On Ink Integration Tests 3

Milestone Report

November, 2023

Milestone 3: Execution

Intro

We have discovered that integration tests for ink! contracts lack some of the functionalities, or present implementation differences, when compared to E2E testing.

We have already conducted a comprehensive analysis to identify any missing functionalities in integration tests and implementation differences with E2E tests, and to propose and develop new testing features based on our findings. This analysis was carried out as part of previous grants link1 and link2 and documented in our analysis repository.

In this grant proposal we resolved the issues identified in the previous milestones.

Timeline

Started on 2023-10-30, ending on 2023-11-24. (Delivered on 2023-12-05).

Total duration: 4 weeks.

What was defined

Definitions below correspond to the deliverables proposed in our grant application CoinFabrik On Ink Integration Tests 3.

1. **Documentation.**

   We will write a comprehensive report that compares the functionalities of integration tests and E2E (End-to-End) tests. This report will focus on the the functions to be implemented in this milestone, corresponding to issues 3-invoke_contract_delegate(), 4-invoke_contract(), 6-set_code_hash(), `8-caller_is_origin()`, `9-code_hash()`, `10-own_code_hash()`, and `17-balance()`.

   In the first week of this milestone, we will contact the ink! development team to provide an initial report on `14-weight_to_fee()`, documenting our efforts to identify the source of its implementation issues and seeking collaboration to assess the feasibility of resolving them. We will document any progress and implementations related to `14-weight_to_fee()` in our final milestone report.

   We will document any additional work that was required in order to ensure consistency between integration and e2e tests.

If applicable, we will suggest additional tests outside of the scope of this milestone. Particularly, for functions declared outside of the `env_access.rs` file, but that could be related to integration or e2e testing.

2. **Testing and Testing Guide.** The newly developed functionalities will be documented and tested following existing contribution guidelines. A testing guide will be included.

3. **Article.** We will prepare a summary report and publish it on our blog https://blog.coinfabrik.com/.

4. **Develop**.

   We will implement the missing functionalities or resolve implementation differences for function issues `3-invoke_contract_delegate()`, `4-invoke_contract()`, `6-set_code_hash()`, `8-caller_is_origin()`, `9-code_hash()`, `10-own_code_hash()` and `17-balance()`.

   We will also make the necessary changes to address the issues highlighted in our initial report on `14-weight_to_fee()`, provided that these changes are deemed feasible during our discussions with the ink! development team.

   All these implementations or modifications will be pushed into the ink! project repository following existing contribution guidelines.

   If applicable, we will develop additional tests or make ad hoc improvements to the ink codebase necessary for the completion of this milestone. Particularly for functions declared outside the `env_access.rs` file that might be related to integration or end-to-end testing.

5. **Quality Assurance.** We will adhere to existing contribution guidelines and add necessary tests to integrate the new implemented or corrected functions to the ink! project repository.

What we did

1. **Documentation.**

   We worked on the comparison report, providing documentation on the developed functions. Check the report included as *Appendix 1*.

   We also contacted the ink! development team and provided a report on `weight_to_fee()`, raised as issue #1985 in the requested repository. The implementation of `weight_to_fee()` in both the integration and E2E environments returned a valid gas price (u128) for a given gas amount. However, the price per gas unit differed in both environments: it was 100 in integration tests and always 0 in E2E

tests.This was discussed with @cmichi and after being reviewed by @smiasojed was resolved in pull request PR #219 to substrate-contracts-node.

We provide additional documentation on new issues we found as we performed our fixes in section *Other Findings*.

2. **Testing and Testing Guide.** We followed contribution guidelines for the pull requests associated with the implementations of functions: `3-invoke_contract_delegate()`, `4-invoke_contract()`, `6-set_code_hash()`, `8-caller_is_origin()`, `9-code_hash()`, `10-own_code_hash()` and `17-balance()`.

   For these functions, we added test cases in their target directories correspondingly. We include a testing guide on how to execute these tests in the *Reference 1: Implementation Summary* of section *Execution* of the *Appendix 1*, and in the documentation of each corresponding pull request.

3. **Article.** Check our summary report published in our blog.

4. **Develop.** We provide implementations for the functions specified in this milestone: `3-invoke_contract_delegate()`, `4-invoke_contract()`, `6-set_code_hash()`, `8-caller_is_origin()`, `9-code_hash()`, `10-own_code_hash()` and `17-balance()`. Check documentation on these developments in *Reference 1: Implementation Summary* of the section *Execution* of the *Appendix 1.* We also provide ad-hoc developments for functions *return_value()* and *set_balance()*, documented in section *Ad-hoc developments* of the *Appendix 1.*

5. **Quality Assurance.** We reviewed and followed the established contribution guidelines for the pull requests performed.

Appendix 1: Report on ink! integration tests

Abstract

Integration tests run significantly faster than e2e tests. If a full range of functionalities were provided, it could reduce testing and QA times. Ideally, each test in e2e should have its counterpart in integration.

We have discovered that integration tests for ink! contracts lack some of the functionalities that are available in e2e testing via the natural functionalities provided by a node.

As part of a previous grant application CoinFabrik_On_Ink_Integration_Tests, we divided the workload into 24 functions exposed for their usage in ink! integration and e2e testing environments. Through this grant, and subsequent grant CoinFabrik_On_Ink_integration_Tests_2, we performed a thorough review of the 24 functions, providing documentation and test cases for each function in our analysis repository.

From this analysis we determined that 13 functions showed missing implementations for integration tests or implementation differences when compared to e2e tests. The implementation of 2 of these functions was deemed unfeasible. For the remaining 11 functions, we established implementation plans and carried out implementations for 10 of them. For `weight_to_fee()`, we raised an issue (#1985) with an associated report in order to discuss it with the ink development team; the issue was finally resolved in PR #219 to substrate-contracts-node.

Of the 10 implemented functions, 3 of them were implemented as part of previous milestone CoinFabrik_On_Ink_integration_Tests_2 and 7 of them as part of current milestone CoinFabrik_On_Ink_Integration_Tests_3.

Scope and Initial Status

Our analysis and implementation work was focused on the 24 functions exposed for their usage in integration and e2e tests in the `env_access.rs` file.

Table 1: Exposed Functions

Test Type	Exposed Functions	Reviewed Commit
Integration	https://github.com/paritytech/ink/blob/ma	c2af39883aab48c71dc09dac5d065

E2E	ster/crates/ink/src/env_access.rs	83f2e84dc54

In the table below *Table 2: Initial Status*, we enumerate these functions, and mention their initial status before the commencement of this milestone. The Issue Number represents the order in which we initially reviewed these functions. This analysis is based on the commit c2af39883aab48c71dc09dac5d06583f2e84dc54 of the ink! Repository.

In subsequent *Table 3: Updated Status after Milestone 2,* we provide the status of each function at the commencement of current Milestone 3.

Table 2: Initial Status

Issue Number	Function	Implemented Integration Tests	Implemented E2E Tests	Initial Status
1	`default_accounts()`	Yes	Yes	Implementation Difference.
2	`set_contract_storage()`	Yes	Yes	Missing limitation on Integration Testing.
3	`invoke_contract_delegate()`	No	Yes	Missing Function Implementation on Integration Testing.
4	`invoke_contract()`	No	Yes	Missing Function Implementation on Integration Testing
5	`gas_left()`	No	Yes	Missing Function Implementation on Integration Testing. Unfeasible Implementation.
6	`set_code_hash()`	No	Yes	Missing Function Implementation on Integration Testing.
7	`instantiate_contract()`	No	Yes	Missing Function Implementation on Integration Testing.
8	`caller_is_origin()`	No	Yes	Missing Function Implementation on Integration Testing.

9	`code_hash()`	No	Yes	Missing Function Implementation on Integration Testing.
10	`own_code_hash()`	No	Yes	Missing Function Implementation on Integration Testing.
11	`call_runtime()`	No	Yes	Missing Function Implementation on Integration Testing. Unfeasible Implementation.
12	`caller()`	Yes	Yes	Pending Analysis for Corrections in Implementation Differences.
13	`transferred_value()`	Yes	Yes	Pending Analysis for Corrections in Implementation Differences.
14	`weight_to_fee()`	Yes	Yes	Pending Analysis for Corrections in Implementation Differences.
15	`block_timestamp()`	Yes	Yes	Pending Analysis for Corrections in Implementation Differences.
16	`account_id()`	Yes	Yes	Pending Analysis for Corrections in Implementation Differences.
17	`balance()`	Yes	Yes	Pending Analysis for Corrections in Implementation Differences.
18	`block_number()`	Yes	Yes	Pending Analysis for Corrections in Implementation Differences.
19	`minimum_balance()`	Yes	Yes	Pending Analysis for Corrections in

				Implementation Differences.
20	`terminate_contract()`	Yes	Yes	Pending Analysis for Corrections in Implementation Differences.
21	`transfer()`	Yes	Yes	Pending Analysis for Corrections in Implementation Differences.
22	`hash_bytes()`	Yes	Yes	Pending Analysis for Corrections in Implementation Differences.
23	`hash_encoded()`	Yes	Yes	Pending Analysis for Corrections in Implementation Differences.
24	`ecdsa_recover()`	Yes	Yes	Pending Analysis for Corrections in Implementation Differences.

Table 3: Updated Status after Milestone 2[1]

In the table below, we use colors to tag the implementation status of the analyzed functions:

- Resolved (in green): functions that have been correctly implemented or where the implementation differences between integration tests and end-to-end (e2e) tests have been resolved.
- Unfeasible (in white), functions for which implementation in integration tests was deemed unfeasible in Milestone 1.
- Pending (in yellow): functions with pending implementation or pending correction of the implementation differences between integration tests and end-to-end (e2e) tests after Milestone 2. These are the functions resolved in current Milestone 3.

Issue Number	Function	Implemented Integration Tests	Implemented E2E Tests	Updated Status
1	default_accounts()	Yes	Yes	Implementation Difference Corrected. Pull request #1955 performed.
2	set_contract_storage()	Yes	Yes	Missing limitation on Integration Testing Implemented. Pull request #1961 performed.
3	invoke_contract_delegate()	No	Yes	Missing Function Implementation on Integration Testing.
4	invoke_contract()	No	Yes	Missing Function Implementation on Integration Testing
5	gas_left()	No	Yes	Missing Function Implementation on Integration Testing. **Unfeasible Implementation.** Because integration tests are performed on native code

[1] Before the commencement of this current Milestone 3.

				rather than WASM code, and because gas cost is based on the number of WASM instructions executed, implementing this function is impractically complex.
6	set_code_hash()	No	Yes	Missing Function Implementation on Integration Testing.
7	instantiate_contract()	No. Pull request performed.	Yes	Missing Function Implementation on Integration Testing Performed. Pull Request #1963.[2]
8	caller_is_origin()	No	Yes	Missing Function Implementation on Integration Testing.
9	code_hash()	No	Yes	Missing Function Implementation on Integration Testing.
10	own_code_hash()	No	Yes	Missing Function Implementation on Integration Testing.
11	call_runtime()	No	Yes	Missing Function Implementation on Integration Testing. **Unfeasible Implementation.** Implementing this function is unfeasible, as it would require emulating practically the entire node to get consistent results.

[2] We initially performed the Pull Request #1963 to implement instantiate_contract(), but decided to close it and perform a new Pull Request #1988 to include necessary implementations of other related functions, which were developed in Milestone 3.

				Testing a contract function that calls into the runtime should be done using E2E tests.
12	caller()	Yes	Yes	Ok. No difference observed in testing.
13	transferred_value()	Yes	Yes	Ok. No difference observed in testing.
14	weight_to_fee()	Yes	Yes	Implementation Difference.
15	block_timestamp()	Yes	Yes	Ok. No difference observed in testing.
16	account_id()	Yes	Yes	Ok. No difference observed in testing.
17	balance()	Yes	Yes	Implementation Difference.
18	block_number()	Yes	Yes	Ok. No difference observed in testing.
19	minimum_balance()	Yes	Yes	Ok. No difference observed in testing.
20	terminate_contract()	Yes	Yes	Ok. No difference observed in testing.
21	transfer()	Yes	Yes	Ok. No difference observed in testing.
22	hash_bytes()	Yes	Yes	Ok. No difference observed in testing.
23	hash_encoded()	Yes	Yes	Ok. No difference observed in testing.
24	ecdsa_recover()	Yes	Yes	Ok. No difference observed in testing.

Execution

We performed the implementation of the 7 selected functions and performed some necessary ad-hoc developments, while adhering to CI/CD best practices.

We have also raised issues in the corresponding directories for findings that remained unresolved from Milestone 2 or were undergoing review.

Implementation of Selected Functions

We performed the implementation of missing functionalities or corrected implementation differences for functions 3-invoke_contract_delegate(), 4-invoke_contract(), 6-set_code_hash(), `8-caller_is_origin()`, `9-code_hash()`, `10-own_code_hash()`, and `17-balance()`.

We also updated function `7-instantiate_contract()`, as we worked on new functions that depended on invoking instantiated contracts.

In the tables below, we summarize the relevant data associated with the development of these functions and provide a short testing guide.

Reference 1: Implementation Summary

Function	`instantiate_contract()`
Initial Status	Missing Function Implementation on Integration Testing.
Issue Description	There is no implementation of `instantiate_contract()` for integration tests. This prevents invoking one contract from another.
Issue Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/blob/main/test-cases/instantiate-contract
Current Status	Missing Implementation Performed. Test Cases Passed. Pull Request Performed to Corresponding Repository.
Implementation Notes	In order to allow contract instantiation, we modified the `instantiate_contract()` function in PR #1988 to call the dispatch code generated by the ink! code generator. This way, `instantiate_contract()` can call contract constructors, allowing for the instantiation of contracts in integration tests. We also correctly keep track of callees as we move along the call stack, and set the code hash for the new contract account, as well as give it its endowment.

	For correct behavior emulation, a test should call `ink::env::test::upload_code()` before calling `instantiate_contract()`, which will simulate uploading the code of a contract to the environment and giving back a code hash. Something else worth noting is that we had to move the types under `ink::reflect to the ink_primitives` crate, since now both the `ink` and `ink_env` crates depend on those types and there was no other way to avoid a circular dependency. Observation 1: In order to use contract instantiation in integration tests it is necessary to invoke `cargo` with: `cargo test --features test_instantiate` This is necessary because getting back the return value of the constructor involved allowing the `execute_dispatchable()` function to return normally, but on-chain it never returns and instead the on-chain implementation of `return_value()` triggers a WASM trap that the runtime catches. In order to avoid changing the behavior of on-chain contracts, we preferred requiring enabling a feature just for off-chain tests, which modifies the signature of `return_value()` and allows a normal return from execute_dispatchable(). Observation 2: Initially, for Milestone 2, we opened PR #1963 just for `instantiate_contract()`, since we internally organized our work separated by function. However, when we got to `invoke_contract()` in Milestone 3 we realized that these functions were too interconnected for separate PRs, so we decided to just have a single PR #1988 for all of them, closing PR #1963. In the first PR #1963 we included a function `set_contract_storage_test()` in order to obtain the storage key of the contract, which we removed in the second PR #1988 because we deemed it was not necessary.
Updated Documentation	We updated the original documentation of this issue in our analysis repo, adding the section Update on Correcting this Issue and informing of the correction. There is no necessary update to the associated documentation: https://paritytech.github.io/ink/ink_env/fn.instantiate_contra

	ct.html We added inline documentation for `instantiate_contract()`.
Testing Guide	In order to test `instantiate_contract()`, in the directory `integration-tests/contract-invocation` of the target directory, we include in our pull request a test case showing that the missing implementation has been developed. Note that this test is different from the original test case in our repository, which showed the difference between integration and e2e tests. In order to run the integration tests run: `cargo test --features test_instantiate`
Link to Pull Request	PR #1988

Functions	`invoke_contract()` and `invoke_contract_delegate()`
Initial Status	Missing Function Implementation on Integration Testing.
Issue Description	There is no implementation of `invoke_contract()` and `invoke_contract_delegate()` for integration tests.
Issue Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/invoke-contract https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/invoke-contract-delegate
Current Status	Missing Implementations Performed. Test Cases Passed. Pull Request Performed to Corresponding Repository.
Implementation Notes	Functions `invoke_contract()` and `invoke_contract_delegate()` have almost identical implementations in our proposed PR #1988. After getting their arguments from the `CallParams` object they call a new internal function `invoke_contract_impl()` which handles the invocation logic. Of note is that the actual call into the generated dispatch function is done by `execute_contract_call()`. This function is only instantiated by rustc per-contract when a test calls `ink::env::test::upload_code()`. Function `upload_code()` adds a reference to `execute_contract_call()` to the environment's database. Function `invoke_contract_impl()` fetches the

	reference and calls it. Before and after this call it takes care of execution context bookkeeping.
Updated Documentation	We updated the original documentation of this issue in our analysis repo, adding the section Update on Correcting this Issue and informing of the correction of both functions. There is no necessary update to the associated documentation: https://paritytech.github.io/ink/ink_env/fn.invoke_contract.html https://paritytech.github.io/ink/ink_env/fn.invoke_contract_delegate.html We added inline documentation for invoke_contract() and invoke_contract_delegate().
Testing Guide	In order to test `invoke_contract()` and `invoke_contract_delegate()`, in the directory `integration-tests/contract-invocation` of the target directory, we include in our pull request a test case showing that the missing implementation has been developed. Note that this test is different from the original test cases in our repository, which showed the difference between integration and e2e tests. In order to run the integration tests run: `cargo test --features test_instantiate`
Link to Pull Request	PR #1988

Function	`set_code_hash()`
Initial Status	Missing Function Implementation on Integration Testing.
Issue Description	Not implemented for integration test environment. Would permit contract upgradability.
Issue Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/set-code-hash
Current Status	Missing Implementation Performed. Test Cases Passed. Pull Request Performed to Corresponding Repository.
Implementation Notes	Our implementation in PR #1988 overwrites the value set for the code hash of an account in the database.
Updated Documentation	We updated the original documentation of this issue in our analysis repository, adding the section Update on Correcting this Issue and informing of the correction.

	There is no necessary update to the associated documentation: https://paritytech.github.io/ink/ink_env/fn.set_code_hash.html
Testing Guide	In order to test `set_code_hash()`, in the directory `integration-tests/contract-invocation` of the target directory, we include in our pull request a test case showing that the missing implementation has been developed. Note that this test is different from the original test case in our repository, which showed the difference between integration and e2e tests. In order to run the integration tests run: `cargo test --features test_instantiate`
Link to Pull Request	PR #1988

Functions	`caller_is_origin()`
Initial Status	Missing Function Implementation on Integration Testing.
Issue Description	Integration testing is missing critical functionality to properly write tests using `caller_is_origin()`.
Issue Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/caller-is-origin
Current Status	Missing Implementation Performed. Test Cases Passed. Pull Request Performed to Corresponding Repository.
Implementation Notes	We have added the `depth` field to the `ExecContext` struct in order to be able to detect caller changes every time calls are made between contracts. Basically every time the callee changes in the `instantiate_contract()` or `invoke_contract()` function calls, 1 is added to `depth`. And when the callee returns to the previous one, 1 is subtracted. The `caller_is_origin()` function compares the value of `depth` to zero to check if the current contract caller is the origin of the entire call stack.
Updated Documentation	We updated the original documentation of this issue in our analysis repo, adding the section Update on Correcting this Issue and informing of the correction.

	There is no necessary update to the associated documentation: https://paritytech.github.io/ink/ink_env/fn.caller_is_origin.html We added inline documentation for the added `depth` variable.

Testing Guide	In order to perform this test, in the directory `integration-tests/caller-is-origin` of the target directory, we include in our pull request a test case showing that the missing implementation has been developed. Note that this test is different from the original test case in our repository, which showed the difference between integration and e2e tests.
	
	In order to run the integration tests run:
	`cargo test`
	
	In order to run the e2e tests run:
	`cargo test --features e2e-tests`

Link to Pull Request	PR #1991

Function	`code_hash()`
Initial Status	Missing Function Implementation on Integration Testing.
Issue Description	`code_hash()` retrieves the code hash of the contract at the specified account id. It is not possible to use this function in the integration test environment.
Issue Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/code-hash
Current Status	Missing Implementation Performed. Test Cases Passed. Pull Request Performed to Corresponding Repository.
Implementation Notes	We implemented the function in PR #1988 so that it returns a value that is unique for the contract, but it does not emulate what would be returned on-chain. This value is a hash of a pointer to a function generated for each contract by the compiler. Therefore, this value is unique for each contract, which is the intended purpose of this function.
Updated Documentation	We updated the original documentation of this issue in our analysis repository, adding the section Update on Correcting this Issue and informing of the correction. There is no necessary update to the associated

	documentation: https://paritytech.github.io/ink/ink_env/fn.code_hash.html
Testing Guide	In order to test `code_hash()`, in the directory `integration-tests/contract-invocation` of the target directory, we include in our pull request a test case showing that the missing implementation has been developed. Note that this test is different from the original test case in our repository, which showed the difference between integration and e2e tests. In order to run the integration tests run: `cargo test --features test_instantiate`
Link to Pull Request	PR #1988

Function	`own_code_hash()`
Initial Status	Missing Function Implementation on Integration Testing.
Issue Description	`own_code_hash()` retrieves the code hash of the currently executing contract. It is not possible to use this function in the integration test environment.
Issue Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/own-code-hash
Current Status	Missing Implementation Performed. Test Cases Passed. Pull Request Performed to Corresponding Repository.
Implementation Notes	Our implementation in PR #1988 returns the code hash of the account of the calling contract. It simply fetches the code hash stored in the database by `instantiate_contract()` or set by `set_code_hash()`.
Updated Documentation	We updated the original documentation of this issue in our analysis repository, adding the section Update on Correcting this Issue and informing of the correction. There is no necessary update to the associated documentation: https://paritytech.github.io/ink/ink_env/fn.own_code_hash.html
Testing Guide	In order to test `own_code_hash()`, in the directory `integration-tests/contract-invocation` of the target directory, we include in our pull request a test case showing that the missing implementation has been developed. Note that this test is different from the original test case in our repository, which showed the difference

	between integration and e2e tests.
	
	In order to run the integration tests run:
	`cargo test`
	
	In order to run the e2e tests run:
	`cargo test --features e2e-tests`

Link to Pull Request	PR #1988

Functions	`balance()`
Initial Status	Implementation Difference
Issue Description	Initial balance differs in both environments. It is set to 1.000.000 in integration tests and to 1.000.000.000 in e2e tests. Furthermore, the initial balance of default addresses also differed in both environments.
Issue Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/balance
Current Status	Implementation Difference Corrected. Pull Request Performed to Corresponding Repository.
Implementation Notes	In PR #1982 we updated the default balance in integration tests to be the same as in e2e-tests. This is not definitive, as we want opinions if this should be like this. We also corrected the initial balance of default addresses, updating PR #1955 from Milestone 2.
Updated Documentation	We updated the original documentation of this issue in our analysis repo, adding the section Update on Correcting this Issue and informing of the correction. There is no necessary update to the associated documentation: https://paritytech.github.io/ink/ink_env/fn.balance.html
Testing Guide	No tests were required for this function since it simply implied updating a variable.
Link to Pull Request	PR #1982

Ad-hoc developments

In addition to the developments of the functions within the scope of this milestone, we performed implementations for `return_value()` and `set_balance()`.

On return_value()

This function, `return_value()`, was not implemented for integration testing and was necessary for the implementation of `instantiate_contract()`, `invoke_contract()` and `invoke_contract_delegate()`.

Our proposed implementation of `return_value()` in PR #1988 uses `set_contract_storage()` to store the return value of the contract method. It uses the key `0xFFFF…` to avoid hash collisions (`0x0000…` is used for the contract's own state). As noted under `instantiate_contract()`, we decided to return normally from `execute_dispatchable()`.

Since `execute_dispatchable()` has no way to communicate a value back to the caller, it was necessary to store the data somehow somewhere. Since the call to `return_value()` is generated anyway, this was the obvious place to do it. This is why we had to implement this function, even though it was not in our milestone implementation plans.

It's also worth noting that enabling `test_instantiate` modifies `return_value()`'s signature to allow it to return normally, instead of never returning.

On set_balance

As we developed missing implementations of `balance()` for integration testing (refer to the `balance()` analisis documentation), we noticed that the `set_balance()` function allowed the balance to be set below the existential minimum, even in cases where the balance being set was not zero.

Given that accounts with a balance below the existential minimum are reaped on-chain, it seems logical to add a check that prevents setting the balance below this threshold in ink! integration tests. We address this issue in PR #1983, providing an implementation with the caveat that setting the balance to 0 is permitted for reaping the account.

Other Findings

The following are issues related to findings that were not completely resolved on Milestone 2.

On weight_to_fee()

When calling weight_to_fee() in a contract while performing an integration test we obtain a value of 100. However when we do it while performing an e2e test we obtain a value of 0, no matter the gas provided.

As we observed in the analisis documentation of `weight_to_fee()`, it was unclear which function was responsible for `weight_to_fee()` returning `0` in e2e tests. In order to determine this, we contacted the ink! development team and provided a report on this finding, raised as issue #1985 in the requested repository. This was discussed with @cmichi and, after being reviewed by @smiasojed, was resolved in PR #219 to substrate-contracts-node.

On invalid length control

In line 174 of /crates/storage/src/lazy/mapping.rs `try_insert()` checks whether `key.length() + value.length() <= 2^14`. If this condition is met, it then calls `insert()`.
However, `insert()` passes the tuple `(&KeyType::KEY, key)` as the key to `set_contract_storage()`. Since `KeyType::KEY` is a `u32`, the effective key size after serialization is `key.length() + 4`. Therefore `try_insert()` should instead check whether `key.length() + value.length() + 4 <= 2^14`.

This issue was resolved in PR #1961 and adds to the work done on function `set_contract_storage()` in Milestone 2.

Summary of results

For the universe of 24 functions in our analysis scope, we update their status in the *Table 4* below.

Notice that all functions highlighted as Pending in *Table 3* have been Resolved.

Table 4: Updated Status

Issue Number	Function	Implemented Integration Tests	Implemented E2E Tests	Updated Status
1	default_accounts()	Yes	Yes	Implementation Difference Corrected. Pull request #1955 performed.
2	set_contract_storage()	Yes	Yes	Missing limitation on Integration Testing Implemented. Pull request #1961 performed.
3	invoke_contract_delegate()	Pull request with implementation performed.	Yes	Missing Function Implementation on Integration Testing Performed. Pull request #1988 performed.
4	invoke_contract()	Pull request with implementation performed.	Yes	Missing Function Implementation on Integration Testing Performed. Pull request #1988 performed.
5	gas_left()	No	Yes	Missing Function Implementation on Integration Testing. **Unfeasible Implementation.** Because integration tests are performed

				on native code rather than WASM code, and because gas cost is based on the number of WASM instructions executed, implementing this function is impractically complex.
6	set_code_hash()	Pull request with implementation performed.	Yes	Missing Function Implementation on Integration Testing Performed. Pull request #1988 performed.
7	instantiate_contract()	Pull request with implementation performed.	Yes	Missing Function Implementation on Integration Testing Performed. Pull request #1988 performed.[3]
8	caller_is_origin()	Pull request with implementation performed.	Yes	Missing Function Implementation on Integration Testing Performed. Pull request #1991 performed.
9	code_hash()	Pull request with implementation performed.	Yes	Missing Function Implementation on Integration Testing Performed. Pull request #1988 performed.
10	own_code_hash()	Pull request with implementation performed.	Yes	Missing Function Implementation on Integration Testing Performed. Pull request #1988 performed.

[3] We initially performed the Pull Request #1963 to implement instantiate_contract(), but decided to close it and perform a new Pull Request #1988 to include necessary implementations of other related functions, which were developed in Milestone 3.

11	call_runtime()	No	Yes	Missing Function Implementation on Integration Testing. **Unfeasible Implementation.** Implementing this function is unfeasible, as it would require emulating practically the entire node to get consistent results. Testing a contract function that calls into the runtime should be done using E2E tests.
12	caller()	Yes	Yes	Ok. No difference observed in testing.
13	transferred_value()	Yes	Yes	Ok. No difference observed in testing.
14	weight_to_fee()	Yes	Yes	Implementation Difference. Issue raised in #1985 and resolved in PR #219 to substrate-contracts-node by @smiasojed.
15	block_timestamp()	Yes	Yes	Ok. No difference observed in testing.
16	account_id()	Yes	Yes	Ok. No difference observed in testing.
17	balance()	Pull request with implementation performed.	Yes	Missing Function Implementation on Integration Testing Performed. Pull request #1982 performed. We also updated the initial balance for default accounts in Pull

				request #1955.
18	block_number()	Yes	Yes	Ok. No difference observed in testing.
19	minimum_balance()	Yes	Yes	Ok. No difference observed in testing.
20	terminate_contract()	Yes	Yes	Ok. No difference observed in testing.
21	transfer()	Yes	Yes	Ok. No difference observed in testing.
22	hash_bytes()	Yes	Yes	Ok. No difference observed in testing.
23	hash_encoded()	Yes	Yes	Ok. No difference observed in testing.
24	ecdsa_recover()	Yes	Yes	Ok. No difference observed in testing.

Next Steps

Having resolved the analyzed functions, we will follow the raised pull requests, performing required changes by ink! evaluators.