Access On-Chain Time

You have options when needing to access network-based time for your transactions. If you need a near real-time measurement (within a few seconds), use the immutable reference of time provided by the Clock module in Move. The reference value from this module updates with every network checkpoint. If you don't need as current a time slice, use the epoch_timestamp_ms function to capture the precise moment the current epoch started.

To access a prompt timestamp, you must pass a read-only reference of sui::clock::Clock as an entry function parameter in your transactions. An instance of Clock is provided at address 0x6 , no new instances can be created.

Use the timestamp_ms function from the sui::clock module to extract a unix timestamp in milliseconds.

The example below demonstrates an entry function that emits an event containing a timestamp from the Clock :

A call to the previous entry function takes the following form, passing 0x6 as the address for the Clock parameter:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

Expect the Clock timestamp to change at the rate the network generates checkpoints, which is about every 1/4 second with Mysticeti consensus. Find the current network checkpoint rate on this public dashboard .

Successive calls to sui::clock::timestamp_ms in the same transaction always produce the same result (transactions are considered to take effect instantly), but timestamps from Clock are otherwise monotonic across transactions that touch the same shared objects: Successive transactions seeing a greater or equal timestamp to their predecessors.

Any transaction that requires access to a Clock must go through consensus because the only available instance is a shared object. As a result, this technique is not suitable for transactions that must use the single-owner fastpath (see Epoch timestamps for a single-owner-compatible source of timestamps).

Transactions that use the clock must accept it as an immutable reference (not a mutable reference or value). This prevents contention, as transactions that access the Clock can only read it, so do not need to be sequenced relative to each other. Validators refuse to sign transactions that do not meet this requirement and packages that include entry functions that accept a Clock or &mut Clock fail to publish.

The following functions test Clock -dependent code by manually creating a Clock object and manipulating its timestamp. This is possible only in test code:

The next example presents a basic test that creates a Clock, increments it, and then checks its value:

Use the following function from the sui::tx_context module to access the timestamp for the start of the current epoch for all transactions (including ones that do not go through consensus):

The preceding function returns the point in time when the current epoch started, as a millisecond granularity unix timestamp in a u64 . This value changes roughly once every 24 hours , when the epoch changes.

Tests based on sui::test_scenario can use later_epoch (following code), to exercise time-sensitive code that uses epoch_timestamp_ms (previous code):

later_epoch behaves like sui::test_scenario::next_epoch (finishes the current transaction and epoch in the test scenario), but also increments the timestamp by delta_ms milliseconds to simulate the progress of time.

The sui::clock::Clock module

To access a prompt timestamp, you must pass a read-only reference of sui::clock::Clock as an entry function parameter in your transactions. An instance of Clock is provided at address 0x6 , no new instances can be created.

Use the timestamp_ms function from the sui::clock module to extract a unix timestamp in milliseconds.

The example below demonstrates an entry function that emits an event containing a timestamp from the Clock :

A call to the previous entry function takes the following form, passing 0x6 as the address for the Clock parameter:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

Expect the Clock timestamp to change at the rate the network generates checkpoints, which is about every 1/4 second with Mysticeti consensus. Find the current network checkpoint rate on this public dashboard .

Successive calls to sui::clock::timestamp_ms in the same transaction always produce the same result (transactions are considered to take effect instantly), but timestamps from Clock are otherwise monotonic across transactions that touch the same shared objects: Successive transactions seeing a greater or equal timestamp to their predecessors.

Any transaction that requires access to a Clock must go through consensus because the only available instance is a shared object. As a result, this technique is not suitable for transactions that must use the single-owner fastpath (see Epoch timestamps for a single-owner-compatible source of timestamps).

Transactions that use the clock must accept it as an immutable reference (not a mutable reference or value). This prevents contention, as transactions that access the Clock can only read it, so do not need to be sequenced relative to each other. Validators refuse to sign transactions that do not meet this requirement and packages that include entry functions that accept a Clock or &mut Clock fail to publish.

The following functions test Clock -dependent code by manually creating a Clock object and manipulating its timestamp. This is possible only in test code:

The next example presents a basic test that creates a Clock, increments it, and then checks its value:

Use the following function from the sui::tx_context module to access the timestamp for the start of the current epoch for all transactions (including ones that do not go through consensus):

The preceding function returns the point in time when the current epoch started, as a millisecond granularity unix timestamp in a u64 . This value changes roughly once every 24 hours , when the epoch changes.

Tests based on sui::test_scenario can use later_epoch (following code), to exercise time-sensitive code that uses epoch_timestamp_ms (previous code):

later_epoch behaves like sui::test_scenario::next_epoch (finishes the current transaction and epoch in the test scenario), but also increments the timestamp by delta_ms milliseconds to simulate the progress of time.

Epoch timestamps

Use the following function from the sui::tx_context module to access the timestamp for the start of the current epoch for all transactions (including ones that do not go through consensus):

The preceding function returns the point in time when the current epoch started, as a millisecond granularity unix timestamp in a u64 . This value changes roughly once every 24 hours , when the epoch changes.

Tests based on sui::test_scenario can use later_epoch (following code), to exercise time-sensitive code that uses epoch_timestamp_ms (previous code):

later_epoch behaves like sui::test_scenario::next_epoch (finishes the current transaction and epoch in the test scenario), but also increments the timestamp by delta_ms milliseconds to simulate the progress of time.