Ledger data types

Compact language version 0.17.0, compiler version 0.25.0.

Kernel

This ADT is a special ADT defining various built-in operations and valid only as a top-level ADT type.

blockTimeGreaterThan

```
blockTimeGreaterThan(time: Uint<64>): Boolean
```

Checks whether the current block time (measured in seconds since the Unix epoch) is greater than the given amount.

blockTimeLessThan

```
blockTimeLessThan(time: Uint<64>): Boolean
```

Checks whether the current block time (measured in seconds since the Unix epoch) is less than the given amount.

checkpoint

```
checkpoint(): []
```

Marks all execution up to this point as being a single atomic unit, allowing partial transaction failures to be split across it.

claimContractCall

```
claimContractCall(addr: Bytes<32>, entry_point: Bytes<32>, comm: Field): []
```

Require the presence of another contract call in the containing transaction, with a match address, entry point hash, and communication commitment, that is not claimed by any other call.

claimZswapCoinReceive

```
claimZswapCoinReceive(note: Bytes<32>): []
```

Requires the presence of a commitment in the containing transaction and that no other call claims it as a receive.

claimZswapCoinSpend

```
claimZswapCoinSpend(note: Bytes<32>): []
```

Requires the presence of a commitment in the containing transaction and that no other call claims it as a spend.

claimZswapNullifier

```
claimZswapNullifier(nul: Bytes<32>): []
```

Requires the presence of a nullifier in the containing transaction and that no other call claims it.

mint

```
mint(domain_sep: Bytes<32>, amount: Uint<64>): []
```

Mints a given amount of shielded coins with a token type derived from the contract's address, and a given domain separator.

self

```
self(): ContractAddress
```

Returns the current contract's address. ContractAddress is defined in CompactStandardLibrary.

Cell<value_type>

This ADT is a single Cell containing a value of type value_type and is used implicitly when the ledger field type is an ordinary Compact type. Programmers cannot write Cell explicitly when declaring a ledger field..

read

```
read(): value_type
```

Returns the current contents of this Cell.

available from Typescript as a getter on the ledger field

resetToDefault

```
resetToDefault(): []
```

Resets this Cell to the default value of its type.

write

```
write(value: value_type): []
```

Overwrites the content of this Cell with the given value.

writeCoin

```
writeCoin(coin: CoinInfo, recipient: Either<ZswapCoinPublicKey, ContractAddress>): []
```

Writes a CoinInfo to this Cell, which is transformed into a QualifiedCoinInfo at runtime by looking up the relevant Merkle tree index. This index must have been allocated within the current transaction or this write fails. CoinInfo, ContractAddress, Either, and ZswapCoinPublicKey are defined in CompactStandardLibrary.

available only for QualifiedCoinInfo value_type

Counter

This ADT is a simple counter.

decrement

```
decrement(amount: Uint<16>): []
```

Decrements the counter by a given amount. Decrementing below zero results in a run-time error.

increment

```
increment(amount: Uint<16>): []
```

Increments the counter by the given amount.

lessThan

```
lessThan(threshold: Uint<64>): Boolean
```

Returns if the counter is less than the given threshold value.

Feedback

read

```
read(): Uint<64>
```

Retrieves the current value of the counter.

available from Typescript as a getter on the ledger field

resetToDefault

```
resetToDefault(): []
```

Resets this Counter to its default value of 0.

Set<value_type>

This ADT is an unbounded set of values of type value_type.

insert

```
insert(elem: value_type): []
```

Updates this Set to include a given element.

insertCoin

```
insertCoin(coin: CoinInfo, recipient: Either<ZswapCoinPublicKey, ContractAddress>): []
```

Inserts a CoinInfo into this Set, which is transformed into a QualifiedCoinInfo at runtime by looking up the relevant Merkle tree index. This index must have been allocated within the current transaction or this insertion fails. CoinInfo, ContractAddress, Either, and ZswapCoinPublicKey are defined in CompactStandardLibrary.

available only for QualifiedCoinInfo value_type

isEmpty

```
isEmpty(): Boolean
```

Returns whether this Set is the empty set.

available from Typescript as `isEmpty(): boolean`

member

```
member(elem: value_type): Boolean
```

Returns if an element is contained within this Set.

available from Typescript as `member(elem: value_type): boolean`

Feedback

remove

```
remove(elem: value_type): []
```

Update this Set to not include a given element.

resetToDefault

```
resetToDefault(): []
```

Resets this Set to the empty set.

size

```
size(): Uint<64>
```

Returns the number of unique entries in this Set.

available from Typescript as `size(): bigint`

[Symbol.iterator]

callable only from TypeScript

```
[Symbol.iterator](): Iterator<value_type>
```

Iterates over the entries in this Set.

Map<key_type, value_type>

This ADT is an unbounded set of mappings between values of type key_type and values of type value_type.

insert

```
insert(key: key_type, value: value_type): []
```

Updates this Map to include a new value at a given key.

insertCoin

```
insertCoin(key: key_type, coin: CoinInfo, recipient: Either<ZswapCoinPublicKey, ContractAddress>):
[]
```

Inserts a CoinInfo into this Map at a given key, where the CoinInfo is transformed into a QualifiedCoinInfo at runtime by looking up the relevant Merkle tree index. This index must have been allocated within the current transaction or this insertion fails. CoinInfo, ContractAddress, Either, and ZswapCoinPublicKey are defined in CompactStandardLibrary.

available only for QualifiedCoinInfo value_type

insertDefault

```
insertDefault(key: key_type): []
```

Updates this Map to include the value type's default value at a given key.

isEmpty

```
isEmpty(): Boolean
```

Returns if this Map is the empty map.

available from Typescript as `isEmpty(): boolean`

lookup

```
lookup(key: key_type): value_type
```

Looks up the value of a key within this Map. The returned value may be another ADT.

available from Typescript as `lookup(key: key_type): value_type`

member

```
member(key: key_type): Boolean
```

Returns if a key is contained within this Map.

available from Typescript as `member(key: key_type): boolean`

remove

```
remove(key: key_type): []
```

Updates this Map to not include a given key.

resetToDefault

```
resetToDefault(): []
```

Resets this Map to the empty map.

size

```
size(): Uint<64>
```

Returns the number of entries in this Map.

available from Typescript as `size(): bigint`

[Symbol.iterator]

callable only from TypeScript

Feedback

```
[Symbol.iterator](): Iterator<[key_type, value_type]>
```

Iterates over the key-value pairs contained in this Map.

List<value_type>

This ADT is an unbounded list of values of type value_type.

head

```
head(): Maybe<value_type>
```

Retrieves the head of this List, returning a Maybe, ensuring this call succeeds on the empty list. Maybe is defined in CompactStandardLibrary (compact-runtime runtime.ts from Typescript).

available from Typescript as `head(): Maybe<value_type>`

isEmpty

```
isEmpty(): Boolean
```

Returns if this List is the empty list.

available from Typescript as `isEmpty(): boolean`

length

```
length(): Uint<64>
```

Returns the number of elements contained in this List.

available from Typescript as `length(): bigint`

popFront

```
popFront(): []
```

Removes the first element from the front of this list.

pushFront

```
pushFront(value: value_type): []
```

Pushes a new element onto the front of this list.

pushFrontCoin

```
pushFrontCoin(coin: CoinInfo, recipient: Either<ZswapCoinPublicKey, ContractAddress>): []
```

Feedback

Pushes a CoinInfo onto the front of this List, where the CoinInfo is transformed into a QualifiedCoinInfo at runtime by looking up the relevant Merkle tree index. This index must have been allocated within the current transaction or this push fails. CoinInfo, ContractAddress, Either, and ZswapCoinPublicKey are defined in CompactStandardLibrary.

available only for QualifiedCoinInfo value_type

resetToDefault

```
resetToDefault(): []
```

Resets this List to the empty list.

[Symbol.iterator]

callable only from TypeScript

```
[Symbol.iterator](): Iterator<value_type>
```

Iterates over the entries in this List.

MerkleTree<nat, value_type>

This ADT is a bounded Merkle tree of depth nat containing values of type value_type.

checkRoot

```
checkRoot(rt: MerkleTreeDigest): Boolean
```

Tests if the given Merkle tree root is the root for this Merkle tree. MerkleTreeDigest is defined in CompactStandardLibrary (compact-runtime runtime.ts from Typescript).

available from Typescript as `checkRoot(rt: MerkleTreeDigest): boolean`

insert

```
insert(item: value_type): []
```

Inserts a new leaf at the first free index in this Merkle tree.

insertHash

```
insertHash(hash: Bytes<32>): []
```

Inserts a new leaf with a given hash at the first free index in this Merkle tree.

insertHashIndex

```
insertHashIndex(hash: Bytes<32>, index: Uint<64>): []
```

Inserts a new leaf with a given hash at a specific index in this Merkle tree.

insertIndex

```
insertIndex(item: value_type, index: Uint<64>): []
```

Inserts a new leaf at a specific index in this Merkle tree.

insertIndexDefault

```
insertIndexDefault(index: Uint<64>): []
```

Inserts a default value leaf at a specific index in this Merkle tree. This can be used to emulate a removal from the tree.

isFull

```
isFull(): Boolean
```

Returns if this Merkle tree is full and further items cannot be directly inserted.

available from Typescript as `isFull(): boolean`

resetToDefault

```
resetToDefault(): []
```

Resets this Merkle tree to the empty Merkle tree.

findPathForLeaf

callable only from TypeScript

```
findPathForLeaf(leaf: value_type): MerkleTreePath<value_type> | undefined
```

Finds the path for a given leaf in a Merkle tree. Be warned that this is O(n) and should be avoided for large trees. Returns undefined if no such leaf exists. MerkleTreePath is defined in compact-runtime runtime.ts.

firstFree

callable only from TypeScript

```
firstFree(): bigint
```

Retrieves the first (guaranteed) free index in the Merkle tree.

pathForLeaf

callable only from TypeScript

```
pathForLeaf(index: bigint, leaf: value_type): MerkleTreePath<value_type>
```

Returns the Merkle path, given the knowledge that a specified leaf is at the given index. It is an error to call this if this leaf is not contained at the given index. MerkleTreePath is defined in compact-runtime runtime.ts.

root

callable only from TypeScript

```
root(): MerkleTreeDigest
```

Retrieves the root of the Merkle tree. MerkleTreeDigest is defined in compact-runtime runtime.ts.

HistoricMerkleTree<nat, value_type>

This ADT is a bounded Merkle tree of depth nat containing values of type value_type, with history.

checkRoot

```
checkRoot(rt: MerkleTreeDigest): Boolean
```

Tests if the given Merkle tree root is one of the past roots for this Merkle tree. MerkleTreeDigest is defined in CompactStandardLibrary (compact-runtime runtime.ts from Typescript).

available from Typescript as `checkRoot(rt: MerkleTreeDigest): boolean`

insert

```
insert(item: value_type): []
```

Inserts a new leaf at the first free index in this Merkle tree.

insertHash

```
insertHash(hash: Bytes<32>): []
```

Inserts a new leaf with a given hash at the first free index in this Merkle tree.

insertHashIndex

```
insertHashIndex(hash: Bytes<32>, index: Uint<64>): []
```

Inserts a new leaf with a given hash at a specific index in this Merkle tree.

insertIndex

```
insertIndex(item: value_type, index: Uint<64>): []
```

Inserts a new leaf at a specific index in this Merkle tree.

insertIndexDefault

```
insertIndexDefault(index: Uint<64>): []
```

Inserts a default value leaf at a specific index in this Merkle tree. This can be used to emulate a removal from the tree.

isFull

```
isFull(): Boolean
```

Returns if this Merkle tree is full and further items cannot be directly inserted.

available from Typescript as `isFull(): boolean`

resetHistory

```
resetHistory(): []
```

Resets the history for this Merkle tree, leaving only the current root valid.

resetToDefault

```
resetToDefault(): []
```

Resets this Merkle tree to the empty Merkle tree.

findPathForLeaf

callable only from TypeScript

```
findPathForLeaf(leaf: value_type): MerkleTreePath<value_type> | undefined
```

Finds the path for a given leaf in a Merkle tree. Be warned that this is O(n) and should be avoided for large trees. Returns undefined if no such leaf exists. MerkleTreePath is defined in compact-runtime runtime.ts.

firstFree

callable only from TypeScript

```
firstFree(): bigint
```

Retrieves the first (guaranteed) free index in the Merkle tree.

history

callable only from TypeScript

```
history(): Iterator<MerkleTreeDigest>
```

An iterator over the roots that are considered valid past roots for this Merkle tree. MerkleTreeDigest is defined in compact-runtime runtime.ts.

pathForLeaf

callable only from TypeScript

```
pathForLeaf(index: bigint, leaf: value_type): MerkleTreePath<value_type>
```

Returns the Merkle path, given the knowledge that a specified leaf is at the given index. It is an error to call this if this leaf is not contained at the given index. MerkleTreePath is defined in compact-runtime runtime.ts.

root

callable only from TypeScript

```
root(): MerkleTreeDigest
```

Retrieves the root of the Merkle tree. MerkleTreeDigest is defined in compact-runtime runtime.ts.