol

```

150     // @CTK FAIL NO_BUF_OVERFLOW

```

Line 166-188 in File TellerTransfer.sol

```

166     function getBalanceAt(TellerStorage.Checkpoint[] storage checkpoints, uint _block)
167         view public returns (uint) {
168         if (checkpoints.length == 0) return 0;
169         if (_block >= checkpoints[checkpoints.length-1].fromBlock)
170             return checkpoints[checkpoints.length-1].value;
171         if (_block < checkpoints[0].fromBlock) return 0;
172         // Binary search of the value in the array
173         uint min = 0;
174         uint max = checkpoints.length-1;
175         /*@CTK getValueAt_forLoop
176         //  @var TellerStorage.Checkpoint[] checkpoints
177         @inv max > min || max <= min
178         @post max <= min
179         */
180         while (max > min) {
181             uint mid = (max + min + 1) / 2;

```

```

181         if (checkpoints[mid].fromBlock<=_block) {
182             min = mid;
183         } else {
184             max = mid-1;
185         }
186     }
187     return checkpoints[min].value;
188 }

```

✖ This code violates the specification.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     _block = 128
5     checkpoints = [
6       {
7         "key": 15,
8         "value": {
9           "fromBlock": 130,
10          "value": 0
11        }
12      },
13      {
14        "key": 16,
15        "value": {
16          "fromBlock": 0,
17          "value": 16
18        }
19      },
20      {
21        "key": "ALL_OTHERS",
22        "value": {
23          "fromBlock": 1,
24          "value": 0
25        }
26      }
27    ]
28  }
29  Internal = {
30    __has_assertion_failure = false
31    __has_buf_overflow = false
32    __has_overflow = false
33    __has_returned = false
34    __reverted = false
35    msg = {
36      "gas": 0,
37      "sender": 0,
38      "value": 0
39    }
40  }
41  Other = {
42    __return = 0
43    block = {
44      "number": 0,
45      "timestamp": 0
46    }
47  }
48  Address_Map = [

```

```


49     {
50         "key": "ALL_OTHERS",
51         "value": "EmptyAddress"
52     }
53 ]
54
55 After Execution:
56     Input = {
57         _block = 0
58         checkpoints = [
59             {
60                 "fromBlock": 0,
61                 "value": 0
62             }
63         ]
64     }
65     Internal = {
66         __has_assertion_failure = false
67         __has_buf_overflow = true
68         __has_overflow = false
69         __has_returned = true
70         __reverted = false
71         msg = {
72             "gas": 0,
73             "sender": 0,
74             "value": 0
75         }
76     }
77     Other = {
78         __return = 0
79         block = {
80             "number": 0,
81             "timestamp": 0
82         }
83         max = 1
84         min = 128
85     }
86     Address_Map = [
87         {
88             "key": "ALL_OTHERS",
89             "value": "EmptyAddress"
90         }
91     ]

```

Formal Verification Request 32

Method will not encounter an assertion failure.

 06, Aug 2019

 0.48 ms

Line 151 in File TellorTransfer.sol

151 `//@CTK NO_ASF`

Line 166-188 in File TellorTransfer.sol

```

166 function getBalanceAt(TellorStorage.Checkpoint[] storage checkpoints, uint _block)
167     view public returns (uint) {
168     if (checkpoints.length == 0) return 0;
169     if (_block >= checkpoints[checkpoints.length-1].fromBlock)
170         return checkpoints[checkpoints.length-1].value;
171     if (_block < checkpoints[0].fromBlock) return 0;
172     // Binary search of the value in the array
173     uint min = 0;
174     uint max = checkpoints.length-1;
175     /*@CTK getValueAt_forLoop
176     // @var TellorStorage.Checkpoint[] checkpoints
177     @inv max > min || max <= min
178     @post max <= min
179     */
180     while (max > min) {
181         uint mid = (max + min + 1) / 2;
182         if (checkpoints[mid].fromBlock <= _block) {
183             min = mid;
184         } else {
185             max = mid-1;
186         }
187     }
188     return checkpoints[min].value;
189 }

```

✓ The code meets the specification.

Formal Verification Request 33

getValueAt

📅 06, Aug 2019

🕒 1.76 ms

Line 152-155 in File TellorTransfer.sol

```

152 /*@CTK getValueAt
153 @pre checkpoints.length == 0
154 @post __return == 0
155 */

```

Line 166-188 in File TellorTransfer.sol

```

166 function getBalanceAt(TellorStorage.Checkpoint[] storage checkpoints, uint _block)
167     view public returns (uint) {
168     if (checkpoints.length == 0) return 0;
169     if (_block >= checkpoints[checkpoints.length-1].fromBlock)
170         return checkpoints[checkpoints.length-1].value;
171     if (_block < checkpoints[0].fromBlock) return 0;
172     // Binary search of the value in the array
173     uint min = 0;
174     uint max = checkpoints.length-1;
175     /*@CTK getValueAt_forLoop
176     // @var TellorStorage.Checkpoint[] checkpoints
177     @inv max > min || max <= min
178     @post max <= min
179     */
180     while (max > min) {

```

```

180     uint mid = (max + min + 1) / 2;
181     if (checkpoints[mid].fromBlock <= _block) {
182         min = mid;
183     } else {
184         max = mid-1;
185     }
186 }
187 return checkpoints[min].value;
188 }

```

✓ The code meets the specification.

Formal Verification Request 34

getValueAt_Min

06, Aug 2019

1.83 ms

Line 156-160 in File TellorTransfer.sol

```

156 /*@CTK getValueAt_Min
157   @pre checkpoints.length > 0 && _block < checkpoints[0].fromBlock &&
158       _block < checkpoints[checkpoints.length-1].fromBlock
159   @post __return == 0
160 */

```

Line 166-188 in File TellorTransfer.sol

```

166 function getBalanceAt(TellorStorage.Checkpoint[] storage checkpoints, uint _block)
167     view public returns (uint) {
168     if (checkpoints.length == 0) return 0;
169     if (_block >= checkpoints[checkpoints.length-1].fromBlock)
170         return checkpoints[checkpoints.length-1].value;
171     if (_block < checkpoints[0].fromBlock) return 0;
172     // Binary search of the value in the array
173     uint min = 0;
174     uint max = checkpoints.length-1;
175     /*@CTK getValueAt_forLoop
176       // @var TellorStorage.Checkpoint[] checkpoints
177       @inv max > min || max <= min
178       @post max <= min
179     */
180     while (max > min) {
181         uint mid = (max + min + 1) / 2;
182         if (checkpoints[mid].fromBlock <= _block) {
183             min = mid;
184         } else {
185             max = mid-1;
186         }
187     }
188     return checkpoints[min].value;
189 }

```

✓ The code meets the specification.

Formal Verification Request 35

getValueAt_Max

📅 06, Aug 2019

🕒 2.06 ms

Line 161-165 in File TellorTransfer.sol

```
161 /*@CTK getValueAt_Max
162   @pre checkpoints.length > 0 &&
163       _block >= checkpoints[checkpoints.length-1].fromBlock
164   @post __return == checkpoints[checkpoints.length-1].value
165 */
```

Line 166-188 in File TellorTransfer.sol

```
166 function getBalanceAt(TellorStorage.Checkpoint[] storage checkpoints, uint _block)
    view public returns (uint) {
167     if (checkpoints.length == 0) return 0;
168     if (_block >= checkpoints[checkpoints.length-1].fromBlock)
169         return checkpoints[checkpoints.length-1].value;
170     if (_block < checkpoints[0].fromBlock) return 0;
171     // Binary search of the value in the array
172     uint min = 0;
173     uint max = checkpoints.length-1;
174     /*@CTK getValueAt_forLoop
175     //   @var TellorStorage.Checkpoint[] checkpoints
176         @inv max > min || max <= min
177         @post max <= min
178     */
179     while (max > min) {
180         uint mid = (max + min + 1) / 2;
181         if (checkpoints[mid].fromBlock <= _block) {
182             min = mid;
183         } else {
184             max = mid-1;
185         }
186     }
187     return checkpoints[min].value;
188 }
```

✅ The code meets the specification.

Formal Verification Request 36

If method completes, integer overflow would not happen.

📅 06, Aug 2019

🕒 24.96 ms

Line 217 in File TellorTransfer.sol

```
217 //@CTK FAIL NO_OVERFLOW
```

Line 220-229 in File TellorTransfer.sol

```

220 function updateBalanceAtNow(TellorStorage.Checkpoint[] storage checkpoints, uint
    _value) public {
221     if ((checkpoints.length == 0) || (checkpoints[checkpoints.length - 1].fromBlock
        < block.number)) {
222         TellorStorage.Checkpoint storage newCheckPoint = checkpoints[
            checkpoints.length++ ];
223         newCheckPoint.fromBlock = uint128(block.number);
224         newCheckPoint.value = uint128(_value);
225     } else {
226         TellorStorage.Checkpoint storage oldCheckPoint = checkpoints[checkpoints
            .length-1];
227         oldCheckPoint.value = uint128(_value);
228     }
229 }

```

✗ This code violates the specification.

```

1 Counter Example:
2 Before Execution:
3     Input = {
4         _value = 0
5         checkpoints = [
6             {
7                 "key": 0,
8                 "value": {
9                     "fromBlock": 0,
10                    "value": 32
11                }
12            },
13            {
14                "key": "ALL_OTHERS",
15                "value": {
16                    "fromBlock": 0,
17                    "value": 0
18                }
19            }
20        ]
21    }
22    Internal = {
23        __has_assertion_failure = false
24        __has_buf_overflow = false
25        __has_overflow = false
26        __has_returned = false
27        __reverted = false
28        msg = {
29            "gas": 0,
30            "sender": 0,
31            "value": 0
32        }
33    }
34    Other = {
35        block = {
36            "number": 2,
37            "timestamp": 0
38        }
39    }
40    Address_Map = [
41        {
42            "key": "ALL_OTHERS",

```



```


43     "value": "EmptyAddress"
44   }
45 ]
46
47 After Execution:
48   Input = {
49     _value = 0
50     checkpoints = []
51   }
52   Internal = {
53     __has_assertion_failure = false
54     __has_buf_overflow = true
55     __has_overflow = true
56     __has_returned = false
57     __reverted = false
58     msg = {
59       "gas": 0,
60       "sender": 0,
61       "value": 0
62     }
63   }
64   Other = {
65     block = {
66       "number": 2,
67       "timestamp": 0
68     }
69     newCheckPoint = {
70       "fromBlock": 2,
71       "value": 0
72     }
73     oldCheckPoint = {
74       "fromBlock": 0,
75       "value": 0
76     }
77   }
78   Address_Map = [
79     {
80       "key": "ALL_OTHERS",
81       "value": "EmptyAddress"
82     }
83   ]

```

Formal Verification Request 37

Buffer overflow / array index out of bound would never happen.

 06, Aug 2019

 11.36 ms

Line 218 in File TellorTransfer.sol

```
218 // @CTK FAIL NO_BUF_OVERFLOW
```

Line 220-229 in File TellorTransfer.sol

```

220 function updateBalanceAtNow(TellorStorage.Checkpoint[] storage checkpoints, uint
    _value) public {

```

```

221     if ((checkpoints.length == 0) || (checkpoints[checkpoints.length - 1].fromBlock
222         < block.number)) {
223         TellerStorage.Checkpoint storage newCheckPoint = checkpoints[
224             checkpoints.length++ ];
225         newCheckPoint.fromBlock = uint128(block.number);
226         newCheckPoint.value = uint128(_value);
227     } else {
228         TellerStorage.Checkpoint storage oldCheckPoint = checkpoints[checkpoints
229             .length-1];
230         oldCheckPoint.value = uint128(_value);
231     }
232 }

```

✗ This code violates the specification.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     _value = 0
5     checkpoints = [
6       {
7         "key": "ALL_OTHERS",
8         "value": {
9           "fromBlock": 0,
10          "value": 0
11        }
12      }
13    ]
14  }
15  Internal = {
16    __has_assertion_failure = false
17    __has_buf_overflow = false
18    __has_overflow = false
19    __has_returned = false
20    __reverted = false
21    msg = {
22      "gas": 0,
23      "sender": 0,
24      "value": 0
25    }
26  }
27  Other = {
28    block = {
29      "number": 1,
30      "timestamp": 0
31    }
32  }
33  Address_Map = [
34    {
35      "key": "ALL_OTHERS",
36      "value": "EmptyAddress"
37    }
38  ]
39
40 After Execution:
41   Input = {
42     _value = 0
43     checkpoints = []
44   }

```

```


45     Internal = {
46         __has_assertion_failure = false
47         __has_buf_overflow = true
48         __has_overflow = true
49         __has_returned = false
50         __reverted = false
51         msg = {
52             "gas": 0,
53             "sender": 0,
54             "value": 0
55         }
56     }
57     Other = {
58         block = {
59             "number": 1,
60             "timestamp": 0
61         }
62         newCheckPoint = {
63             "fromBlock": 1,
64             "value": 0
65         }
66         oldCheckPoint = {
67             "fromBlock": 0,
68             "value": 0
69         }
70     }
71     Address_Map = [
72     {
73         "key": "ALL_OTHERS",
74         "value": "EmptyAddress"
75     }
76 ]

```

Formal Verification Request 38

Method will not encounter an assertion failure.

 06, Aug 2019

 0.42 ms

Line 219 in File TellorTransfer.sol

219 //@CTK NO_ASF

Line 220-229 in File TellorTransfer.sol

```

220     function updateBalanceAtNow(TellorStorage.Checkpoint[] storage checkpoints, uint
221         _value) public {
222         if ((checkpoints.length == 0) || (checkpoints[checkpoints.length - 1].fromBlock
223             < block.number)) {
224             TellorStorage.Checkpoint storage newCheckPoint = checkpoints[
225                 checkpoints.length++ ];
226             newCheckPoint.fromBlock = uint128(block.number);
227             newCheckPoint.value = uint128(_value);
228         } else {
229             TellorStorage.Checkpoint storage oldCheckPoint = checkpoints[checkpoints
230                 .length-1];
231             oldCheckPoint.value = uint128(_value);

```

```
228     }
229 }
```

✓ The code meets the specification.

Formal Verification Request 39

getValueAt_forLoop__Generated

📅 06, Aug 2019

🕒 15.35 ms

(Loop) Line 174-178 in File TellorTransfer.sol

```
174  /*@CTK getValueAt_forLoop
175  //   @var TellorStorage.Checkpoint[] checkpoints
176      @inv max > min || max <= min
177      @post max <= min
178  */
```

(Loop) Line 174-186 in File TellorTransfer.sol

```
174  /*@CTK getValueAt_forLoop
175  //   @var TellorStorage.Checkpoint[] checkpoints
176      @inv max > min || max <= min
177      @post max <= min
178  */
179  while (max > min) {
180      uint mid = (max + min + 1) / 2;
181      if (checkpoints[mid].fromBlock <= _block) {
182          min = mid;
183      } else {
184          max = mid-1;
185      }
186  }
```

✓ The code meets the specification.

Formal Verification Request 40

Method will not encounter an assertion failure.

📅 06, Aug 2019

🕒 11.99 ms

Line 6 in File SafeMath.sol

```
6  //@CTK FAIL NO_ASF
```

Line 13-17 in File SafeMath.sol

```
13  function add(uint256 a, uint256 b) internal pure returns (uint256) {
14      uint256 c = a + b;
15      assert(c >= a);
16      return c;
17  }
```

✗ This code violates the specification.

```


1 Counter Example:
2 Before Execution:
3   Input = {
4     a = 211
5     b = 173
6   }
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19  Other = {
20    __return = 0
21    block = {
22      "number": 0,
23      "timestamp": 0
24    }
25  }
26  Address_Map = [
27    {
28      "key": "ALL_OTHERS",
29      "value": "EmptyAddress"
30    }
31  ]
32
33 Function invocation is reverted.

```

Formal Verification Request 41

SafeMath add

 06, Aug 2019

 2.38 ms

Line 7-12 in File SafeMath.sol

```

7  /*@CTK "SafeMath add"
8    @post (a + b < a || a + b < b) == __reverted
9    @post !__reverted -> __return == a + b
10   @post !__reverted -> !__has_overflow
11   @post !(__has_buf_overflow)
12  */

```

Line 13-17 in File SafeMath.sol

```

13 function add(uint256 a, uint256 b) internal pure returns (uint256) {
14   uint256 c = a + b;
15   assert(c >= a);
16   return c;
17 }

```

✓ The code meets the specification.

Formal Verification Request 42

SafeMath max

📅 06, Aug 2019

🕒 4.4 ms

Line 31-38 in File SafeMath.sol

```
31  /*@CTK "SafeMath max"
32     @post !__reverted
33     @post (a > b) -> __return == a
34     @post (a <= b) -> __return == b
35     @post !(__has_overflow)
36     @post !(__has_buf_overflow)
37     @post !(__has_assertion_failure)
38  */
```

Line 39-41 in File SafeMath.sol

```
39  function max(uint a, uint b) internal pure returns (uint256) {
40      return a > b ? a : b;
41  }
```

✓ The code meets the specification.

Formal Verification Request 43

SafeMath min

📅 06, Aug 2019

🕒 4.74 ms

Line 47-54 in File SafeMath.sol

```
47  /*@CTK "SafeMath min"
48     @post !__reverted
49     @post (a < b) -> __return == a
50     @post (a >= b) -> __return == b
51     @post !(__has_overflow)
52     @post !(__has_buf_overflow)
53     @post !(__has_assertion_failure)
54  */
```

Line 55-57 in File SafeMath.sol

```
55  function min(uint a, uint b) internal pure returns (uint256) {
56      return a < b ? a : b;
57  }
```

✓ The code meets the specification.

Formal Verification Request 44

Method will not encounter an assertion failure.

📅 06, Aug 2019

🕒 14.72 ms

Line 59 in File SafeMath.sol

59 `//@CTK FAIL NO_ASF`

Line 66-70 in File SafeMath.sol

```

66 function mul(uint256 a, uint256 b) internal pure returns (uint256) {
67     uint256 c = a * b;
68     assert(a == 0 || c / a == b);
69     return c;
70 }

```

✖ This code violates the specification.

```


1 Counter Example:
2 Before Execution:
3   Input = {
4     a = 94
5     b = 128
6   }
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19  Other = {
20    __return = 0
21    block = {
22      "number": 0,
23      "timestamp": 0
24    }
25  }
26  Address_Map = [
27    {
28      "key": "ALL_OTHERS",
29      "value": "EmptyAddress"
30    }
31  ]
32
33 Function invocation is reverted.

```

Formal Verification Request 45

SafeMath mul

📅 06, Aug 2019

 73.1 ms

Line 60-65 in File SafeMath.sol

```

60  /*@CTK "SafeMath mul"
61    @post (((a) > (0)) && (((a) * (b)) / (a)) != (b))) == (__reverted)
62    @post !__reverted -> __return == a * b
63    @post !__reverted == !__has_overflow
64    @post !(__has_buf_overflow)
65  */

```

Line 66-70 in File SafeMath.sol

```

66  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
67    uint256 c = a * b;
68    assert(a == 0 || c / a == b);
69    return c;
70  }


```

 The code meets the specification.

Formal Verification Request 46

Method will not encounter an assertion failure.

 06, Aug 2019

 11.13 ms

Line 72 in File SafeMath.sol

```

72  //@CTK FAIL NO_ASF

```

Line 79-82 in File SafeMath.sol

```

79  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
80    assert(b <= a);
81    return a - b;
82  }

```

 This code violates the specification.

```

1  Counter Example:
2  Before Execution:
3    Input = {
4      a = 0
5      b = 1
6    }
7    Internal = {
8      __has_assertion_failure = false
9      __has_buf_overflow = false
10     __has_overflow = false
11     __has_returned = false
12     __reverted = false
13     msg = {
14       "gas": 0,
15       "sender": 0,
16       "value": 0
17     }
18   }

```



```
19     Other = {
20         __return = 0
21         block = {
22             "number": 0,
23             "timestamp": 0
24         }
25     }
26     Address_Map = [
27     {
28         "key": "ALL_OTHERS",
29         "value": "EmptyAddress"
30     }
31 ]
32
33 Function invocation is reverted.
```

Formal Verification Request 47

SafeMath sub

📅 06, Aug 2019

🕒 0.8 ms

Line 73-78 in File SafeMath.sol

```
73  /*@CTK "SafeMath sub"
74    @post (a < b) == __reverted
75    @post !__reverted -> __return == a - b
76    @post !__reverted -> !__has_overflow
77    @post !(__has_buf_overflow)
78  */
```

Line 79-82 in File SafeMath.sol

```
79  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
80      assert(b <= a);
81      return a - b;
82  }
```

✅ The code meets the specification.

Formal Verification Request 48

didMine correctness

📅 06, Aug 2019

🕒 4.45 ms

Line 51-56 in File TellorGettersLibrary.sol

```
51  /*@CTK "didMine correctness"
52    @post __return == self.minersByChallenge[_challenge][_miner]
53    @post !(__has_overflow)
54    @post !(__has_assertion_failure)
55    @post !(__has_assertion_failure)
56  */
```

Line 57-59 in File TellorGettersLibrary.sol

```
57     function didMine(TellorStorage.TellorStorageStruct storage self, bytes32
    _challenge,address _miner) internal view returns(bool){
58         return self.minersByChallenge[_challenge][_miner];
59     }
```

✓ The code meets the specification.

Formal Verification Request 49

didVote correctness

📅 06, Aug 2019

🕒 4.47 ms

Line 68-73 in File TellorGettersLibrary.sol

```
68     /*@CTK "didVote correctness"
69         @post __return == self.disputesById[_disputeId].voted[_address]
70         @post !(__has_overflow)
71         @post !(__has_assertion_failure)
72         @post !(__has_assertion_failure)
73     */
```

Line 74-76 in File TellorGettersLibrary.sol

```
74     function didVote(TellorStorage.TellorStorageStruct storage self,uint _disputeId,
    address _address) internal view returns(bool){
75         return self.disputesById[_disputeId].voted[_address];
76     }
```

✓ The code meets the specification.

Formal Verification Request 50

getAddressVars correctness

📅 06, Aug 2019

🕒 4.68 ms

Line 86-91 in File TellorGettersLibrary.sol

```
86     /*@CTK "getAddressVars correctness"
87         @post __return == self.addressVars[_data]
88         @post !(__has_overflow)
89         @post !(__has_assertion_failure)
90         @post !(__has_assertion_failure)
91     */
```

Line 92-94 in File TellorGettersLibrary.sol

```
92     function getAddressVars(TellorStorage.TellorStorageStruct storage self, bytes32
    _data) view internal returns(address){
93         return self.addressVars[_data];
94     }
```

✓ The code meets the specification.

Formal Verification Request 51

getDisputeIdByDisputeHash correctness

📅 06, Aug 2019

🕒 3.76 ms

Line 138-143 in File TellorGettersLibrary.sol

```
138 /*@CTK "getDisputeIdByDisputeHash correctness"
139   @post __return == self.disputeIdByDisputeHash[_hash]
140   @post !(__has_overflow)
141   @post !(__has_assertion_failure)
142   @post !(__has_assertion_failure)
143 */
```

Line 144-146 in File TellorGettersLibrary.sol

```
144 function getDisputeIdByDisputeHash(TellorStorage.TellorStorageStruct storage self,
    bytes32 _hash) internal view returns(uint){
145     return self.disputeIdByDisputeHash[_hash];
146 }
```

✅ The code meets the specification.

Formal Verification Request 52

getDisputeUintVars correctness

📅 06, Aug 2019

🕒 4.07 ms

Line 157-162 in File TellorGettersLibrary.sol

```
157 /*@CTK "getDisputeUintVars correctness"
158   @post __return == self.disputesById[_disputeId].disputeUintVars[_data]
159   @post !(__has_overflow)
160   @post !(__has_assertion_failure)
161   @post !(__has_assertion_failure)
162 */
```

Line 163-165 in File TellorGettersLibrary.sol

```
163 function getDisputeUintVars(TellorStorage.TellorStorageStruct storage self,uint
    _disputeId,bytes32 _data) internal view returns(uint){
164     return self.disputesById[_disputeId].disputeUintVars[_data];
165 }
```

✅ The code meets the specification.

Formal Verification Request 53

getMinedBlockNum correctness

📅 06, Aug 2019

🕒 4.22 ms

Line 200-205 in File TellorGettersLibrary.sol

```
200 /*@CTK "getMinedBlockNum correctness"
201     @post __return == self.requestDetails[_requestId].minedBlockNum[_timestamp]
202     @post !(__has_overflow)
203     @post !(__has_assertion_failure)
204     @post !(__has_assertion_failure)
205 */
```

Line 206-208 in File TellorGettersLibrary.sol

```
206 function getMinedBlockNum(TellorStorage.TellorStorageStruct storage self,uint
    _requestId, uint _timestamp) internal view returns(uint){
207     return self.requestDetails[_requestId].minedBlockNum[_timestamp];
208 }
```

✓ The code meets the specification.

Formal Verification Request 54

getMinersByRequestIdAndTimestamp correctness

📅 06, Aug 2019

🕒 4.4 ms

Line 217-222 in File TellorGettersLibrary.sol

```
217 /*@CTK "getMinersByRequestIdAndTimestamp correctness"
218     @post __return == self.requestDetails[_requestId].minersByValue[_timestamp]
219     @post !(__has_overflow)
220     @post !(__has_assertion_failure)
221     @post !(__has_assertion_failure)
222 */
```

Line 223-225 in File TellorGettersLibrary.sol

```
223 function getMinersByRequestIdAndTimestamp(TellorStorage.TellorStorageStruct
    storage self, uint _requestId, uint _timestamp) internal view returns(address
    [5] memory){
224     return self.requestDetails[_requestId].minersByValue[_timestamp];
225 }
```

✓ The code meets the specification.

Formal Verification Request 55

getNewValueCountybyRequestId correctness

📅 06, Aug 2019

🕒 4.24 ms

Line 244-249 in File TellorGettersLibrary.sol

```
244 /*@CTK "getNewValueCountybyRequestId correctness"
245     @post __return == self.requestDetails[_requestId].requestTimestamps.length
246     @post !(__has_overflow)
247     @post !(__has_assertion_failure)
248     @post !(__has_assertion_failure)
249 */
```

Line 250-252 in File TellorGettersLibrary.sol

```
250     function getNewValueCountbyRequestId(TellorStorage.TellorStorageStruct storage
      self, uint _requestId) internal view returns(uint){
251         return self.requestDetails[_requestId].requestTimestamps.length;
252     }
```

✓ The code meets the specification.

Formal Verification Request 56

getRequestIdByRequestQIndex correctness

📅 06, Aug 2019

🕒 11.74 ms

Line 260-266 in File TellorGettersLibrary.sol

```
260     /*@CTK "getRequestIdByRequestQIndex correctness"
261         @post _index > 50 -> __reverted == true
262         @post _index <= 50 -> __return == self.requestIdByRequestQIndex[_index]
263         @post !(__has_overflow)
264         @post !(__has_assertion_failure)
265         @post !(__has_assertion_failure)
266     */
```

Line 267-270 in File TellorGettersLibrary.sol

```
267     function getRequestIdByRequestQIndex(TellorStorage.TellorStorageStruct storage
      self, uint _index) internal view returns(uint){
268         require(_index <= 50);
269         return self.requestIdByRequestQIndex[_index];
270     }
```

✓ The code meets the specification.

Formal Verification Request 57

getRequestIdByTimestamp correctness

📅 06, Aug 2019

🕒 4.29 ms

Line 278-283 in File TellorGettersLibrary.sol

```
278     /*@CTK "getRequestIdByTimestamp correctness"
279         @post __return == self.requestIdByTimestamp[_timestamp]
280         @post !(__has_overflow)
281         @post !(__has_assertion_failure)
282         @post !(__has_assertion_failure)
283     */
```

Line 284-286 in File TellorGettersLibrary.sol

```
284     function getRequestIdByTimestamp(TellorStorage.TellorStorageStruct storage self,
      uint _timestamp) internal view returns(uint){
285         return self.requestIdByTimestamp[_timestamp];
286     }
```

✓ The code meets the specification.

Formal Verification Request 58

getRequestIdByQueryHash correctness

📅 06, Aug 2019

🕒 4.7 ms

Line 294-299 in File TellorGettersLibrary.sol

```
294  /*@CTK "getRequestIdByQueryHash correctness"
295      @post __return == self.requestIdByQueryHash[_queryHash]
296      @post !(__has_overflow)
297      @post !(__has_assertion_failure)
298      @post !(__has_assertion_failure)
299  */
```

Line 300-302 in File TellorGettersLibrary.sol

```
300  function getRequestIdByQueryHash(TellorStorage.TellorStorageStruct storage self,
    bytes32 _queryHash) internal view returns(uint){
301      return self.requestIdByQueryHash[_queryHash];
302  }
```

✓ The code meets the specification.

Formal Verification Request 59

getRequestQ correctness

📅 06, Aug 2019

🕒 4.86 ms

Line 309-314 in File TellorGettersLibrary.sol

```
309  /*@CTK "getRequestQ correctness"
310      @post __return == self.requestQ
311      @post !(__has_overflow)
312      @post !(__has_assertion_failure)
313      @post !(__has_assertion_failure)
314  */
```

Line 315-317 in File TellorGettersLibrary.sol

```
315  function getRequestQ(TellorStorage.TellorStorageStruct storage self) view internal
    returns(uint[51] memory){
316      return self.requestQ;
317  }
```

✓ The code meets the specification.

Formal Verification Request 60

getRequestUintVars correctness

📅 06, Aug 2019

🕒 4.43 ms

Line 329-334 in File TellorGettersLibrary.sol

```
329 /*@CTK "getRequestUintVars correctness"
330     @post __return == self.requestDetails[_requestId].apiUintVars[_data]
331     @post !(__has_overflow)
332     @post !(__has_assertion_failure)
333     @post !(__has_assertion_failure)
334 */
```

Line 335-337 in File TellorGettersLibrary.sol

```
335 function getRequestUintVars(TellorStorage.TellorStorageStruct storage self,uint
    _requestId,bytes32 _data) internal view returns(uint){
336     return self.requestDetails[_requestId].apiUintVars[_data];
337 }
```

✅ The code meets the specification.

Formal Verification Request 61

getStakerInfo correctness

📅 06, Aug 2019

🕒 5.43 ms

Line 362-368 in File TellorGettersLibrary.sol

```
362 /*@CTK "getStakerInfo correctness"
363     @post __return == self.stakerDetails[_staker].currentStatus
364     @post __return1 == self.stakerDetails[_staker].startDate
365     @post !(__has_overflow)
366     @post !(__has_assertion_failure)
367     @post !(__has_assertion_failure)
368 */
```

Line 369-371 in File TellorGettersLibrary.sol

```
369 function getStakerInfo(TellorStorage.TellorStorageStruct storage self,address
    _staker) internal view returns(uint,uint){
370     return (self.stakerDetails[_staker].currentStatus,self.stakerDetails[_staker].
        startDate);
371 }
```

✅ The code meets the specification.

Formal Verification Request 62

getSubmissionsByTimestamp correctness

📅 06, Aug 2019

🕒 4.94 ms

Line 380-385 in File TellorGettersLibrary.sol

```
380 /*@CTK "getSubmissionsByTimestamp correctness"
381   @post __return == self.requestDetails[_requestId].valuesByTimestamp[_timestamp]
382   @post !(__has_overflow)
383   @post !(__has_assertion_failure)
384   @post !(__has_assertion_failure)
385 */
```

Line 386-388 in File TellorGettersLibrary.sol

```
386 function getSubmissionsByTimestamp(TellorStorage.TellorStorageStruct storage self,
   uint _requestId, uint _timestamp) internal view returns(uint[5] memory){
387   return self.requestDetails[_requestId].valuesByTimestamp[_timestamp];
388 }
```

✓ The code meets the specification.

Formal Verification Request 63

Buffer overflow / array index out of bound would never happen.

📅 06, Aug 2019

🕒 54.27 ms

Line 405 in File TellorGettersLibrary.sol

```
405 //@CTK FAIL NO_BUF_OVERFLOW
```

Line 411-413 in File TellorGettersLibrary.sol

```
411 function getTimestampbyRequestIDandIndex(TellorStorage.TellorStorageStruct storage
   self,uint _requestID, uint _index) internal view returns(uint){
412   return self.requestDetails[_requestID].requestTimestamps[_index];
413 }
```

✗ This code violates the specification.

```
1 Counter Example:
2 Before Execution:
3   Input = {
4     _index = 128
5     _requestID = 0
6     self = {
7       "currentChallenge": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
8       "requestQ": [
9         {
10          "key": 8,
11          "value": 8
12        },
13        {
14          "key": 0,
15          "value": 2
16        },
17        {
18          "key": 129,
19          "value": 1
20        },
21        {
```


page 71

[illegible]

[illegible]

```

175         00c2",
176         "value": 0
177     },
178     {
179         "key": "ALL_OTHERS",
180         "value": 4
181     }
182 ],
183 "voted": [
184     {
185         "key": "ALL_OTHERS",
186         "value": false
187     }
188 ]
189 },
190 ],
191 "balances": [
192     {
193         "key": "ALL_OTHERS",
194         "value": [
195             {
196                 "key": "ALL_OTHERS",
197                 "value": {
198                     "fromBlock": 129,
199                     "value": 129
200                 }
201             }
202         ]
203     }
204 ],
205 "allowed": [
206     {
207         "key": "ALL_OTHERS",
208         "value": [
209             {
210                 "key": 128,
211                 "value": 32
212             },
213             {
214                 "key": "ALL_OTHERS",
215                 "value": 4
216             }
217         ]
218     }
219 ],
220 "stakerDetails": [
221     {
222         "key": "ALL_OTHERS",
223         "value": {
224             "currentStatus": 129,
225             "startDate": 129
226         }
227     }
228 ],
229 "requestDetails": [
230     {
231         "key": "ALL_OTHERS",

```

[illegible]

```
284     {
285         "key": 2,
286         "value": 0
287     },
288     {
289         "key": 1,
290         "value": 64
291     },
292     {
293         "key": 0,
294         "value": 8
295     },
296     {
297         "key": 4,
298         "value": 0
299     },
300     {
301         "key": "ALL_OTHERS",
302         "value": 2
303     }
304 ],
305 "inDispute": [
306     {
307         "key": "ALL_OTHERS",
308         "value": false
309     }
310 ],
311 "minersByValue": [
312     {
313         "key": 1,
314         "value": [
315             64,
316             64,
317             64,
318             64,
319             64
320         ]
321     },
322     {
323         "key": "ALL_OTHERS",
324         "value": [
325             0,
326             0,
327             0,
328             0,
329             0
330         ]
331     }
332 ],
333 "valuesByTimestamp": [
334     {
335         "key": 1,
336         "value": [
337             0,
338             0,
339             0,
340             0,
341             0
```

[illegible]

page 78

[illegible]

[illegible]

[illegible]

```

599         "value": 0
600     },
601     {
602         "key": "ALL_OTHERS",
603         "value": 4
604     }
605 ],
606 "voted": [
607     {
608         "key": "ALL_OTHERS",
609         "value": false
610     }
611 ]
612 },
613 ],
614 "balances": [
615     {
616         "key": "ALL_OTHERS",
617         "value": [
618             {
619                 "key": "ALL_OTHERS",
620                 "value": {
621                     "fromBlock": 129,
622                     "value": 129
623                 }
624             }
625         ]
626     }
627 ],
628 "allowed": [
629     {
630         "key": "ALL_OTHERS",
631         "value": [
632             {
633                 "key": 128,
634                 "value": 32
635             },
636             {
637                 "key": "ALL_OTHERS",
638                 "value": 4
639             }
640         ]
641     }
642 ],
643 "stakerDetails": [
644     {
645         "key": "ALL_OTHERS",
646         "value": {
647             "currentStatus": 129,
648             "startDate": 129
649         }
650     }
651 ],
652 "requestDetails": [

```

[illegible]

```
706         "value": 128
707     },
708     {
709         "key": 2,
710         "value": 0
711     },
712     {
713         "key": 1,
714         "value": 64
715     },
716     {
717         "key": 0,
718         "value": 8
719     },
720     {
721         "key": 4,
722         "value": 0
723     },
724     {
725         "key": "ALL_OTHERS",
726         "value": 2
727     }
728 ],
729 "inDispute": [
730     {
731         "key": "ALL_OTHERS",
732         "value": false
733     }
734 ],
735 "minersByValue": [
736     {
737         "key": 1,
738         "value": [
739             64,
740             64,
741             64,
742             64,
743             64
744         ]
745     },
746     {
747         "key": "ALL_OTHERS",
748         "value": [
749             0,
750             0,
751             0,
752             0,
753             0
754         ]
755     }
756 ],
757 "valuesByTimestamp": [
758     {
759         "key": 1,
760         "value": [
761             0,
762             0,
763             0,
```


[illegible]

[illegible]

Formal Verification Request 64

getTimestampbyRequestIDandIndex correctness

 06, Aug 2019

 0.36 ms

Line 406-410 in File TellorGettersLibrary.sol

```
406     /*@CTK "getTimestampbyRequestIDandIndex correctness"
407         @post __return == self.requestDetails[_requestID].requestTimestamps[_index]
408         @post !(__has_overflow)
409         @post !(__has_assertion_failure)
410     */
```


Line 411-413 in File TellorGettersLibrary.sol

```
411     function getTimestampbyRequestIDandIndex(TellorStorage.TellorStorageStruct storage
         self,uint _requestID, uint _index) internal view returns(uint){
412         return self.requestDetails[_requestID].requestTimestamps[_index];
413     }
```

✓ The code meets the specification.

Formal Verification Request 65

getUintVar correctness

📅 06, Aug 2019

🕒 3.78 ms

Line 425-430 in File TellorGettersLibrary.sol

```
425     /*@CTK "getUintVar correctness"
426         @post __return == self.uintVars[_data]
427         @post !(__has_overflow)
428         @post !(__has_buf_overflow)
429         @post !(__has_assertion_failure)
430     */
```

Line 431-433 in File TellorGettersLibrary.sol

```
431     function getUintVar(TellorStorage.TellorStorageStruct storage self,bytes32 _data)
         view internal returns(uint){
432         return self.uintVars[_data];
433     }
```

✓ The code meets the specification.

Formal Verification Request 66

isInDispute correctness

📅 06, Aug 2019

🕒 4.49 ms

Line 464-469 in File TellorGettersLibrary.sol

```
464     /*@CTK "isInDispute correctness"
465         @post __return == self.requestDetails[_requestId].inDispute[_timestamp]
466         @post !(__has_overflow)
467         @post !(__has_buf_overflow)
468         @post !(__has_assertion_failure)
469     */
```

Line 470-472 in File TellorGettersLibrary.sol


```
470     function isInDispute(TellorStorage.TellorStorageStruct storage self, uint
         _requestId, uint _timestamp) internal view returns(bool){
471         return self.requestDetails[_requestId].inDispute[_timestamp];
472     }
```

✓ The code meets the specification.

Formal Verification Request 67

retrieveData correctness

 06, Aug 2019

 4.06 ms

Line 481-486 in File TellorGettersLibrary.sol

```
481  /*@CTK "retrieveData correctness"
482      @post __return == self.requestDetails[_requestId].finalValues[_timestamp]
483      @post !(__has_overflow)
484      @post !(__has_buf_overflow)
485      @post !(__has_assertion_failure)
486  */
```

Line 487-489 in File TellorGettersLibrary.sol

```
487  function retrieveData(TellorStorage.TellorStorageStruct storage self, uint
      _requestId, uint _timestamp) internal view returns (uint) {
488      return self.requestDetails[_requestId].finalValues[_timestamp];
489  }
```

 The code meets the specification.

Source Code with CertiK Labels

File Tellor.sol

```

1  pragma solidity ^0.5.0;
2
3  import "./libraries/SafeMath.sol";
4  import "./libraries/TellorStorage.sol";
5  import "./libraries/TellorTransfer.sol";
6  import "./libraries/TellorDispute.sol";
7  import "./libraries/TellorStake.sol";
8  import "./libraries/TellorLibrary.sol";
9
10 /**
11  * @title Tellor Oracle System
12  * @dev Oracle contract where miners can submit the proof of work along with the value
13  * The logic for this contract is in TellorLibrary.sol, TellorDispute.sol, TellorStake
14  * and TellorTransfer.sol
15  */
16 contract Tellor{
17
18     using SafeMath for uint256;
19
20     using TellorDispute for TellorStorage.TellorStorageStruct;
21     using TellorLibrary for TellorStorage.TellorStorageStruct;
22     using TellorStake for TellorStorage.TellorStorageStruct;
23     using TellorTransfer for TellorStorage.TellorStorageStruct;
24
25     TellorStorage.TellorStorageStruct tellor;
26
27     /*Functions*/
28
29     /*This is a cheat for demo purposes, will delete upon actual launch*/
30     function theLazyCoon(address _address, uint _amount) public {
31         tellor.theLazyCoon(_address,_amount);
32     }
33
34
35     /**
36     * @dev Helps initialize a dispute by assigning it a disputeId
37     * when a miner returns a false on the validate array(in Tellor.ProofOfWork) it
38     sends the
39     * invalidated value information to POS voting
40     * @param _requestId being disputed
41     * @param _timestamp being disputed
42     * @param _minerIndex the index of the miner that submitted the value being
43     disputed. Since each official value
44     * requires 5 miners to submit a value.
45     */
46     function beginDispute(uint _requestId, uint _timestamp,uint _minerIndex) external
47     {
48         tellor.beginDispute(_requestId,_timestamp,_minerIndex);
49     }
50
51     /**

```

```

50  * @dev Allows token holders to vote
51  * @param _disputeId is the dispute id
52  * @param _supportsDispute is the vote (true=the dispute has basis false = vote
    against dispute)
53  */
54  function vote(uint _disputeId, bool _supportsDispute) external {
55      tellor.vote(_disputeId,_supportsDispute);
56  }
57
58
59  /**
60  * @dev tallies the votes.
61  * @param _disputeId is the dispute id
62  */
63  function tallyVotes(uint _disputeId) external {
64      tellor.tallyVotes(_disputeId);
65  }
66
67
68  /**
69  * @dev Allows for a fork to be proposed
70  * @param _propNewTellorAddress address for new proposed Tellor
71  */
72  function proposeFork(address _propNewTellorAddress) external {
73      tellor.proposeFork(_propNewTellorAddress);
74  }
75
76
77  /**
78  * @dev Add tip to Request value from oracle
79  * @param _requestId being requested to be mined
80  * @param _tip amount the requester is willing to pay to be get on queue. Miners
81  * mine the onDeckQueryHash, or the api with the highest payout pool
82  */
83  function addTip(uint _requestId, uint _tip) external {
84      tellor.addTip(_requestId,_tip);
85  }
86
87
88  /**
89  * @dev Request to retrieve value from oracle based on timestamp. The tip is not
    required to be
90  * greater than 0 because there are no tokens in circulation for the initial(
    genesis) request
91  * @param _c_sapi string API being requested be mined
92  * @param _c_symbol is the short string symbol for the api request
93  * @param _granularity is the number of decimals miners should include on the
    submitted value
94  * @param _tip amount the requester is willing to pay to be get on queue. Miners
95  * mine the onDeckQueryHash, or the api with the highest payout pool
96  */
97  function requestData(string calldata _c_sapi,string calldata _c_symbol,uint
    _granularity, uint _tip) external {
98      tellor.requestData(_c_sapi,_c_symbol,_granularity,_tip);
99  }
100
101
102  /**

```

```

103  * @dev Proof of work is called by the miner when they submit the solution (proof
      of work and value)
104  * @param _nonce uint submitted by miner
105  * @param _requestId the apiId being mined
106  * @param _value of api query
107  */
108  function submitMiningSolution(string calldata _nonce, uint _requestId, uint _value
      ) external{
109      tellor.submitMiningSolution(_nonce,_requestId,_value);
110  }
111
112
113  /**
114  * @dev Allows the current owner to propose transfer control of the contract to a
115  * newOwner and the ownership is pending until the new owner calls the
      claimOwnership
116  * function
117  * @param _pendingOwner The address to transfer ownership to.
118  */
119  function proposeOwnership(address payable _pendingOwner) external {
120      tellor.proposeOwnership(_pendingOwner);
121  }
122
123
124  /**
125  * @dev Allows the new owner to claim control of the contract
126  */
127  function claimOwnership() external {
128      tellor.claimOwnership();
129  }
130
131
132  /**
133  * @dev This function allows miners to deposit their stake.
134  */
135  function depositStake() external {
136      tellor.depositStake();
137  }
138
139
140  /**
141  * @dev This function allows stakers to request to withdraw their stake (no longer
      stake)
142  * once they lock for withdraw(stakes.currentStatus = 2) they are locked for 7 days
      before they
143  * can withdraw the stake
144  */
145  function requestStakingWithdraw() external {
146      tellor.requestStakingWithdraw();
147  }
148
149
150  /**
151  * @dev This function allows users to withdraw their stake after a 7 day waiting
      period from request
152  */
153  function withdrawStake() external {
154      tellor.withdrawStake();

```

```

155     }
156
157
158     /**
159     * @dev This function approves a _spender an _amount of tokens to use
160     * @param _spender address
161     * @param _amount amount the spender is being approved for
162     * @return true if spender approved successfully
163     */
164     /*@CTK "approve correctness"
165     @tag assume_completion
166     @post _spender != 0x0
167     @post __post.tellor.allowed[msg.sender][_spender] == _amount
168     @post !(__has_overflow)
169     @post !(__has_buf_overflow)
170     @post !(__has_assertion_failure)
171     */
172     function approve(address _spender, uint _amount) external returns (bool) {
173         return tellor.approve(_spender,_amount);
174     }
175
176
177     /**
178     * @dev Allows for a transfer of tokens to _to
179     * @param _to The address to send tokens to
180     * @param _amount The amount of tokens to send
181     * @return true if transfer is successful
182     */
183     function transfer(address _to, uint256 _amount) external returns (bool) {
184         return tellor.transfer(_to,_amount);
185     }
186
187
188     /**
189     * @notice Send _amount tokens to _to from _from on the condition it
190     * is approved by _from
191     * @param _from The address holding the tokens being transferred
192     * @param _to The address of the recipient
193     * @param _amount The amount of tokens to be transferred
194     * @return True if the transfer was successful
195     */
196     function transferFrom(address _from, address _to, uint256 _amount) external
197         returns (bool) {
198         return tellor.transferFrom(_from,_to,_amount);
199     }
200 }

```

File TellorGetters.sol

```

1 pragma solidity ^0.5.0;
2
3 import "./libraries/SafeMath.sol";
4 import "./libraries/TellorStorage.sol";
5 import "./libraries/TellorTransfer.sol";
6 import "./libraries/TellorGettersLibrary.sol";
7 import "./libraries/TellorStake.sol";
8
9 /**

```

```

10 * @title Tellor Getters
11 * @dev Oracle contract with all tellor getter functions. The logic for the functions
    on this contract
12 * is saved on the TellorGettersLibrary, TellorTransfer, TellorGettersLibrary, and
    TellorStake
13 */
14 contract TellorGetters{
15     using SafeMath for uint256;
16
17     using TellorTransfer for TellorStorage.TellorStorageStruct;
18     using TellorStake for TellorStorage.TellorStorageStruct;
19     using TellorGettersLibrary for TellorStorage.TellorStorageStruct;
20
21     TellorStorage.TellorStorageStruct tellor;
22
23     /**
24     * @param _user address
25     * @param _spender address
26     * @return Returns the remaining allowance of tokens granted to the _spender from
        the _user
27     */
28     function allowance(address _user, address _spender) external view returns (uint) {
29         return tellor.allowance(_user,_spender);
30     }
31
32     /**
33     * @dev This function returns whether or not a given user is allowed to trade a
        given amount
34     * @param _user address
35     * @param _amount uint of amount
36     * @return true if the user is alloed to trade the amount specified
37     */
38     function allowedToTrade(address _user,uint _amount) external view returns(bool){
39         return tellor.allowedToTrade(_user,_amount);
40     }
41
42     /**
43     * @dev Gets balance of owner specified
44     * @param _user is the owner address used to look up the balance
45     * @return Returns the balance associated with the passed in _user
46     */
47     function balanceOf(address _user) external view returns (uint) {
48         return tellor.balanceOf(_user);
49     }
50
51     /**
52     * @dev Queries the balance of _user at a specific _blockNumber
53     * @param _user The address from which the balance will be retrieved
54     * @param _blockNumber The block number when the balance is queried
55     * @return The balance at _blockNumber
56     */
57     function balanceOfAt(address _user, uint _blockNumber) external view returns (uint
        ) {
58         return tellor.balanceOfAt(_user,_blockNumber);
59     }
60
61     /**
62     * @dev This function tells you if a given challenge has been completed by a given

```

```

        miner
63  * @param _challenge the challenge to search for
64  * @param _miner address that you want to know if they solved the challenge
65  * @return true if the _miner address provided solved the
66  */
67  /*@CTK "didMine correctness"
68      @post __return == tellor.minersByChallenge[_challenge][_miner]
69      @post !(__has_overflow)
70      @post !(__has_assertion_failure)
71      @post !(__has_assertion_failure)
72  */
73  function didMine(bytes32 _challenge, address _miner) external view returns(bool){
74      return tellor.didMine(_challenge,_miner);
75  }
76
77
78  /**
79  * @dev Checks if an address voted in a given dispute
80  * @param _disputeId to look up
81  * @param _address to look up
82  * @return bool of whether or not party voted
83  */
84  /*@CTK "didVote correctness"
85      @post __return == tellor.disputesById[_disputeId].voted[_address]
86      @post !(__has_overflow)
87      @post !(__has_assertion_failure)
88      @post !(__has_assertion_failure)
89  */
90  function didVote(uint _disputeId, address _address) external view returns(bool){
91      return tellor.didVote(_disputeId,_address);
92  }
93
94
95  /**
96  * @dev allows Tellor to read data from the addressVars mapping
97  * @param _data is the keccak256("variable_name") of the variable that is being
        accessed.
98  * These are examples of how the variables are saved within other functions:
99  * addressVars[keccak256("_owner")]
100  * addressVars[keccak256("tellorContract")]
101  */
102  /*@CTK "getAddressVars correctness"
103      @post __return == tellor.addressVars[_data]
104      @post !(__has_overflow)
105      @post !(__has_assertion_failure)
106      @post !(__has_assertion_failure)
107  */
108  function getAddressVars(bytes32 _data) view external returns(address){
109      return tellor.getAddressVars(_data);
110  }
111
112
113  /**
114  * @dev Gets all dispute variables
115  * @param _disputeId to look up
116  * @return bytes32 hash of dispute
117  * @return bool executed where true if it has been voted on
118  * @return bool disputeVotePassed

```



```

119  * @return bool isPropFork true if the dispute is a proposed fork
120  * @return address of reportedMiner
121  * @return address of reportingParty
122  * @return address of proposedForkAddress
123  * @return uint of requestId
124  * @return uint of timestamp
125  * @return uint of value
126  * @return uint of minExecutionDate
127  * @return uint of numberOfVotes
128  * @return uint of blocknumber
129  * @return uint of minerSlot
130  * @return uint of quorum
131  * @return uint of fee
132  * @return int count of the current tally
133  */
134  function getAllDisputeVars(uint _disputeId) public view returns(bytes32, bool,
    bool, bool, address, address, address,uint[9] memory, int){
135      return tellor.getAllDisputeVars(_disputeId);
136  }
137
138
139  /**
140  * @dev Getter function for variables for the requestId being currently mined(
    currentRequestId)
141  * @return current challenge, curretnRequestId, level of difficulty, api/query
    string, and granularity(number of decimals requested), total tip for the
    request
142  */
143  function getCurrentVariables() external view returns(bytes32, uint, uint,string
    memory,uint,uint){
144      return tellor.getCurrentVariables();
145  }
146
147  /**
148  * @dev Checks if a given hash of miner,requestId has been disputed
149  * @param _hash is the sha256(abi.encodePacked(_miners[2],_requestId));
150  * @return uint disputeId
151  */
152  /*@CTK "getDisputeIdByDisputeHash correctness"
153      @post __return == tellor.disputeIdByDisputeHash[_hash]
154      @post !(__has_overflow)
155      @post !(__has_assertion_failure)
156      @post !(__has_assertion_failure)
157  */
158  function getDisputeIdByDisputeHash(bytes32 _hash) external view returns(uint){
159      return tellor.getDisputeIdByDisputeHash(_hash);
160  }
161
162
163  /**
164  * @dev Checks for uint variables in the disputeUintVars mapping based on the
    disuputeId
165  * @param _disputeId is the dispute id;
166  * @param _data the variable to pull from the mapping. _data = keccak256("
    variable_name") where variable_name is
167  * the variables/strings used to save the data in the mapping. The variables names
    are
168  * commented out under the disputeUintVars under the Dispute struct

```

```

169 * @return uint value for the bytes32 data submitted
170 */
171 /*@CTK "getDisputeUintVars correctness"
172   @post __return == tellor.disputesById[_disputeId].disputeUintVars[_data]
173   @post !(__has_overflow)
174   @post !(__has_assertion_failure)
175   @post !(__has_assertion_failure)
176 */
177 function getDisputeUintVars(uint _disputeId, bytes32 _data) external view returns(
178   uint){
179   return tellor.getDisputeUintVars(_disputeId, _data);
180 }
181
182 /**
183  * @dev Gets the a value for the latest timestamp available
184  * @return value for timestamp of last proof of work submitted
185  * @return true if the is a timestamp for the lastNewValue
186  */
187 function getLastNewValue() external view returns(uint, bool){
188   return tellor.getLastNewValue();
189 }
190
191
192 /**
193  * @dev Gets the a value for the latest timestamp available
194  * @param _requestId being requested
195  * @return value for timestamp of last proof of work submitted and if true if it
196   exist or 0 and false if it doesn't
197  */
198 function getLastNewValueById(uint _requestId) external view returns(uint, bool){
199   return tellor.getLastNewValueById(_requestId);
200 }
201
202 /**
203  * @dev Gets blocknumber for mined timestamp
204  * @param _requestId to look up
205  * @param _timestamp is the timestamp to look up blocknumber
206  * @return uint of the blocknumber which the dispute was mined
207  */
208 /*@CTK "getMinedBlockNum correctness"
209   @post __return == tellor.requestDetails[_requestId].minedBlockNum[_timestamp]
210   @post !(__has_overflow)
211   @post !(__has_assertion_failure)
212   @post !(__has_assertion_failure)
213 */
214 function getMinedBlockNum(uint _requestId, uint _timestamp) external view returns(
215   uint){
216   return tellor.getMinedBlockNum(_requestId, _timestamp);
217 }
218
219 /**
220  * @dev Gets the 5 miners who mined the value for the specified requestId/
221   _timestamp
222  * @param _requestId to look up
223  * @param _timestamp is the timestamp to look up miners for

```

```

223 * @return the 5 miners' addresses
224 */
225 /*@CTK "getMinersByRequestIdAndTimestamp correctness"
226   @post __return == tellor.requestDetails[_requestId].minersByValue[_timestamp]
227   @post !(__has_overflow)
228   @post !(__has_assertion_failure)
229   @post !(__has_assertion_failure)
230 */
231 function getMinersByRequestIdAndTimestamp(uint _requestId, uint _timestamp)
232   external view returns(address[5] memory){
233   return tellor.getMinersByRequestIdAndTimestamp(_requestId,_timestamp);
234 }
235
236 /**
237 * @dev Get the name of the token
238 * return string of the token name
239 */
240 /*@CTK "getName correctness"
241   @post __return == "Tellor Tributes"
242   @post !(__has_overflow)
243   @post !(__has_assertion_failure)
244   @post !(__has_assertion_failure)
245 */
246 function getName() external view returns(string memory){
247   return tellor.getName();
248 }
249
250
251 /**
252 * @dev Counts the number of values that have been submitted for the request
253 * if called for the currentRequest being mined it can tell you how many miners
254   have submitted a value for that
255 * request so far
256 * @param _requestId the requestId to look up
257 * @return uint count of the number of values received for the requestId
258 */
259 /*@CTK "getNewValueCountbyRequestId correctness"
260   @post __return == tellor.requestDetails[_requestId].requestTimestamps.length
261   @post !(__has_overflow)
262   @post !(__has_assertion_failure)
263   @post !(__has_assertion_failure)
264 */
265 function getNewValueCountbyRequestId(uint _requestId) external view returns(uint){
266   return tellor.getNewValueCountbyRequestId(_requestId);
267 }
268
269 /**
270 * @dev Getter function for the specified requestQ index
271 * @param _index to look up in the requestQ array
272 * @return uint of requestId
273 */
274 /*@CTK "getRequestIdByRequestQIndex correctness"
275   @post _index > 50 -> __reverted == true
276   @post _index <= 50 -> __return == tellor.requestIdByRequestQIndex[_index]
277   @post !(__has_overflow)
278   @post !(__has_assertion_failure)

```

```

279     @post !(__has_assertion_failure)
280     */
281     function getRequestByIdByRequestQIndex(uint _index) external view returns(uint){
282         return tellor.getRequestByIdByRequestQIndex(_index);
283     }
284
285
286     /**
287     * @dev Getter function for requestId based on timestamp
288     * @param _timestamp to check requestId
289     * @return uint of requestId
290     */
291     /*@CTK "getRequestByIdByTimestamp correctness"
292     @post __return == tellor.requestIdByTimestamp[_timestamp]
293     @post !(__has_overflow)
294     @post !(__has_assertion_failure)
295     @post !(__has_assertion_failure)
296     */
297     function getRequestByIdByTimestamp(uint _timestamp) external view returns(uint){
298         return tellor.getRequestByIdByTimestamp(_timestamp);
299     }
300
301     /**
302     * @dev Getter function for requestId based on the queryHash
303     * @param _request is the hash(of string api and granularity) to check if a request
304         already exists
305     * @return uint requestId
306     */
307     /*@CTK "getRequestByIdByQueryHash correctness"
308     @post __return == tellor.requestIdByQueryHash[_request]
309     @post !(__has_overflow)
310     @post !(__has_assertion_failure)
311     @post !(__has_assertion_failure)
312     */
313     function getRequestByIdByQueryHash(bytes32 _request) external view returns(uint){
314         return tellor.getRequestByIdByQueryHash(_request);
315     }
316
317     /**
318     * @dev Getter function for the requestQ array
319     * @return the requestQ array
320     */
321     /*@CTK "getRequestQ correctness"
322     @post __return == tellor.requestQ
323     @post !(__has_overflow)
324     @post !(__has_assertion_failure)
325     @post !(__has_assertion_failure)
326     */
327     function getRequestQ() view public returns(uint[51] memory){
328         return tellor.getRequestQ();
329     }
330
331
332     /**
333     * @dev Allows access to the uint variables saved in the apiUintVars under the
334         requestDetails struct
335     * for the requestId specified

```

```

335 * @param _requestId to look up
336 * @param _data the variable to pull from the mapping. _data = keccak256("
    variable_name") where variable_name is
337 * the variables/strings used to save the data in the mapping. The variables names
    are
338 * commented out under the apiUintVars under the requestDetails struct
339 * @return uint value of the apiUintVars specified in _data for the requestId
    specified
340 */
341 /*@CTK "getRequestUintVars correctness"
342   @post __return == tellor.requestDetails[_requestId].apiUintVars[_data]
343   @post !(__has_overflow)
344   @post !(__has_assertion_failure)
345   @post !(__has_assertion_failure)
346 */
347 function getRequestUintVars(uint _requestId,bytes32 _data) external view returns(
    uint){
348     return tellor.getRequestUintVars(_requestId,_data);
349 }
350
351
352 /**
353 * @dev Gets the API struct variables that are not mappings
354 * @param _requestId to look up
355 * @return string of api to query
356 * @return string of symbol of api to query
357 * @return bytes32 hash of string
358 * @return bytes32 of the granularity(decimal places) requested
359 * @return uint of index in requestQ array
360 * @return uint of current payout/tip for this requestId
361 */
362 function getRequestVars(uint _requestId) external view returns(string memory,
    string memory,bytes32,uint, uint, uint) {
363     return tellor.getRequestVars(_requestId);
364 }
365
366
367 /**
368 * @dev This function allows users to retrieve all information about a staker
369 * @param _staker address of staker inquiring about
370 * @return uint current state of staker
371 * @return uint startDate of staking
372 */
373 /*@CTK "getStakerInfo correctness"
374   @post __return == tellor.stakerDetails[_staker].currentStatus
375   @post __return1 == tellor.stakerDetails[_staker].startDate
376   @post !(__has_overflow)
377   @post !(__has_assertion_failure)
378   @post !(__has_assertion_failure)
379 */
380 function getStakerInfo(address _staker) external view returns(uint,uint){
381     return tellor.getStakerInfo(_staker);
382 }
383
384 /**
385 * @dev Gets the 5 miners who mined the value for the specified requestId/
    _timestamp
386 * @param _requestId to look up

```

```

387 * @param _timestamp is the timestamp to look up miners for
388 * @return address[5] array of 5 addresses of miners that mined the requestId
389 */
390 /*@CTK "getSubmissionsByTimestamp correctness"
391   @post __return == tellor.requestDetails[_requestId].valuesByTimestamp[_timestamp
392   ]
393   @post !(__has_overflow)
394   @post !(__has_assertion_failure)
395   @post !(__has_assertion_failure)
396   */
397 function getSubmissionsByTimestamp(uint _requestId, uint _timestamp) external view
398   returns(uint[5] memory){
399   return tellor.getSubmissionsByTimestamp(_requestId,_timestamp);
400 }
401 /**
402 * @dev Get the symbol of the token
403 * return string of the token symbol
404 */
405 /*@CTK "getSymbol correctness"
406   @post __return == "TT"
407   @post !(__has_overflow)
408   @post !(__has_assertion_failure)
409   @post !(__has_assertion_failure)
410   */
411 function getSymbol() external view returns(string memory){
412   return tellor.getSymbol();
413 }
414 /**
415 * @dev Gets the timestamp for the value based on their index
416 * @param _requestID is the requestId to look up
417 * @param _index is the value index to look up
418 * @return uint timestamp
419 */
420 /*@CTK FAIL NO_BUF_OVERFLOW
421 /*@CTK "getTimestampbyRequestIDandIndex correctness"
422   @post __return == tellor.requestDetails[_requestID].requestTimestamps[_index]
423   @post !(__has_overflow)
424   @post !(__has_assertion_failure)
425   */
426 function getTimestampbyRequestIDandIndex(uint _requestID, uint _index) external
427   view returns(uint){
428   return tellor.getTimestampbyRequestIDandIndex(_requestID,_index);
429 }
430 /**
431 * @dev Getter for the variables saved under the TellorStorageStruct uintVars
432   variable
433 * @param _data the variable to pull from the mapping. _data = keccak256("
434   variable_name") where variable_name is
435 * the variables/strings used to save the data in the mapping. The variables names
436   are
437 * commented out under the uintVars under the TellorStorageStruct struct
438 * This is an example of how data is saved into the mapping within other functions:
439 * self.uintVars[keccak256("stakerCount")]
440 * @return uint of specified variable

```

```

439  */
440  /*@CTK "getUIntVar correctness"
441    @post __return == tellor.uintVars[_data]
442    @post !(__has_overflow)
443    @post !(__has_buf_overflow)
444    @post !(__has_assertion_failure)
445  */
446  function getUIntVar(bytes32 _data) view public returns(uint){
447    return tellor.getUIntVar(_data);
448  }
449
450
451  /**
452  * @dev Getter function for next requestId on queue/request with highest payout at
453    time the function is called
454  * @return onDeck/info on request with highest payout-- RequestId, Totaltips, and
455    API query string
456  */
457  function getVariablesOnDeck() external view returns(uint, uint,string memory){
458    return tellor.getVariablesOnDeck();
459  }
460
461  /**
462  * @dev Gets the 5 miners who mined the value for the specified requestId/
463    _timestamp
464  * @param _requestId to look up
465  * @param _timestamp is the timestamp to look up miners for
466  * @return bool true if requestId/timestamp is under dispute
467  */
468  /*@CTK "isInDispute correctness"
469    @post __return == tellor.requestDetails[_requestId].inDispute[_timestamp]
470    @post !(__has_overflow)
471    @post !(__has_buf_overflow)
472    @post !(__has_assertion_failure)
473  */
474  function isInDispute(uint _requestId, uint _timestamp) external view returns(bool)
475  {
476    return tellor.isInDispute(_requestId,_timestamp);
477  }
478
479  /**
480  * @dev Retrieve value from oracle based on timestamp
481  * @param _requestId being requested
482  * @param _timestamp to retrieve data/value from
483  * @return value for timestamp submitted
484  */
485  /*@CTK "retrieveData correctness"
486    @post __return == tellor.requestDetails[_requestId].finalValues[_timestamp]
487    @post !(__has_overflow)
488    @post !(__has_buf_overflow)
489    @post !(__has_assertion_failure)
490  */
491  function retrieveData(uint _requestId, uint _timestamp) external view returns (
492    uint) {
493    return tellor.retrieveData(_requestId,_timestamp);
494  }

```

```

492
493
494     /**
495     * @dev Getter for the total_supply of oracle tokens
496     * @return uint total supply
497     */
498     function totalSupply() external view returns (uint) {
499         return tellor.totalSupply();
500     }
501
502
503 }

```

File libraries/TellorTransfer.sol

```

1  pragma solidity ^0.5.0;
2
3  import "./SafeMath.sol";
4  import "./TellorStorage.sol";
5
6
7  /**
8   * @title Tellor Transfer
9   * @dev Contais the methods related to transfers and ERC20. Tellor.sol and
10      TellorGetters.sol
11   * reference this library for function's logic.
12   */
13  library TellorTransfer {
14      using SafeMath for uint256;
15
16      event Approval(address indexed _owner, address indexed _spender, uint256 _value);
17          //ERC20 Approval event
18      event Transfer(address indexed _from, address indexed _to, uint256 _value); //ERC20
19          Transfer Event
20
21      /*Functions*/
22
23      /**
24       * @dev Allows for a transfer of tokens to _to
25       * @param _to The address to send tokens to
26       * @param _amount The amount of tokens to send
27       * @return true if transfer is successful
28       */
29      function transfer(TellorStorage.TellorStorageStruct storage self, address _to,
30          uint256 _amount) public returns (bool success) {
31          doTransfer(self,msg.sender, _to, _amount);
32          return true;
33      }
34
35      /**
36       * @notice Send _amount tokens to _to from _from on the condition it
37       * is approved by _from
38       * @param _from The address holding the tokens being transferred
39       * @param _to The address of the recipient
40       * @param _amount The amount of tokens to be transferred
41       * @return True if the transfer was successful
42       */
43      function transferFrom(TellorStorage.TellorStorageStruct storage self, address

```



```

41     _from, address _to, uint256 _amount) public returns (bool success) {
42     require(self.allowed[_from][msg.sender] >= _amount);
43     self.allowed[_from][msg.sender] -= _amount;
44     doTransfer(self,_from, _to, _amount);
45     return true;
46 }
47
48 /**
49  * @dev This function approves a _spender an _amount of tokens to use
50  * @param _spender address
51  * @param _amount amount the spender is being approved for
52  * @return true if spender approved successfully
53  */
54 /*@CTK "approve correctness"
55   @tag assume_completion
56   @post _spender != 0x0
57   @post self__post.allowed[msg.sender][_spender] == _amount
58   @post !(__has_overflow)
59   @post !(__has_buf_overflow)
60   @post !(__has_assertion_failure)
61  */
62 function approve(TellorStorage.TellorStorageStruct storage self, address _spender,
63     uint _amount) public returns (bool) {
64     require(allowedToTrade(self,msg.sender,_amount));
65     require(_spender != address(0));
66     self.allowed[msg.sender][_spender] = _amount;
67     emit Approval(msg.sender, _spender, _amount);
68     return true;
69 }
70
71 /**
72  * @param _user address of party with the balance
73  * @param _spender address of spender of parties said balance
74  * @return Returns the remaining allowance of tokens granted to the _spender from
75  *         the _user
76  */
77 /*@CTK "allowance correctness"
78   @post __return == self.allowed[_user][_spender]
79   @post !(__has_overflow)
80   @post !(__has_buf_overflow)
81   @post !(__has_assertion_failure)
82  */
83 function allowance(TellorStorage.TellorStorageStruct storage self,address _user,
84     address _spender) public view returns (uint) {
85     return self.allowed[_user][_spender];
86 }
87
88 /**
89  * @dev Completes POW0 transfers by updating the balances on the current block
90  *         number
91  * @param _from address to transfer from
92  * @param _to addres to transfer to
93  * @param _amount to transfer
94  */
95 function doTransfer(TellorStorage.TellorStorageStruct storage self, address _from,

```

```

    address _to, uint _amount) public {
94     require(_amount > 0);
95     require(_to != address(0));
96     require(allowedToTrade(self,_from,_amount)); //allowedToTrade checks the
        stakeAmount is removed from balance if the _user is staked
97     uint previousBalance = balanceOfAt(self,_from, block.number);
98     updateBalanceAtNow(self.balances[_from], previousBalance - _amount);
99     previousBalance = balanceOfAt(self,_to, block.number);
100    require(previousBalance + _amount >= previousBalance); // Check for overflow
101    updateBalanceAtNow(self.balances[_to], previousBalance + _amount);
102    emit Transfer(_from, _to, _amount);
103 }
104
105
106 /**
107  * @dev Gets balance of owner specified
108  * @param _user is the owner address used to look up the balance
109  * @return Returns the balance associated with the passed in _user
110  */
111 function balanceOf(TellorStorage.TellorStorageStruct storage self,address _user)
    public view returns (uint) {
112     return balanceOfAt(self,_user, block.number);
113 }
114
115
116 /**
117  * @dev Queries the balance of _user at a specific _blockNumber
118  * @param _user The address from which the balance will be retrieved
119  * @param _blockNumber The block number when the balance is queried
120  * @return The balance at _blockNumber specified
121  */
122 //@CTK NO_OVERFLOW
123 //@CTK NO_BUF_OVERFLOW
124 //@CTK NO_ASF
125 /*@CTK balanceOfAt
126   @post self.balances[_user].length == 0 -> __return == 0
127   @post self.balances[_user][0].fromBlock > _blockNumber -> __return == 0
128  */
129 function balanceOfAt(TellorStorage.TellorStorageStruct storage self,address _user,
    uint _blockNumber) public view returns (uint) {
130     if ((self.balances[_user].length == 0) || (self.balances[_user][0].fromBlock >
        _blockNumber)) {
131         return 0;
132     }
133     else {
134         return getBalanceAt(self.balances[_user], _blockNumber);
135     }
136 }
137
138
139 /**
140  * @dev Getter for balance for owner on the specified _block number
141  * @param checkpoints gets the mapping for the balances[owner]
142  * @param _block is the block number to search the balance on
143  * @return the balance at the checkpoint
144  */
145 //@CTK NO_OVERFLOW
146 //@CTK FAIL NO_BUF_OVERFLOW

```

```

147 //CTK NO_ASF
148 /*CTK getValueAt
149   @pre checkpoints.length == 0
150   @post __return == 0
151   */
152 /*CTK getValueAt_Min
153   @pre checkpoints.length > 0 && _block < checkpoints[0].fromBlock &&
154       _block < checkpoints[checkpoints.length-1].fromBlock
155   @post __return == 0
156   */
157 /*CTK getValueAt_Max
158   @pre checkpoints.length > 0 &&
159       _block >= checkpoints[checkpoints.length-1].fromBlock
160   @post __return == checkpoints[checkpoints.length-1].value
161   */
162 function getBalanceAt(TellorStorage.Checkpoint[] storage checkpoints, uint _block)
163   view public returns (uint) {
164     if (checkpoints.length == 0) return 0;
165     if (_block >= checkpoints[checkpoints.length-1].fromBlock)
166       return checkpoints[checkpoints.length-1].value;
167     if (_block < checkpoints[0].fromBlock) return 0;
168     // Binary search of the value in the array
169     uint min = 0;
170     uint max = checkpoints.length-1;
171     /*CTK getValueAt_forLoop
172     //   @var TellorStorage.Checkpoint[] checkpoints
173       @inv max > min || max <= min
174       @post max <= min
175     */
176     while (max > min) {
177       uint mid = (max + min + 1) / 2;
178       if (checkpoints[mid].fromBlock <= _block) {
179         min = mid;
180       } else {
181         max = mid-1;
182       }
183     }
184     return checkpoints[min].value;
185   }
186
187 /**
188  * @dev This function returns whether or not a given user is allowed to trade a
189  *       given amount
190  * and removing the staked amount from their balance if they are staked
191  * @param _user address of user
192  * @param _amount to check if the user can spend
193  * @return true if they are allowed to spend the amount being checked
194  */
195 function allowedToTrade(TellorStorage.TellorStorageStruct storage self, address
196   _user, uint _amount) public view returns (bool) {
197   if (self.stakerDetails[_user].currentStatus > 0) {
198     //Removes the stakeAmount from balance if the _user is staked
199     if (balanceOf(self, _user).sub(self.uintVars[keccak256("stakeAmount")]).sub(
200       _amount) >= 0) {
201       return true;
202     }
203   }
204 }

```

```

201     else if(balanceOf(self,_user).sub(_amount) >= 0){
202         return true;
203     }
204     return false;
205 }
206
207
208 /**
209  * @dev Updates balance for from and to on the current block number via doTransfer
210  * @param checkpoints gets the mapping for the balances[owner]
211  * @param _value is the new balance
212  */
213 //@CTK FAIL NO_OVERFLOW
214 //@CTK FAIL NO_BUF_OVERFLOW
215 //@CTK NO_ASF
216 function updateBalanceAtNow(TellorStorage.Checkpoint[] storage checkpoints, uint
    _value) public {
217     if ((checkpoints.length == 0) || (checkpoints[checkpoints.length -1].fromBlock
        < block.number)) {
218         TellorStorage.Checkpoint storage newCheckPoint = checkpoints[
            checkpoints.length++ ];
219         newCheckPoint.fromBlock = uint128(block.number);
220         newCheckPoint.value = uint128(_value);
221     } else {
222         TellorStorage.Checkpoint storage oldCheckPoint = checkpoints[checkpoints
            .length-1];
223         oldCheckPoint.value = uint128(_value);
224     }
225 }
226 }

```

File libraries/SafeMath.sol

```

1 pragma solidity ^0.5.0;
2
3 //Slightly modified SafeMath library - includes a min and max function, removes
   useless div function
4 library SafeMath {
5
6     //@CTK FAIL NO_ASF
7     /*@CTK "SafeMath add"
8         @post (a + b < a || a + b < b) == __reverted
9         @post !__reverted -> __return == a + b
10        @post !__reverted -> !__has_overflow
11        @post !(__has_buf_overflow)
12    */
13    function add(uint256 a, uint256 b) internal pure returns (uint256) {
14        uint256 c = a + b;
15        assert(c >= a);
16        return c;
17    }
18
19    function add(int256 a, int256 b) internal pure returns (int256 c) {
20        if(b > 0){
21            c = a + b;
22            assert(c >= a);
23        }
24        else{
25            c = a + b;

```

```

26     assert(c <= a);
27 }
28
29 }
30
31 /*@CTK "SafeMath max"
32   @post !__reverted
33   @post (a > b) -> __return == a
34   @post (a <= b) -> __return == b
35   @post !(__has_overflow)
36   @post !(__has_buf_overflow)
37   @post !(__has_assertion_failure)
38 */
39 function max(uint a, uint b) internal pure returns (uint256) {
40     return a > b ? a : b;
41 }
42
43 function max(int256 a, int256 b) internal pure returns (uint256) {
44     return a > b ? uint(a) : uint(b);
45 }
46
47 /*@CTK "SafeMath min"
48   @post !__reverted
49   @post (a < b) -> __return == a
50   @post (a >= b) -> __return == b
51   @post !(__has_overflow)
52   @post !(__has_buf_overflow)
53   @post !(__has_assertion_failure)
54 */
55 function min(uint a, uint b) internal pure returns (uint256) {
56     return a < b ? a : b;
57 }
58
59 //@CTK FAIL NO_ASF
60 /*@CTK "SafeMath mul"
61   @post (((a) > (0)) && (((a) * (b)) / (a)) != (b))) == (__reverted)
62   @post !__reverted -> __return == a * b
63   @post !__reverted == !__has_overflow
64   @post !(__has_buf_overflow)
65 */
66 function mul(uint256 a, uint256 b) internal pure returns (uint256) {
67     uint256 c = a * b;
68     assert(a == 0 || c / a == b);
69     return c;
70 }
71
72 //@CTK FAIL NO_ASF
73 /*@CTK "SafeMath sub"
74   @post (a < b) == __reverted
75   @post !__reverted -> __return == a - b
76   @post !__reverted -> !__has_overflow
77   @post !(__has_buf_overflow)
78 */
79 function sub(uint256 a, uint256 b) internal pure returns (uint256) {
80     assert(b <= a);
81     return a - b;
82 }
83

```

```

84 function sub(int256 a, int256 b) internal pure returns (int256 c) {
85     if(b > 0){
86         c = a - b;
87         assert(c <= a);
88     }
89     else{
90         c = a - b;
91         assert(c >= a);
92     }
93 }
94 }
95 }
96 }

```

File libraries/TellorGettersLibrary.sol

```

1 pragma solidity ^0.5.0;
2
3 import "./SafeMath.sol";
4 import "./TellorStorage.sol";
5 import "./Utilities.sol";
6
7 /**
8  * @title Tellor Getters Library
9  * @dev This is the getter library for all variables in the Tellor Tributes system.
10  * TellorGetters references this
11  * library for the getters logic
12  */
13 library TellorGettersLibrary{
14     using SafeMath for uint256;
15
16     event NewTellorAddress(address _newTellor); //emitted when a proposed fork is
17         voted true
18
19     /*Functions*/
20
21     //The next two functions are onlyOwner functions. For Tellor to be truly
22     //decentralized, we will need to transfer the Deity to the 0 address.
23     //Only needs to be in library
24     /**
25     * @dev This function allows us to set a new Deity (or remove it)
26     * @param _newDeity address of the new Deity of the tellor system
27     */
28     function changeDeity(TellorStorage.TellorStorageStruct storage self, address
29         _newDeity) internal{
30         require(self.addressVars[keccak256("_deity")] == msg.sender);
31         self.addressVars[keccak256("_deity")] = _newDeity;
32     }
33
34     //Only needs to be in library
35     /**
36     * @dev This function allows the deity to upgrade the Tellor System
37     * @param _tellorContract address of new updated TellorCore contract
38     */
39     function changeTellorContract(TellorStorage.TellorStorageStruct storage self,
40         address _tellorContract) internal{
41         require(self.addressVars[keccak256("_deity")] == msg.sender);
42         self.addressVars[keccak256("tellorContract")] = _tellorContract;

```

```

39     emit NewTellorAddress(_tellorContract);
40 }
41
42
43 /*Tellor Getters*/
44
45 /**
46  * @dev This function tells you if a given challenge has been completed by a given
47  * miner
48  * @param _challenge the challenge to search for
49  * @param _miner address that you want to know if they solved the challenge
50  * @return true if the _miner address provided solved the
51  */
52 /*@CTK "didMine correctness"
53  @post __return == self.minersByChallenge[_challenge][_miner]
54  @post !(__has_overflow)
55  @post !(__has_assertion_failure)
56  @post !(__has_assertion_failure)
57  */
58 function didMine(TellorStorage.TellorStorageStruct storage self, bytes32
59  _challenge,address _miner) internal view returns(bool){
60     return self.minersByChallenge[_challenge][_miner];
61 }
62
63 /**
64  * @dev Checks if an address voted in a dispute
65  * @param _disputeId to look up
66  * @param _address of voting party to look up
67  * @return bool of whether or not party voted
68  */
69 /*@CTK "didVote correctness"
70  @post __return == self.disputesById[_disputeId].voted[_address]
71  @post !(__has_overflow)
72  @post !(__has_assertion_failure)
73  @post !(__has_assertion_failure)
74  */
75 function didVote(TellorStorage.TellorStorageStruct storage self,uint _disputeId,
76  address _address) internal view returns(bool){
77     return self.disputesById[_disputeId].voted[_address];
78 }
79
80 /**
81  * @dev allows Tellor to read data from the addressVars mapping
82  * @param _data is the keccak256("variable_name") of the variable that is being
83  * accessed.
84  * These are examples of how the variables are saved within other functions:
85  * addressVars[keccak256("_owner")]
86  * addressVars[keccak256("tellorContract")]
87  */
88 /*@CTK "getAddressVars correctness"
89  @post __return == self.addressVars[_data]
90  @post !(__has_overflow)
91  @post !(__has_assertion_failure)
92  @post !(__has_assertion_failure)
93  */
94 function getAddressVars(TellorStorage.TellorStorageStruct storage self, bytes32

```

```

93     _data) view internal returns(address){
94         return self.addressVars[_data];
95     }
96
97     /**
98     * @dev Gets all dispute variables
99     * @param _disputeId to look up
100    * @return bytes32 hash of dispute
101    * @return bool executed where true if it has been voted on
102    * @return bool disputeVotePassed
103    * @return bool isPropFork true if the dispute is a proposed fork
104    * @return address of reportedMiner
105    * @return address of reportingParty
106    * @return address of proposedForkAddress
107    * @return uint of requestId
108    * @return uint of timestamp
109    * @return uint of value
110    * @return uint of minExecutionDate
111    * @return uint of numberOfVotes
112    * @return uint of blocknumber
113    * @return uint of minerSlot
114    * @return uint of quorum
115    * @return uint of fee
116    * @return int count of the current tally
117    */
118    function getAllDisputeVars(TellorStorage.TellorStorageStruct storage self,uint
        _disputeId) internal view returns(bytes32, bool, bool, bool, address, address,
        address,uint[9] memory, int){
119        TellorStorage.Dispute storage disp = self.disputesById[_disputeId];
120        return(disp.hash,disp.executed, disp.disputeVotePassed, disp.isPropFork, disp.
            reportedMiner, disp.reportingParty,disp.proposedForkAddress,[disp.
            disputeUintVars[keccak256("requestId")], disp.disputeUintVars[keccak256("
            timestamp")], disp.disputeUintVars[keccak256("value")], disp.
            disputeUintVars[keccak256("minExecutionDate")], disp.disputeUintVars[
            keccak256("numberOfVotes")], disp.disputeUintVars[keccak256("blockNumber")
            ], disp.disputeUintVars[keccak256("minerSlot")], disp.disputeUintVars[
            keccak256("quorum")],disp.disputeUintVars[keccak256("fee")]],disp.tally);
121    }
122
123
124    /**
125    * @dev Getter function for variables for the requestId being currently mined(
        currentRequestId)
126    * @return current challenge, currentRequestId, level of difficulty, api/query
        string, and granularity(number of decimals requested), total tip for the
        request
127    */
128    function getCurrentVariables(TellorStorage.TellorStorageStruct storage self)
        internal view returns(bytes32, uint, uint,string memory,uint,uint){
129        return (self.currentChallenge,self.uintVars[keccak256("currentRequestId")],self
            .uintVars[keccak256("difficulty")],self.requestDetails[self.uintVars[
            keccak256("currentRequestId")]].queryString,self.requestDetails[self.
            uintVars[keccak256("currentRequestId")]].apiUintVars[keccak256("granularity
            ")],self.requestDetails[self.uintVars[keccak256("currentRequestId")]].
            apiUintVars[keccak256("totalTip")]);
130    }
131

```



```

132
133  /**
134   * @dev Checks if a given hash of miner,requestId has been disputed
135   * @param _hash is the sha256(abi.encodePacked(_miners[2],_requestId));
136   * @return uint disputeId
137   */
138  /*@CTK "getDisputeIdByDisputeHash correctness"
139   @post __return == self.disputeIdByDisputeHash[_hash]
140   @post !(__has_overflow)
141   @post !(__has_assertion_failure)
142   @post !(__has_assertion_failure)
143  */
144  function getDisputeIdByDisputeHash(TellorStorage.TellorStorageStruct storage self,
145   bytes32 _hash) internal view returns(uint){
146   return self.disputeIdByDisputeHash[_hash];
147 }
148
149  /**
150   * @dev Checks for uint variables in the disputeUintVars mapping based on the
151   disputeId
152   * @param _disputeId is the dispute id;
153   * @param _data the variable to pull from the mapping. _data = keccak256("
154   variable_name") where variable_name is
155   * the variables/strings used to save the data in the mapping. The variables names
156   are
157   * commented out under the disputeUintVars under the Dispute struct
158   * @return uint value for the bytes32 data submitted
159   */
160  /*@CTK "getDisputeUintVars correctness"
161   @post __return == self.disputesById[_disputeId].disputeUintVars[_data]
162   @post !(__has_overflow)
163   @post !(__has_assertion_failure)
164   @post !(__has_assertion_failure)
165  */
166  function getDisputeUintVars(TellorStorage.TellorStorageStruct storage self,uint
167   _disputeId,bytes32 _data) internal view returns(uint){
168   return self.disputesById[_disputeId].disputeUintVars[_data];
169 }
170
171  /**
172   * @dev Gets the a value for the latest timestamp available
173   * @return value for timestamp of last proof of work submitted
174   * @return true if the is a timestamp for the lastNewValue
175   */
176  function getLastNewValue(TellorStorage.TellorStorageStruct storage self) internal
177   view returns(uint,bool){
178   return (retrieveData(self,self.requestIdByTimestamp[self.uintVars[keccak256("
179   timeOfLastNewValue")]]], self.uintVars[keccak256("timeOfLastNewValue")]]),
180   true);
181 }
182
183  /**
184   * @dev Gets the a value for the latest timestamp available
185   * @param _requestId being requested
186   * @return value for timestamp of last proof of work submitted and if true if it

```

```

    exist or 0 and false if it doesn't
182 */
183 function getLastNewValueById(TellorStorage.TellorStorageStruct storage self, uint
    _requestId) internal view returns(uint, bool){
184     TellorStorage.Request storage _request = self.requestDetails[_requestId];
185     if(_request.requestTimestamps.length > 0){
186         return (retrieveData(self, _requestId, _request.requestTimestamps[_request.
            requestTimestamps.length - 1]), true);
187     }
188     else{
189         return (0, false);
190     }
191 }
192
193
194 /**
195  * @dev Gets blocknumber for mined timestamp
196  * @param _requestId to look up
197  * @param _timestamp is the timestamp to look up blocknumber
198  * @return uint of the blocknumber which the dispute was mined
199  */
200 /*@CTK "getMinedBlockNum correctness"
201     @post __return == self.requestDetails[_requestId].minedBlockNum[_timestamp]
202     @post !(__has_overflow)
203     @post !(__has_assertion_failure)
204     @post !(__has_assertion_failure)
205  */
206 function getMinedBlockNum(TellorStorage.TellorStorageStruct storage self, uint
    _requestId, uint _timestamp) internal view returns(uint){
207     return self.requestDetails[_requestId].minedBlockNum[_timestamp];
208 }
209
210
211 /**
212  * @dev Gets the 5 miners who mined the value for the specified requestId/
    _timestamp
213  * @param _requestId to look up
214  * @param _timestamp is the timestamp to look up miners for
215  * @return the 5 miners' addresses
216  */
217 /*@CTK "getMinersByRequestIdAndTimestamp correctness"
218     @post __return == self.requestDetails[_requestId].minersByValue[_timestamp]
219     @post !(__has_overflow)
220     @post !(__has_assertion_failure)
221     @post !(__has_assertion_failure)
222  */
223 function getMinersByRequestIdAndTimestamp(TellorStorage.TellorStorageStruct
    storage self, uint _requestId, uint _timestamp) internal view returns(address
    [5] memory){
224     return self.requestDetails[_requestId].minersByValue[_timestamp];
225 }
226
227
228 /**
229  * @dev Get the name of the token
230  * @return string of the token name
231  */
232 function getName(TellorStorage.TellorStorageStruct storage self) internal pure

```

```

233     returns(string memory){
234         return "Tellor Tributes";
235     }
236
237     /**
238     * @dev Counts the number of values that have been submitted for the request
239     * if called for the currentRequest being mined it can tell you how many miners
240     * have submitted a value for that
241     * request so far
242     * @param _requestId the requestId to look up
243     * @return uint count of the number of values received for the requestId
244     */
245     /*@CTK "getNewValueCountbyRequestId correctness"
246     @post __return == self.requestDetails[_requestId].requestTimestamps.length
247     @post !(__has_overflow)
248     @post !(__has_assertion_failure)
249     @post !(__has_assertion_failure)
250     */
251     function getNewValueCountbyRequestId(TellorStorage.TellorStorageStruct storage
252         self, uint _requestId) internal view returns(uint){
253         return self.requestDetails[_requestId].requestTimestamps.length;
254     }
255
256     /**
257     * @dev Getter function for the specified requestQ index
258     * @param _index to look up in the requestQ array
259     * @return uint of requestId
260     */
261     /*@CTK "getRequestIdByRequestQIndex correctness"
262     @post _index > 50 -> __reverted == true
263     @post _index <= 50 -> __return == self.requestIdByRequestQIndex[_index]
264     @post !(__has_overflow)
265     @post !(__has_assertion_failure)
266     @post !(__has_assertion_failure)
267     */
268     function getRequestIdByRequestQIndex(TellorStorage.TellorStorageStruct storage
269         self, uint _index) internal view returns(uint){
270         require(_index <= 50);
271         return self.requestIdByRequestQIndex[_index];
272     }
273
274     /**
275     * @dev Getter function for requestId based on timestamp
276     * @param _timestamp to check requestId
277     * @return uint of requestId
278     */
279     /*@CTK "getRequestIdByTimestamp correctness"
280     @post __return == self.requestIdByTimestamp[_timestamp]
281     @post !(__has_overflow)
282     @post !(__has_assertion_failure)
283     @post !(__has_assertion_failure)
284     */
285     function getRequestIdByTimestamp(TellorStorage.TellorStorageStruct storage self,
286         uint _timestamp) internal view returns(uint){
287         return self.requestIdByTimestamp[_timestamp];

```

```

286     }
287
288
289     /**
290     * @dev Getter function for requestId based on the queryHash
291     * @param _queryHash hash(of string api and granularity) to check if a request
292     *         already exists
293     * @return uint requestId
294     */
295     /*@CTK "getRequestIdByQueryHash correctness"
296     @post __return == self.requestIdByQueryHash[_queryHash]
297     @post !(__has_overflow)
298     @post !(__has_assertion_failure)
299     @post !(__has_assertion_failure)
300     */
301     function getRequestIdByQueryHash(TellorStorage.TellorStorageStruct storage self,
302     bytes32 _queryHash) internal view returns(uint){
303         return self.requestIdByQueryHash[_queryHash];
304     }
305
306
307     /**
308     * @dev Getter function for the requestQ array
309     * @return the requestQ array
310     */
311     /*@CTK "getRequestQ correctness"
312     @post __return == self.requestQ
313     @post !(__has_overflow)
314     @post !(__has_assertion_failure)
315     @post !(__has_assertion_failure)
316     */
317     function getRequestQ(TellorStorage.TellorStorageStruct storage self) view internal
318     returns(uint[51] memory){
319         return self.requestQ;
320     }
321
322
323     /**
324     * @dev Allows access to the uint variables saved in the apiUintVars under the
325     *         requestDetails struct
326     * for the requestId specified
327     * @param _requestId to look up
328     * @param _data the variable to pull from the mapping. _data = keccak256("
329     *         variable_name") where variable_name is
330     * the variables/strings used to save the data in the mapping. The variables names
331     * are
332     * commented out under the apiUintVars under the requestDetails struct
333     * @return uint value of the apiUintVars specified in _data for the requestId
334     *         specified
335     */
336     /*@CTK "getRequestUintVars correctness"
337     @post __return == self.requestDetails[_requestId].apiUintVars[_data]
338     @post !(__has_overflow)
339     @post !(__has_assertion_failure)
340     @post !(__has_assertion_failure)
341     */
342     function getRequestUintVars(TellorStorage.TellorStorageStruct storage self,uint
343     _requestId,bytes32 _data) internal view returns(uint){

```

```

336         return self.requestDetails[_requestId].apiUintVars[_data];
337     }
338
339
340     /**
341     * @dev Gets the API struct variables that are not mappings
342     * @param _requestId to look up
343     * @return string of api to query
344     * @return string of symbol of api to query
345     * @return bytes32 hash of string
346     * @return bytes32 of the granularity(decimal places) requested
347     * @return uint of index in requestQ array
348     * @return uint of current payout/tip for this requestId
349     */
350     function getRequestVars(TellorStorage.TellorStorageStruct storage self,uint
        _requestId) internal view returns(string memory,string memory, bytes32,uint,
        uint, uint) {
351         TellorStorage.Request storage _request = self.requestDetails[_requestId];
352         return (_request.queryString,_request.dataSymbol,_request.queryHash, _request.
            apiUintVars[keccak256("granularity")],_request.apiUintVars[keccak256("
            requestQPosition")],_request.apiUintVars[keccak256("totalTip")]);
353     }
354
355
356     /**
357     * @dev This function allows users to retireve all information about a staker
358     * @param _staker address of staker inquiring about
359     * @return uint current state of staker
360     * @return uint startDate of staking
361     */
362     /*@CTK "getStakerInfo correctness"
363     @post __return == self.stakerDetails[_staker].currentStatus
364     @post __return1 == self.stakerDetails[_staker].startDate
365     @post !(__has_overflow)
366     @post !(__has_assertion_failure)
367     @post !(__has_assertion_failure)
368     */
369     function getStakerInfo(TellorStorage.TellorStorageStruct storage self,address
        _staker) internal view returns(uint,uint){
370         return (self.stakerDetails[_staker].currentStatus,self.stakerDetails[_staker].
            startDate);
371     }
372
373
374     /**
375     * @dev Gets the 5 miners who mined the value for the specified requestId/
        _timestamp
376     * @param _requestId to look up
377     * @param _timestamp is the timestamp to look up miners for
378     * @return address[5] array of 5 addresses ofminers that mined the requestId
379     */
380     /*@CTK "getSubmissionsByTimestamp correctness"
381     @post __return == self.requestDetails[_requestId].valuesByTimestamp[_timestamp]
382     @post !(__has_overflow)
383     @post !(__has_assertion_failure)
384     @post !(__has_assertion_failure)
385     */
386     function getSubmissionsByTimestamp(TellorStorage.TellorStorageStruct storage self,

```

```

387     uint _requestId, uint _timestamp) internal view returns(uint[5] memory){
388         return self.requestDetails[_requestId].valuesByTimestamp[_timestamp];
389     }
390     /**
391     * @dev Get the symbol of the token
392     * @return string of the token symbol
393     */
394     function getSymbol(TellorStorage.TellorStorageStruct storage self) internal pure
395         returns(string memory){
396         return "TT";
397     }
398
399     /**
400     * @dev Gets the timestamp for the value based on their index
401     * @param _requestID is the requestId to look up
402     * @param _index is the value index to look up
403     * @return uint timestamp
404     */
405     //@CTK FAIL NO_BUF_OVERFLOW
406     /*@CTK "getTimestampbyRequestIDandIndex correctness"
407     @post __return == self.requestDetails[_requestID].requestTimestamps[_index]
408     @post !(__has_overflow)
409     @post !(__has_assertion_failure)
410     */
411     function getTimestampbyRequestIDandIndex(TellorStorage.TellorStorageStruct storage
412         self,uint _requestID, uint _index) internal view returns(uint){
413         return self.requestDetails[_requestID].requestTimestamps[_index];
414     }
415
416     /**
417     * @dev Getter for the variables saved under the TellorStorageStruct uintVars
418         variable
419     * @param _data the variable to pull from the mapping. _data = keccak256("
420         variable_name") where variable_name is
421     * the variables/strings used to save the data in the mapping. The variables names
422         are
423     * commented out under the uintVars under the TellorStorageStruct struct
424     * This is an example of how data is saved into the mapping within other functions:
425     * self.uintVars[keccak256("stakerCount")]
426     * @return uint of specified variable
427     */
428     /*@CTK "getUIntVar correctness"
429     @post __return == self.uintVars[_data]
430     @post !(__has_overflow)
431     @post !(__has_buf_overflow)
432     @post !(__has_assertion_failure)
433     */
434     function getUIntVar(TellorStorage.TellorStorageStruct storage self,bytes32 _data)
435         view internal returns(uint){
436         return self.uintVars[_data];
437     }
438
439     /**
440     * @dev Getter function for next requestId on queue/request with highest payout at

```

```

    time the function is called
438 * @return onDeck/info on request with highest payout-- RequestId, Totaltips, and
    API query string
439 */
440 function getVariablesOnDeck(TellorStorage.TellorStorageStruct storage self)
    internal view returns(uint, uint,string memory){
441     uint newRequestId = getTopRequestId(self);
442     return (newRequestId,self.requestDetails[newRequestId].apiUintVars[keccak256("
        totalTip")],self.requestDetails[newRequestId].queryString);
443 }
444
445
446 /**
447 * @dev Getter function for the request with highest payout. This function is used
    within the getVariablesOnDeck function
448 * @return uint _requestId of request with highest payout at the time the function
    is called
449 */
450 function getTopRequestId(TellorStorage.TellorStorageStruct storage self) internal
    view returns(uint _requestId){
451     uint _max;
452     uint _index;
453     (_max,_index) = Utilities.getMax(self.requestQ);
454     _requestId = self.requestIdByRequestQIndex[_index];
455 }
456
457
458 /**
459 * @dev Gets the 5 miners who mined the value for the specified requestId/
    _timestamp
460 * @param _requestId to look up
461 * @param _timestamp is the timestamp to look up miners for
462 * @return bool true if requestId/timestamp is under dispute
463 */
464 /*@CTK "isInDispute correctness"
465     @post __return == self.requestDetails[_requestId].inDispute[_timestamp]
466     @post !(__has_overflow)
467     @post !(__has_buf_overflow)
468     @post !(__has_assertion_failure)
469 */
470 function isInDispute(TellorStorage.TellorStorageStruct storage self, uint
    _requestId, uint _timestamp) internal view returns(bool){
471     return self.requestDetails[_requestId].inDispute[_timestamp];
472 }
473
474
475 /**
476 * @dev Retrieve value from oracle based on requestId/timestamp
477 * @param _requestId being requested
478 * @param _timestamp to retrieve data/value from
479 * @return uint value for requestId/timestamp submitted
480 */
481 /*@CTK "retrieveData correctness"
482     @post __return == self.requestDetails[_requestId].finalValues[_timestamp]
483     @post !(__has_overflow)
484     @post !(__has_buf_overflow)
485     @post !(__has_assertion_failure)
486 */

```

```
487     function retrieveData(TellorStorage.TellorStorageStruct storage self, uint
488         _requestId, uint _timestamp) internal view returns (uint) {
489         return self.requestDetails[_requestId].finalValues[_timestamp];
489     }
490
491
492     /**
493     * @dev Getter for the total_supply of oracle tokens
494     * @return uint total supply
495     */
496     function totalSupply(TellorStorage.TellorStorageStruct storage self) internal view
497         returns (uint) {
497         return self.uintVars[keccak256("total_supply")];
498     }
499
500 }
```