Extensible Blockchain Object Model (XBOM)

Foundation Classes and Objects

and

Supporting Data Structures

David Beberman, CTO
Prasaga, LLC
May 3, 2019

Author	Description	Version	Date
David Beberman	Initial version	1	05/03/19

Extensible Blockchain Object Model (XBOM) Foundation Classes and Objects and Supporting Data Structures

The XBOM is a first-class object model for blockchain executable ledger entries (i.e. smart contracts). It creates a class inheritance hierarchy and object instances that promote global code reuse and simplify smart contract instantiation and use. The following descriptions assume the use of a stack based virtual machine and temporary memory address space.

Table of Contents

Definitions

Blockchain State

Blockchain state refers to a global list of entries of type Executable Ledger Entry. The global list is considered to be able to grow infinitely. All entries are persistent until explicitly deleted. The Blockchain state is virtual in the sense that the means of storing the state is implementation dependent.

Executable Ledger

The global list of Executable Ledger Entries that comprise the Blockchain State. The Executable Ledger is indexed by an address word of sufficient size to span the entire Blockchain State. An example suitable address word size is 256 bits.

Executable Ledger Entry

An Executable Ledger Entry consists of the following tuple:

{address; state reference; code/text reference; nonce}

An Executable Ledger Entry is conceptually similar to the term “smart contract” used by Blockchains such as Ethereum [ref].

State Reference

A reference to a word addressable storage space of infinite size. The state is understood to be virtual in the sense that the means of storing the state is implementation dependent. The state reference value is understood to be a collapsing function spanning the current value of all initialize words in the storage space (e.g. a merkle tree root hash)

Code/Text Reference

A reference to a word addressable storage space of infinite size for storing executable code, initialized static data, and data types and structure definitions. The code/text reference value is understood to be a collapsing function spanning the value of all the initialized works in the code/text storage space (e.g. SHA256 value).

Nonce

A monotonically incremented value that is incremented with each transaction sent to the Executable Ledger Entry

Object Table

A list of Object Entries, indexed by an Object Address.

Object Entry

An entry in an Object Table consists of the following tuple:

{Object Address; Ledger State Data Address}

Object Address

An address word that indexes an Object Table, of suitable size to span the entire Object Table.

Ledger State Storage Data Space

A word addressable storage space of infinite size, whose spanning value is the collapsing state reference stored in an Executable Ledger Entry.

XBOM Objects

All XBOM objects are referenced by unique object references. An XBOM object reference is the concatenation of a Blockchain Ledger Address, a scope table designation, and an object index.

OR = {*BLA*, *SC*, *OI*}

OR = Object Reference
BLA = Blockchain Ledger Address
SC = Scope table
OI = Object Index

By definition, is a message is sent to an *OR* = {*BLA*, NULL, NULL}, the first object in the Blockchain Global Scope Object Table is sent the message. This is normally either an instance of a ClassExecutableLedgerEntityMonitor, or a ClassExecutableLedgerEntity.

Message Dispatching

Message Dispatching – Sending a Message to an Object in a Executable Ledger Entity (I.e Smart Contract) (This causes the state address space to switch to the Executable Ledger Entity containing the Object. The code is executed in the address space of the called Executable Ledger Entity)

To dispatch a message to an object, the *OR* is uses as follows:

1: The *BLA* is used to look up the state for the storage for the Executable Ledger Entry. The VM Object State Register is set to the state from the Executable Ledger Entry. This establishes the data address space for the code execution.

2: The scope table in the state address space is selected using the *SC*.

3: The *OI* is used to index the selected scope table.

4: The *OI* state data address is set as the current object context (i.e. object self).

5: The object context clstype field's *OR* value is used to locate the class for the object.

6: The *BLA* state for the class object sets the VM Class State Register.

7: The *SC* and *OI* set the class object context: clsctx

8: The method table for the class is searched by the Class State Register → clsctx → method table, for a matching message entry.

8.1: If there is a matching message entry, call the method with the object self → result

8.2: If there isn't a matching message entry, do CallAncestor(object self) → result

9: Return result to caller.

Message Dispatching – Calling a Message on an Object in a Executable Ledger Entity (I.e Smart Contract) (The state address space is not switched. The code is executed in the address space of the calling Executable Ledger Entity)

To call a message on an object, the *OR* is uses as follows:

1: The scope table in the state address space is selected using the *SC*.
[Note: The *BLA* is not used to change the state address space.]

2: The *OI* is used to index the selected scope table.

3: The *OI* state data address is set as the current object context (i.e. object self).

4: The object context clstype field's *OR* value is used to locate the class for the object.

5: The *BLA* state for the class object sets the VM Class State Register.

6: The *SC* and *OI* set the class object context: clsctx

7: The method table for the class is searched by the Class State Register → clsctx → method table, for a matching message entry.

8.1: If there is a matching message entry, call the method with the object self → result

8.2: If there isn't a matching message entry, do CallAncestor(object self) → result

9: Return result

Message Dispatching – Calling the ancestor class on an Object in an Executable Ledger Entity (I.e Smart Contract)
(The state address space is not switched. The code is executed in the address space of the calling Executable Ledger Entity)

1: The scope table in the state address space is selected using the *SC*.

[Note: The *BLA* is not used to change the state address space.]

2: The *OI* is used to index the selected scope table.

3: The *OI* state data address is set as the current object context (i.e. object self).

4: The object context superclass field's *OR* value is used to locate the class for the object.

5: The *BLA* state for the class object sets the VM Class State Register.

6: The *SC* and *OI* set the class object context: clsctx

7: The method table for the class is searched by the Class State Register → clsctx → method table, for a matching message entry.

8.1: If there is a matching message entry, call the method with the object self → result

8.2: If there isn't a matching message entry, do CallAncestor(object self) → result

9: Return result

XBOM Foundation Classes Definitions

There is only one Class XBOM Object and one Class Class.
All objects inherit from Class Object
All classes are instances of Class Class.
All metaclasses inherit from Class Class
To use the metaclass methods, a class must be an instance of the metaclass – which inherits from ClassClass.

The two instances, Class XBOM Object and Class Class are defined to exist, and are never created or destroyed.

Class XBOM Object

Fields

> classtype = NULL – can not send messages directly to Class XBOM Object, only subclasses
> mtdtable = Class XBOM Object methods: new, createobject, createfields, newfield, init, delete
> super = NULL

Class Class

Fields

classtype = Class XBOM Object
mtdtable = Class Class methods: new, createfields, init, delete
super = Class XBOM Object

Class XBOM Object does not inherit from any other objects, therefore its super is NULL.

Methods

Class Class New – creates a new object from a class (instance of Class Class)

1: self.createobject – calls ancestor Class XBOM Object.createobject, returns newobject

2: self.createfields(newobject) – calls Class Class.createfields for newobject. Creates the fields from each class

3. newobject.classtype = self – sets the class type for the new object

4. newobject.init (params)

5. return newobject

Class XBOM Object New – creates a new class (new classname, ancestor class object ID)

1: self.createObject – calls Class XBOM Object.createobject, returns newobject
2: self.creatfields(newobject) – Class XBOM Object createfields

3: newobject.clstype = self

4: newobject.createfield (super, mtd, fieldlist)

5: newobject.super = ancestor

6: newobject.mtdtable = classname mtdtable – from code/text address space

7: newobject.fieldlist = classname fieldlist – from code/text address space

8: newobject.init (params)

9: return newobject

Creating New Executable Ledger Entities (i.e. smart contracts)

The foundation classes for creating new ledger entities are:

ClassExecutableMetaLedgerEntity – inherits from ClassClass, creates new Ledger Entries.

ClassExecutableLedgerEntityMgr – inherits from ClassExecutableMetaLedgerEntity, creates new ClassExecutableLedgerEntityMonitors

ClassExecutableLedgerEntityMonitor – inherits from ClassMetaExecutableLedgerEntity, creates new ClassExecutableLedgerEntity instances

ClassExecutableLedgerEntity – the root class for all Executable Ledger Entity classes

The relationship between the classes and object instances are show in the attached diagrams.

ClassExecutableMetaLedgerEntity

Inherits from Class Class.
Creates new Ledger Entries.

Methods

ClassExecutableMetaLedgerEntity New (classname, code & text reference, ancestor class)

1: newLEAddress = self.NewLE(code & text, new state)

2: Changes the VM Register State to the new state value

3: Call ancestor with the classname and ancestor class: self.CallAncestor(classname, ancestor class)
- classname is the main class in the referenced code&text data
  - returns the new object reference

4: resets the VM Register state

5: returns the new object reference

ClassExecutableMetaLedgerEntity NewLE (code & text, new state)

1: newLEAddress = create new LE entry

2: Sets newLEAddress.code&text = code & text

3: Sets newLEAddress.nonce=1

4: Sets newLEAddress.state=empty value

5: new state = newLEAddress.state

6: returns newLEAddress and new state

ClassExecutableLedgerEntityMgr

Inherits from ClassExecutableMetaLedgerEntity to create new Executable Ledger Entries.
Each new entry contains an instance of ClassExecutableLedgerEntityMonitor or subclass.
This instance contains a reference to a ClassExecutableLedgerEntity subclass to be used to create new Executable Ledger Entities of the Class Executable Ledger Entity type.

Methods

ClassExecutableLedgerEntityMgr New (classname, code & text, ancestor class)

1: New ledger entry = self.NewLE(code & text, new state)

2: Get LedgerEntityMonitor class:
LedgerEntityMonitorClass = self.getLedgerEntityMonitorClass

Comment: subclasses of Class LedgerEntityMgr override getLedgerEntityMonitorClass and return a different class.

3: Change VM Register State to new state

4: LedgerEntityMonitor Object = LedgerEntityMonitorClass.new – create new monitor object

5: NewLedgerEntityClass = ClassClass.new (classname, ancestor class) – create new Ledger Entity Class

6: LedgerEntityMonitor Object.LedgerEntityClass = NewLedgerEntityClass – store the class

7: LedgerEntityMonitorObject.createClasses() - creates any additional classes that the LedgerEntityClass may use for its instances' operation.

8: Return LedgerEntityMonitorObject.

ClassExecutableLedgerEntityMonitor

Inherits from ClassExecutableMetaLedgerEntity
Creates new Executable Ledger Entity instances.

Creates new object of type ClassExecutableLedgerEntity subclass in the new Ledger Entity instance.

Methods

ClassExecutableLedgerEntityMonitor New (params)

1: NewLEAddress = self.NewLE (null, new state) – no code in instances of a LedgerEntity (i.e. smart contract instances)

2: local objectref = self.LedgerEntityClass

3: Change VM Register State to new state

4: newobjectreference = objectref.new (params) – creates new object instance of LedgerEntityClass

5: restore state

6: return newobjectreference – a new smart contract instance is now created and ready to receive messages.

ClassExecutableLedgerEntity

Parent class for all Executable Ledger Entity instances (i.e. instances of smart contracts) ClassExecutableLedgerEntity instances receive all transaction messages from user accounts.

Methods

ClassExecutableLedgerEntity Init (params)

Subclasses initialize their state to the parameters passed in.

Other methods

Specific behavior to each subclass

XBOM Class and Object Diagrams

Blockchain Ledger Entry Tuple

Address	State Reference	State Nonce	Code & Text	Optional parameters

Blockchain Ledger State

Address	State Reference	State Nonce	Code & Text	Optional parameters
Address	State Reference	State Nonce	Code & Text	Optional parameters
Address	State Reference	State Nonce	Code & Text	Optional parameters

Object Entry Type

Object Address	Ledger State Data Address

Ledger Object Table Type

Object Address	Ledger State Data Address
Object Address	Ledger State Data Address
Object Address	Ledger State Data Address

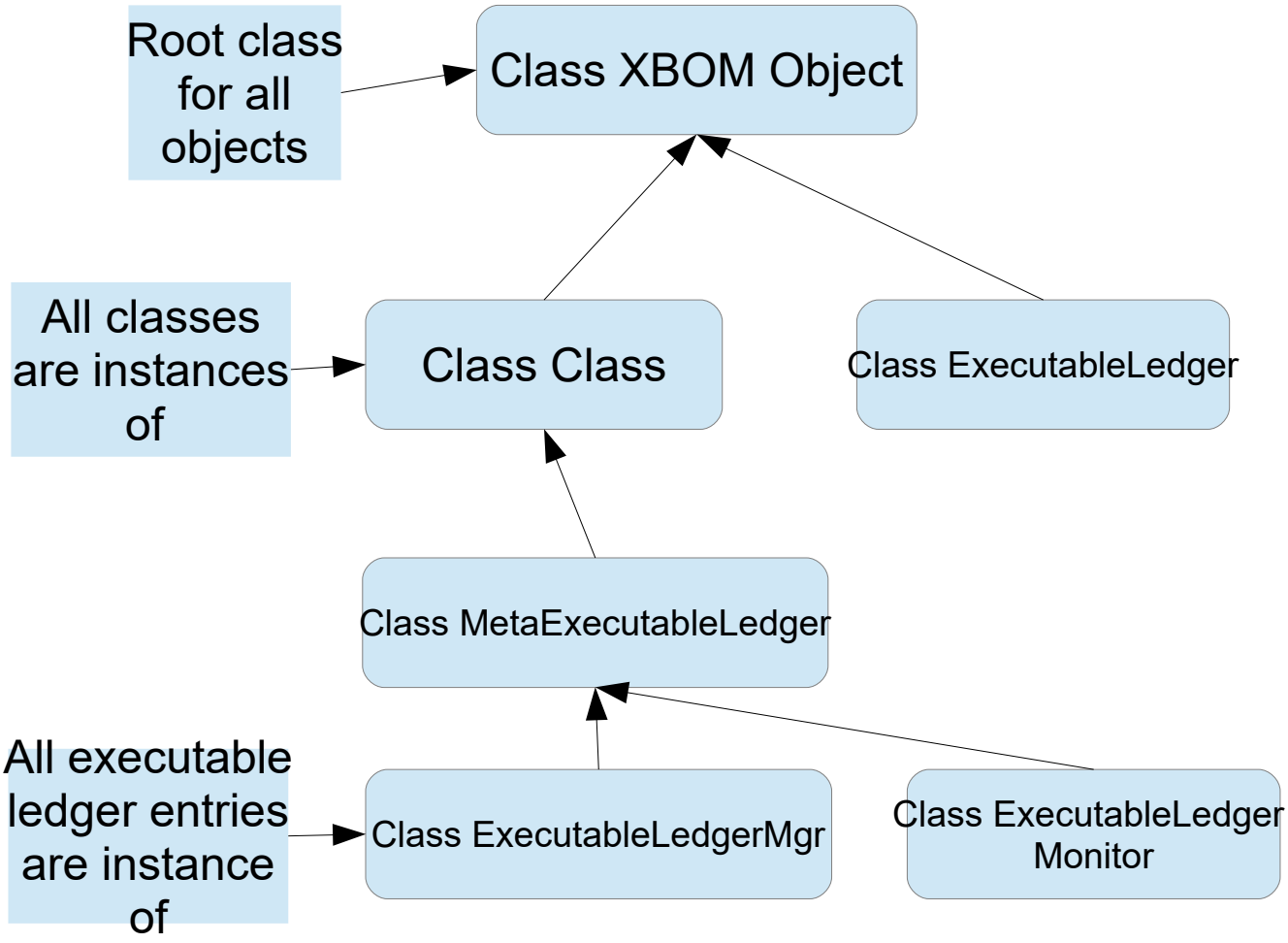
Ledger State Storage Data Space

Data Word Address	Data Word Value
Data Word Address	Data Word Value
Data Word Address	Data Word Value

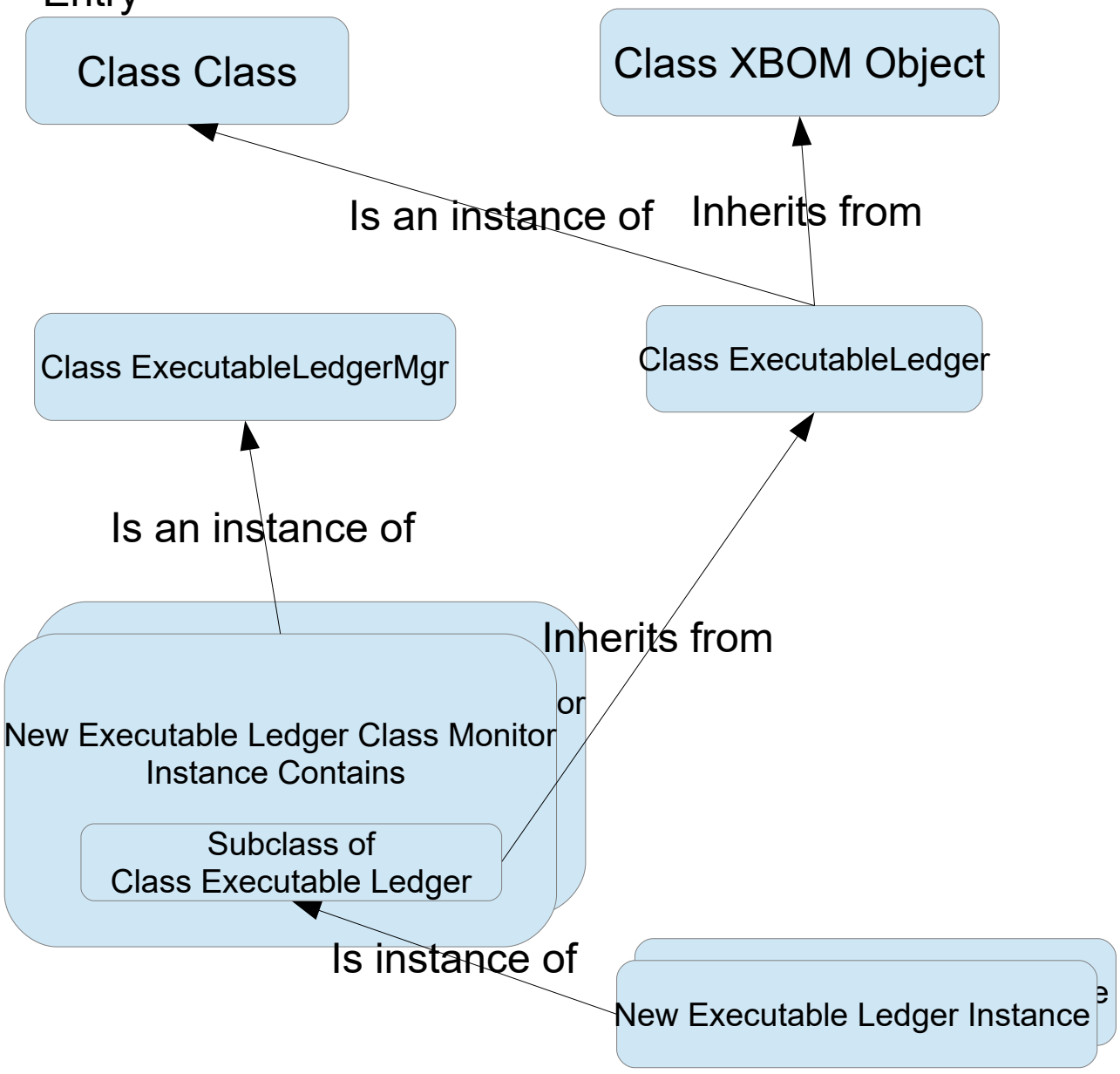
Ledger State Storage Object Tables

Blockchain Global Scope Objects Table
Ledger Global Scope Objects Table
Ledger Private Scope Objects Table

Blockchain Object Model Foundation Classes and Inheritance



Instance tree for an instantiated Executable Ledger Entry



Every executable ledger manager is an instance of ClassExecutableLedgerMgr or a subclass.

Every executable ledger entry object is an instance of Class Executable Ledger or a subclass.