

NEO Community Bluepaper\*

NEO Persistable Classes: An Efficient Object-Oriented Framework for C#.NEO Smart Contract Development

mwherman2000/neo-persistibleclasses  
  
Version 1.0  
February 2018

Prepared for  
City of Zion #2 dApp Competition  
Closing date: February 25, 2018

Submitted by  
Michael Herman  
@mwherman2000

[mwherman@parallelspace.net](mailto:mwherman@parallelspace.net)

\* For more information on bluepapers, read Appendix A – What is a Bluepaper? on 31

#### License

/\*

MIT License

Copyright (c) 2018 Michael Herman (mwherman@parallelspace.net)

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

\*/

#### Table of Contents

[List of Figures 5](#_Toc506981408)

[List of Tables 6](#_Toc506981409)

[1 Purpose 7](#_Toc506981410)

[1.1 Goals 7](#_Toc506981411)

[1.2 Principles 7](#_Toc506981412)

[1.3 Drivers 7](#_Toc506981413)

[2 Scenario 8](#_Toc506981414)

[3 Problem 11](#_Toc506981415)

[4 Analysis 12](#_Toc506981416)

[5 Options 13](#_Toc506981417)

[6 Solution 14](#_Toc506981418)

[6.1 NPC Level 0 Fields Only 14](#_Toc506981419)

[6.2 NPC Level 1 Managed 14](#_Toc506981420)

[6.3 NPC Level 2 Persistable 17](#_Toc506981421)

[6.4 NPC Level 3 Deletable (Bury/Tombstone) 22](#_Toc506981422)

[6.5 NPC Level 4 Collectible 22](#_Toc506981423)

[6.6 Future Implementation Levels 23](#_Toc506981424)

[7 Results 24](#_Toc506981425)

[8 Roadmap 25](#_Toc506981426)

[9 Summary 26](#_Toc506981427)

[10 Appendix A – What is a Bluepaper? 27](#_Toc506981428)

[10.1 References 27](#_Toc506981429)

[11 Appendix B - NEO Blockchain Architecture Reference Model (ARM) 28](#_Toc506981430)

[11.1 NEO Developer Environment 28](#_Toc506981431)

[11.2 NEO Virtual Machine Environment 30](#_Toc506981432)

[12 About the Author 35](#_Toc506981433)

[12.1 Michael Herman 35](#_Toc506981434)

# List of Figures

[Figure 1. Sukhoi SuperJet Full Flight Simulator [credit: Wikipedia] 6](#_Toc506928966)

[Figure 2. NEO Blockchain Architecture Reference Model 8](#_Toc506928967)

[Figure 3. NEO Developer Environment Architecture 9](#_Toc506928968)

[Figure 4. NEO Virtual Machine Architecture 11](#_Toc506928969)

# List of Tables

[Table 1. NEO Developer Environment Component Descriptions 9](#_Toc506928970)

[Table 2. NEO Virtual Machine Component Descriptions 12](#_Toc506928971)

1. Purpose

The purpose of this bluepaper is document the NEO Persistable Class (NPC) framework that enables efficient object-oriented smart contract development using C#.NEO, Visual Studio and the NEO Developer Tools to build distributed apps (dApps) on the NEO Blockchain.

The audience for the document is Microsoft .NET developers and architects who want to know the detailed differences between C#.NEO, the subset of the C# 6.0 language supported by neon.exe, the NEO C# smart contract transcompiler.

In this document C#.MS will be used to represent the version of the C# 6.0 language supported in Visual Studio 2017 (Community Edition) – the integrated development environment used for this project. C#.NEO will be used to refer to the subset of CS#.MS that is supported by the NEO Virtual Machine (NEO VM) as January 2018.

* 1. Goals

TODO

* 1. Principles

TODO

* 1. Drivers

TODO

1. Scenario

When you're learning to write NEO smart contracts, you're really learning how to program the NEO VM to do what you want; for example,

* perform a calculation
* read persisted data (state) from the blockchain
* add new data (state) to the blockchain
* verify someone's identity (and, in turn, determine which roles and permissions they have with respect to the purpose and goals of your smart contract)
* etc.

As an analogy, learning to program the NEO VM is like learning to fly a flight simulator.



Figure . Sukhoi SuperJet Full Flight Simulator [credit: [Wikipedia](https://en.wikipedia.org/wiki/Full_flight_simulator#/media/File:SSJ100_FFS_1_(9318513805).jpg)]

In a flight simulator cockpit, the pilot is manipulating the controls (foot pedals, switches and knobs as well as the control column) and the simulator, in turn, receives those stimuli and causes multiple subsystems to respond and create an illusion that you are controlling a sophisticated aircraft. The subsystems responsible for creating these illusions include visual displays, audio cues and sound effects, hydraulic actuators, dials and other instruments.

A NEO smart contract running in the NEO VM is like the pilot in control of the flight simulator. The smart contract is performing various calculations and calling various virtual machine APIs to accomplish the smart contract's purpose and goals. In response to these API calls, the VM is interacting with its various subsystems (just like the flight simulator). In the case of the NEO VM, these subsystems include:

* Account management
* Asset management
* Block management
* Blockchain management
* Contract management
* Header management
* Runtime management
* Storage management
* Transaction management
* Attribute management
* Input management
* Output management
* Validation management

...in addition to the basic execution engine capabilities [[NEO Smart Contract Tutorial](http://docs.neo.org/en-us/sc/tutorial.html)] such as:

* Flow control
* Stack operations (including parameter passing)
* Bit operations
* Arithmetic operations
* Logical operations
* Data type conversion
* Cryptographic operations
* Etc.

As a smart contract developer, it is your responsibility to learn and understand the purpose and goals each of the VM subsystems, the use cases they support, and best practices and programming patterns to use when developing best-in-class smart contacts.

1. Problem

TODO

1. Analysis

TODO

1. Options

TODO

1. Solution

The proposed solution is called NEO Persistable Classes (NPCs). There are a number of different levels when it comes to applying the NPC Framework to classes in C#.NEO. There are:

* NPC Level 0 Fields Only
* NPC Level 1 Managed
* NPC Level 2 Persistable
* NPC Level 3 Deletable (Bury/Tombstone)
* NPC Level 4 Collectible
* NPC Level 5 Extendible (Roadmap)
* NPC Level 6 Authorized (Roadmap)
  1. NPC Level 0 Fields Only

NPC Level 0 Fields Only refers to the definition of a simple class consisting of field definitions only.

public class Point /\* Level 0 Fields Only \*/

{

private BigInteger \_x;

private BigInteger \_y;

}

* 1. NPC Level 1 Managed

NPC Level 1 Managed refers the basic design and implementation requirements for a C#.NEO class to be an NPC and in a very practical sense, for classes to be useful at all for development simple as well as complex or more elaborate classes in an object-oriented paradigm. Functional capabilities include:

* New() – creating new entities
* Initialize() – initializing new entity field values (internal functionality)
* Null() – creating Null entities
* IsNull() – test to see if an entity is in a Null state
* Log() – Logging of an entity’s core field values
* LogExt() – Logging of entity’s core and system field values

TODO

public class NeoTrace /\* Level \*all\* \*/

{

public static void Trace(params object[] args)

{

Neo.SmartContract.Framework.Services.Neo.Runtime.Notify(args);

}

}

TODO

public static class NeoEntityModel /\* Level 1 Managed \*/

{

public enum EntityState

{

NULL,

INIT,

SET

}

public static BigInteger AsBigInteger(this EntityState state)

{

int istate = (int)state;

BigInteger bis = istate;

return bis;

}

public static EntityState BytesToEntityState(byte[] bsta)

{

int ista = (int)bsta.AsBigInteger();

NeoEntityModel.EntityState sta = (NeoEntityModel.EntityState)ista;

return sta;

}

}

TODO

public class Point : NeoTrace /\* Level 1 Managed \*/

{

private BigInteger \_x;

private BigInteger \_y;

private NeoEntityModel.EntityState \_state;

// Accessors

public static void SetX(Point p, BigInteger value)

{ p.\_x = value; p.\_state = NeoEntityModel.EntityState.SET; }

public static BigInteger GetX(Point p) { return p.\_x; }

public static void SetY(Point p, BigInteger value)

{ p.\_y = value; p.\_state = NeoEntityModel.EntityState.SET; }

public static BigInteger GetY(Point p) { return p.\_y; }

public static void Set(Point p, BigInteger xvalue, BigInteger yvalue)

{ p.\_x = xvalue; p.\_y = yvalue; p.\_state = NeoEntityModel.EntityState.SET; }

// Factory methods

private Point()

{

}

private static Point \_Initialize(Point p)

{

p.\_x = 0;

p.\_y = 0;

p.\_state = NeoEntityModel.EntityState.NULL; ;

LogExt("\_Initialize(p).p", p);

return p;

}

public static Point New()

{

Point p = new Point();

\_Initialize(p);

LogExt("New().p", p);

return p;

}

public static Point New(int x, int y)

{

Point p = new Point();

p.\_x = x;

p.\_y = y;

p.\_state = NeoEntityModel.EntityState.INIT;

LogExt("New(x,y).p", p);

return p;

}

public static Point Null()

{

Point p = new Point();

\_Initialize(p);

LogExt("Null().p", p);

return p;

}

// EntityState wrapper methods

public static bool IsNull(Point p)

{

return (p.\_state == NeoEntityModel.EntityState.NULL);

}

// Log/trace methods

public static void Log(string label, Point p)

{

NeoTrace.Trace(label, p.\_x, p.\_y);

}

public static void LogExt(string label, Point p)

{

NeoTrace.Trace(label, p.\_x, p.\_y, p.\_state);

}

}

TODO

* 1. NPC Level 2 Persistable

NPC Level 2 Persistable support includes the ability for an entity to serialize and deserialize itself into NEO Storage as well as support deletion (using a form of tombstoning). Functional capabilities at this level include:

* Put(string key), Put(byte[] key)
* Get(string key), Get(byte[] key)
* IsMissing()

TODO

public static class NeoEntityModel /\* Level 2 Persistable \*/

{

public enum EntityState

{

NULL,

INIT,

SET,

PUTTED,

GETTED,

MISSING

}

public static BigInteger AsBigInteger(this EntityState state)

{

int istate = (int)state;

BigInteger bis = istate;

return bis;

}

public static EntityState BytesToEntityState(byte[] bsta)

{

int ista = (int)bsta.AsBigInteger();

NeoEntityModel.EntityState sta = (NeoEntityModel.EntityState)ista;

return sta;

}

}

TODO

public class Point : NeoTrace /\* Level 2 Persistable \*/

{

private BigInteger \_x;

private BigInteger \_y;

private NeoEntityModel.EntityState \_state;

// Accessors

public static void SetX(Point p, BigInteger value)

{ p.\_x = value; p.\_state = NeoEntityModel.EntityState.SET; }

public static BigInteger GetX(Point p) { return p.\_x; }

public static void SetY(Point p, BigInteger value)

{ p.\_y = value; p.\_state = NeoEntityModel.EntityState.SET; }

public static BigInteger GetY(Point p) { return p.\_y; }

public static void Set(Point p, BigInteger xvalue, BigInteger yvalue)

{ p.\_x = xvalue; p.\_y = yvalue; p.\_state = NeoEntityModel.EntityState.SET; }

// Class name and property names

private const string \_className = "Point";

private const string \_sX = "X";

private const string \_sY = "Y";

private const string \_sSTA = "\_STA";

private const string \_sEXT = "\_EXT";

private static readonly byte[] \_bX = Helper.AsByteArray(\_sX);

private static readonly byte[] \_bY = Helper.AsByteArray(\_sY);

private static readonly byte[] \_bSTA = Helper.AsByteArray(\_sSTA);

private static readonly byte[] \_bEXT = Helper.AsByteArray(\_sEXT);

// Internal fields

private const string \_classKeyTag = "/#" + \_className + ".";

private static readonly byte[] \_bclassKeyTag = Helper.AsByteArray(\_classKeyTag);

// Factory methods

private Point()

{

}

private static Point \_Initialize(Point p)

{

p.\_x = 0;

p.\_y = 0;

p.\_state = NeoEntityModel.EntityState.NULL; ;

LogExt("\_Initialize(p).p", p);

return p;

}

public static Point New()

{

Point p = new Point();

\_Initialize(p);

LogExt("New().p", p);

return p;

}

public static Point New(int x, int y)

{

Point p = new Point();

p.\_x = x;

p.\_y = y;

p.\_state = NeoEntityModel.EntityState.INIT;

LogExt("New(x,y).p", p);

return p;

}

public static Point Null()

{

Point p = new Point();

\_Initialize(p);

LogExt("Null().p", p);

return p;

}

// EntityState wrapper methods

public static bool IsNull(Point p)

{

return (p.\_state == NeoEntityModel.EntityState.NULL);

}

// Log/trace methods

public static void Log(string label, Point p)

{

NeoTrace.Trace(label, p.\_x, p.\_y);

}

public static void LogExt(string label, Point p)

{

NeoTrace.Trace(label, p.\_x, p.\_y, p.\_state);

}

// Persistable methods

public static bool IsMissing(Point p)

{

return (p.\_state == NeoEntityModel.EntityState.MISSING);

}

public static Point Missing()

{

Point p = new Point();

p.\_x = 0;

p.\_y = 0;

p.\_state = NeoEntityModel.EntityState.MISSING;

LogExt("Missing().p", p);

return p;

}

public static bool Put(Point p, byte[] key)

{

if (key.Length == 0) return false;

Neo.SmartContract.Framework.Services.Neo.StorageContext ctx = Neo.SmartContract.Framework.Services.Neo.Storage.CurrentContext;

byte[] \_bkeyTag = Helper.Concat(key, \_bclassKeyTag);

p.\_state = NeoEntityModel.EntityState.PUTTED;

/\*STA\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, Helper.Concat(\_bkeyTag, \_bSTA), p.\_state.AsBigInteger());

/\*FIELD\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, Helper.Concat(\_bkeyTag, \_bX), p.\_x);

/\*FIELD\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, Helper.Concat(\_bkeyTag, \_bY), p.\_y);

LogExt("Put(bkey).p", p);

return true;

}

public static bool Put(Point p, string key)

{

if (key.Length == 0) return false;

LogExt("Put(ks).p", p);

Neo.SmartContract.Framework.Services.Neo.StorageContext ctx = Neo.SmartContract.Framework.Services.Neo.Storage.CurrentContext;

string \_skeyTag = key + \_classKeyTag;

Trace("Put(ks).\_skeyTag", \_skeyTag);

p.\_state = NeoEntityModel.EntityState.PUTTED;

BigInteger bis = p.\_state.AsBigInteger();

Trace("Put(ks).bis", bis);

/\*STA\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, \_skeyTag + \_sSTA, bis);

/\*FIELD\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, \_skeyTag + \_sX, p.\_x);

/\*FIELD\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, \_skeyTag + \_sY, p.\_y);

LogExt("Put(ks).p", p);

return true;

}

public static Point Get(byte[] key)

{

if (key.Length == 0) return Null();

Neo.SmartContract.Framework.Services.Neo.StorageContext ctx = Neo.SmartContract.Framework.Services.Neo.Storage.CurrentContext;

byte[] \_bkeyTag = Helper.Concat(key, \_bclassKeyTag);

Point p;

/\*STA\*/

byte[] bsta = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, Helper.Concat(\_bkeyTag, \_bSTA));

NeoTrace.Trace("Get(kb).bs", bsta.Length, bsta);

if (bsta.Length == 0)

{

p = Point.Missing();

}

else // not MISSING

{

/\*EXT\*/

byte[] bext = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, Helper.Concat(\_bkeyTag, \_bEXT));

int ista = (int)bsta.AsBigInteger();

NeoEntityModel.EntityState sta = (NeoEntityModel.EntityState)ista;

p = new Point();

/\*FIELD\*/

BigInteger x = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, Helper.Concat(\_bkeyTag, \_bX)).AsBigInteger();

/\*FIELD\*/

BigInteger y = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, Helper.Concat(\_bkeyTag, \_bY)).AsBigInteger();

p.\_x = x;

p.\_y = y;

p.\_state = sta;

p.\_state = NeoEntityModel.EntityState.GETTED; /\* OVERRIDE \*/

}

LogExt("Get(kb).p", p);

return p;

}

public static Point Get(string key)

{

if (key.Length == 0) return Null();

Neo.SmartContract.Framework.Services.Neo.StorageContext ctx = Neo.SmartContract.Framework.Services.Neo.Storage.CurrentContext;

string \_skeyTag = key + \_classKeyTag;

Point p;

/\*STA\*/

byte[] bsta = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, \_skeyTag + \_sSTA);

NeoTrace.Trace("Get(ks).bs", bsta.Length, bsta);

if (bsta.Length == 0)

{

p = Point.Missing();

}

else // not MISSING

{

/\*EXT\*/

byte[] bext = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, \_skeyTag + \_sEXT);

int ista = (int)bsta.AsBigInteger();

NeoEntityModel.EntityState sta = (NeoEntityModel.EntityState)ista;

p = new Point();

/\*FIELD\*/

BigInteger x = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, \_skeyTag + \_sX).AsBigInteger();

/\*FIELD\*/

BigInteger y = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, \_skeyTag + \_sY).AsBigInteger();

NeoTrace.Trace("Get(ks).x,y", x, y);

p.\_x = x;

p.\_y = y;

p.\_state = sta;

p.\_state = NeoEntityModel.EntityState.GETTED; /\* OVERRIDE \*/

}

LogExt("Get(ks).p", p);

return p;

}

}

TODO

* 1. NPC Level 3 Deletable (Bury/Tombstone)

NPC Level 3 Deletable support includes the ability to delete (bury) an entity that already exists in NEO Storage (using a form of tombstoning). Functional capabilities at this level include:

* Bury(string key), Bury(byte[] key)
* IsBuried()

TODO

public static class NeoEntityModel /\* Level 3 Deletable (Bury/Tombstone) \*/

{

public enum EntityState

{

NULL,

INIT,

SET,

PUTTED,

GETTED,

MISSING,

TOMBSTONED

}

public static BigInteger AsBigInteger(this EntityState state)

{

int istate = (int)state;

BigInteger bis = istate;

return bis;

}

public static EntityState BytesToEntityState(byte[] bsta)

{

int ista = (int)bsta.AsBigInteger();

NeoEntityModel.EntityState sta = (NeoEntityModel.EntityState)ista;

return sta;

}

}

TODO

public class Point : NeoTrace /\* Level 3 Deletable (Bury/Tombstone) \*/

{

private BigInteger \_x;

private BigInteger \_y;

private NeoEntityModel.EntityState \_state;

// Accessors

public static void SetX(Point p, BigInteger value)

{ p.\_x = value; p.\_state = NeoEntityModel.EntityState.SET; }

public static BigInteger GetX(Point p) { return p.\_x; }

public static void SetY(Point p, BigInteger value)

{ p.\_y = value; p.\_state = NeoEntityModel.EntityState.SET; }

public static BigInteger GetY(Point p) { return p.\_y; }

public static void Set(Point p, BigInteger xvalue, BigInteger yvalue)

{ p.\_x = xvalue; p.\_y = yvalue; p.\_state = NeoEntityModel.EntityState.SET; }

// Class name and property names

private const string \_className = "Point";

private const string \_sX = "X";

private const string \_sY = "Y";

private const string \_sSTA = "\_STA";

private const string \_sEXT = "\_EXT";

private static readonly byte[] \_bX = Helper.AsByteArray(\_sX);

private static readonly byte[] \_bY = Helper.AsByteArray(\_sY);

private static readonly byte[] \_bSTA = Helper.AsByteArray(\_sSTA);

private static readonly byte[] \_bEXT = Helper.AsByteArray(\_sEXT);

// Internal fields

private const string \_classKeyTag = "/#" + \_className + ".";

private static readonly byte[] \_bclassKeyTag = Helper.AsByteArray(\_classKeyTag);

// Factory methods

private Point()

{

}

private static Point \_Initialize(Point p)

{

p.\_x = 0;

p.\_y = 0;

p.\_state = NeoEntityModel.EntityState.NULL; ;

LogExt("\_Initialize(p).p", p);

return p;

}

public static Point New()

{

Point p = new Point();

\_Initialize(p);

LogExt("New().p", p);

return p;

}

public static Point New(int x, int y)

{

Point p = new Point();

p.\_x = x;

p.\_y = y;

p.\_state = NeoEntityModel.EntityState.INIT;

LogExt("New(x,y).p", p);

return p;

}

public static Point Null()

{

Point p = new Point();

\_Initialize(p);

LogExt("Null().p", p);

return p;

}

// EntityState wrapper methods

public static bool IsNull(Point p)

{

return (p.\_state == NeoEntityModel.EntityState.NULL);

}

// Log/trace methods

public static void Log(string label, Point p)

{

NeoTrace.Trace(label, p.\_x, p.\_y);

}

public static void LogExt(string label, Point p)

{

NeoTrace.Trace(label, p.\_x, p.\_y, p.\_state);

}

// Persistable methods

public static bool IsMissing(Point p)

{

return (p.\_state == NeoEntityModel.EntityState.MISSING);

}

public static Point Missing()

{

Point p = new Point();

p.\_x = 0;

p.\_y = 0;

p.\_state = NeoEntityModel.EntityState.MISSING;

LogExt("Missing().p", p);

return p;

}

public static bool Put(Point p, byte[] key)

{

if (key.Length == 0) return false;

Neo.SmartContract.Framework.Services.Neo.StorageContext ctx = Neo.SmartContract.Framework.Services.Neo.Storage.CurrentContext;

byte[] \_bkeyTag = Helper.Concat(key, \_bclassKeyTag);

p.\_state = NeoEntityModel.EntityState.PUTTED;

/\*STA\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, Helper.Concat(\_bkeyTag, \_bSTA), p.\_state.AsBigInteger());

/\*FIELD\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, Helper.Concat(\_bkeyTag, \_bX), p.\_x);

/\*FIELD\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, Helper.Concat(\_bkeyTag, \_bY), p.\_y);

LogExt("Put(bkey).p", p);

return true;

}

public static bool Put(Point p, string key)

{

if (key.Length == 0) return false;

LogExt("Put(ks).p", p);

Neo.SmartContract.Framework.Services.Neo.StorageContext ctx = Neo.SmartContract.Framework.Services.Neo.Storage.CurrentContext;

string \_skeyTag = key + \_classKeyTag;

Trace("Put(ks).\_skeyTag", \_skeyTag);

p.\_state = NeoEntityModel.EntityState.PUTTED;

BigInteger bis = p.\_state.AsBigInteger();

Trace("Put(ks).bis", bis);

/\*STA\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, \_skeyTag + \_sSTA, bis);

/\*FIELD\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, \_skeyTag + \_sX, p.\_x);

/\*FIELD\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, \_skeyTag + \_sY, p.\_y);

LogExt("Put(ks).p", p);

return true;

}

public static Point Get(byte[] key)

{

if (key.Length == 0) return Null();

Neo.SmartContract.Framework.Services.Neo.StorageContext ctx = Neo.SmartContract.Framework.Services.Neo.Storage.CurrentContext;

byte[] \_bkeyTag = Helper.Concat(key, \_bclassKeyTag);

Point p;

/\*STA\*/

byte[] bsta = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, Helper.Concat(\_bkeyTag, \_bSTA));

NeoTrace.Trace("Get(kb).bs", bsta.Length, bsta);

if (bsta.Length == 0)

{

p = Point.Missing();

}

else // not MISSING

{

/\*EXT\*/

byte[] bext = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, Helper.Concat(\_bkeyTag, \_bEXT));

int ista = (int)bsta.AsBigInteger();

NeoEntityModel.EntityState sta = (NeoEntityModel.EntityState)ista;

p = new Point();

/\*FIELD\*/

BigInteger x = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, Helper.Concat(\_bkeyTag, \_bX)).AsBigInteger();

/\*FIELD\*/

BigInteger y = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, Helper.Concat(\_bkeyTag, \_bY)).AsBigInteger();

p.\_x = x;

p.\_y = y;

p.\_state = sta;

p.\_state = NeoEntityModel.EntityState.GETTED; /\* OVERRIDE \*/

}

LogExt("Get(kb).p", p);

return p;

}

public static Point Get(string key)

{

if (key.Length == 0) return Null();

Neo.SmartContract.Framework.Services.Neo.StorageContext ctx = Neo.SmartContract.Framework.Services.Neo.Storage.CurrentContext;

string \_skeyTag = key + \_classKeyTag;

Point p;

/\*STA\*/

byte[] bsta = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, \_skeyTag + \_sSTA);

NeoTrace.Trace("Get(ks).bs", bsta.Length, bsta);

if (bsta.Length == 0)

{

p = Point.Missing();

}

else // not MISSING

{

/\*EXT\*/

byte[] bext = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, \_skeyTag + \_sEXT);

int ista = (int)bsta.AsBigInteger();

NeoEntityModel.EntityState sta = (NeoEntityModel.EntityState)ista;

p = new Point();

/\*FIELD\*/

BigInteger x = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, \_skeyTag + \_sX).AsBigInteger();

/\*FIELD\*/

BigInteger y = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, \_skeyTag + \_sY).AsBigInteger();

NeoTrace.Trace("Get(ks).x,y", x, y);

p.\_x = x;

p.\_y = y;

p.\_state = sta;

p.\_state = NeoEntityModel.EntityState.GETTED; /\* OVERRIDE \*/

}

LogExt("Get(ks).p", p);

return p;

}

// Deletable methods

public static bool IsBuried(Point p)

{

return (p.\_state == NeoEntityModel.EntityState.TOMBSTONED);

}

public static Point Tombstone()

{

Point p = new Point();

p.\_x = 0;

p.\_y = 0;

p.\_state = NeoEntityModel.EntityState.TOMBSTONED;

LogExt("Tombstone().p", p);

return p;

}

public static Point Bury(byte[] key)

{

if (key.Length == 0) return Null(); // TODO - create NeoEntityModel.EntityState.BADKEY?

Neo.SmartContract.Framework.Services.Neo.StorageContext ctx = Neo.SmartContract.Framework.Services.Neo.Storage.CurrentContext;

byte[] \_bkeyTag = Helper.Concat(key, \_bclassKeyTag);

Point p;

/\*STA\*/

byte[] bsta = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, Helper.Concat(\_bkeyTag, \_bSTA));

NeoTrace.Trace("Bury(kb).bs", bsta.Length, bsta);

if (bsta.Length == 0)

{

p = Point.Missing();

}

else // not MISSING - bury it

{

p = Point.Tombstone(); // TODO - should Bury() preserve the exist field values or re-initialize them? Preserve is cheaper but not as private

/\*STA\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, Helper.Concat(\_bkeyTag, \_bSTA), p.\_state.AsBigInteger());

/\*FIELD\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, Helper.Concat(\_bkeyTag, \_bX), p.\_x);

/\*FIELD\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, Helper.Concat(\_bkeyTag, \_bY), p.\_y);

}

LogExt("Bury(kb).p", p);

return p; // return Point p to signal if key is Missing or bad key

}

public static Point Bury(string key)

{

if (key.Length == 0) return Null(); // TODO - create NeoEntityModel.EntityState.BADKEY?

Neo.SmartContract.Framework.Services.Neo.StorageContext ctx = Neo.SmartContract.Framework.Services.Neo.Storage.CurrentContext;

string \_skeyTag = key + \_classKeyTag;

Point p;

/\*STA\*/

byte[] bsta = Neo.SmartContract.Framework.Services.Neo.Storage.Get(ctx, \_skeyTag + \_sSTA);

NeoTrace.Trace("Bury(ks).bs", bsta.Length, bsta);

if (bsta.Length == 0)

{

p = Point.Missing();

}

else // not MISSING - bury it

{

p = Point.Tombstone(); // TODO - should Bury() preserve the exist field values or re-initialize them? Preserve is cheaper but not as private

/\*STA\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, \_skeyTag + \_sSTA, p.\_state.AsBigInteger());

/\*FIELD\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, \_skeyTag + \_sX, p.\_x);

/\*FIELD\*/

Neo.SmartContract.Framework.Services.Neo.Storage.Put(ctx, \_skeyTag + \_sY, p.\_y);

}

LogExt("Bury(ks).p", p);

return p; // return Point p to signal if key is Missing or bad key

}

}

TODO

* 1. NPC Level 4 Collectible

At the NPC Level 4 Collectible level, NPCs use a unique scheme called NEO Storage Keys to manage the representation and lifecycle of large collections of diverse entities across an unlimited number of applications (and application versions), specific users, single entities as well as unlimited sized, BigInteger-indexed collections of entities with an arbitrary number of fields. Functional capabilities at the Collectible level include:

* PutElement(NEO Storage Key)
* GetElement(NEO Storage Key)
* BuryElement(NEO Storage Key)

TODO

public static class NeoEntityModel /\* Level 4 Collectible \*/

{

public enum EntityState

{

NULL,

INIT,

SET,

PUTTED,

GETTED,

MISSING,

TOMBSTONED,

NOTAUTHORIZED /\* Future \*/

}

public static BigInteger AsBigInteger(this EntityState state)

{

int istate = (int)state;

BigInteger bis = istate;

return bis;

}

public static EntityState BytesToEntityState(byte[] bsta)

{

int ista = (int)bsta.AsBigInteger();

NeoEntityModel.EntityState sta = (NeoEntityModel.EntityState)ista;

return sta;

}

}

TODO

TODO

* 1. Future Implementation Levels

NPC Level 5 Extendible is a future extension to the current NPC framework to support arbitrary extensions to the number of fields.

NPC Level 6 Authorized is also a future extension to the NPC framework to support authenticated and role-based authorization to entities you create (and others create).

1. Results

TODO

1. Roadmap

TODO Extendibility

TODO Storage space optimizations

TODO Authorization controls

1. Summary

TODO

1. Appendix A – What is a Bluepaper?

Quoting from the following are entitled *What are Bluepapers?*

To prepare a blue paper, one must first collect every technical aspect of a new technology (what it is, how it is useful, what its potential implications are, etc.) or the technical specifications of a new piece of equipment (components, features, uses, etc.)

The general outline for a blue paper consists of:

* *Introduction – brief description of the technology or equipment being discussed.*
* *Features or Characteristics – describes the parameters and scope of the new technology or the functional features of the equipment being discussed.*
* *Uses – how the new technology or equipment can be used.*
* *Implications – effects the new technology or equipment might have on society and its industry.*

Blue papers are usually written at higher levels of detail as they are used primarily by people in the particular technological field in question, rather than by the public at large. Although you are writing to industry insiders for the most part, it is still advisable to define seldom used words and phrases or to explain new concepts or concepts that have limited public exposure when writing a blue paper.

* 1. References
* <https://en.wikipedia.org/wiki/White_paper#Variants>
* <http://www.genuinewriting.com/blue_paper.html>

1. Appendix B - NEO Blockchain Architecture Reference Model (ARM)

TODO

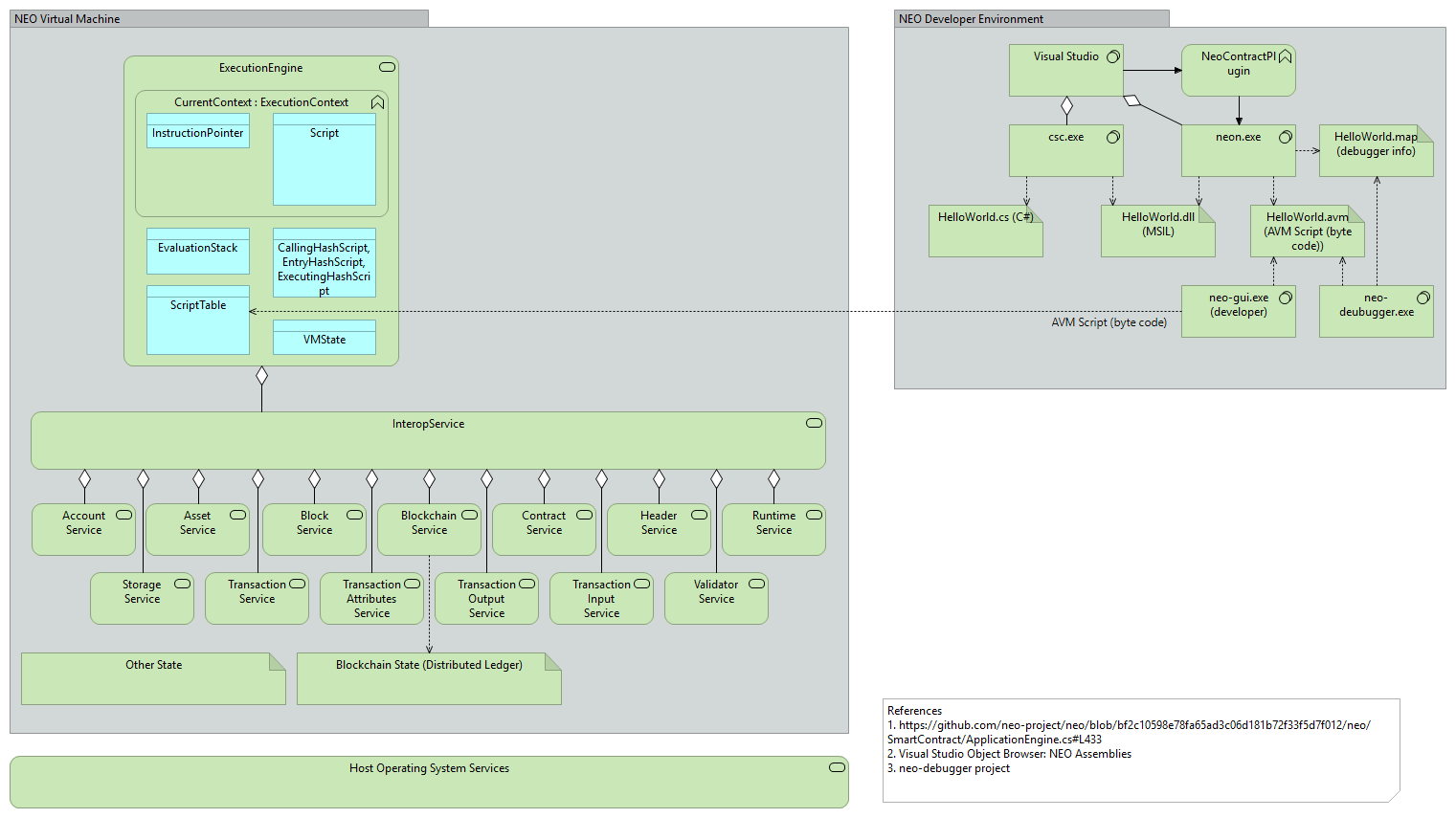


Figure . NEO Blockchain Architecture Reference Model

TODO

* 1. NEO Developer Environment

The architecture of the NEO environment from a developer perspective is illustrated in the following diagram. This diagram was inspired by the Relfos/neo-debugger-tools project [[Relfos/neo-debugger-tools](https://github.com/Relfos/neo-debugger-tools)].

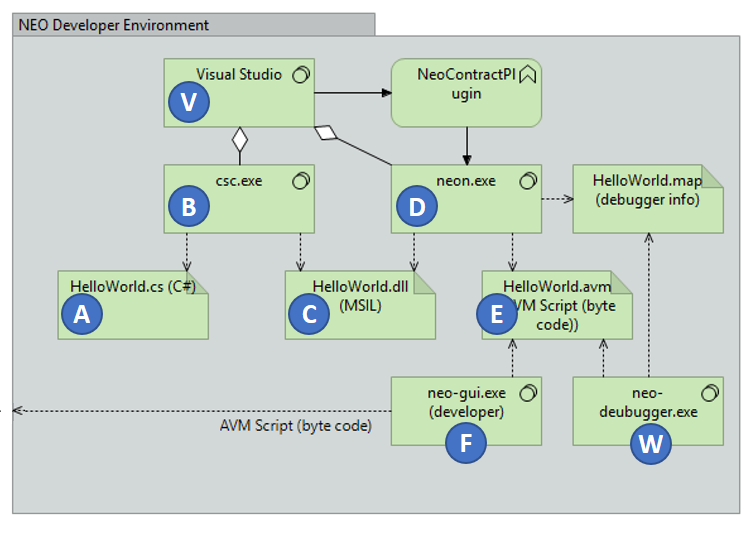


Figure . NEO Developer Environment Architecture

TODO

The following table describes the key components of the NEO developer environment.

Table . NEO Developer Environment Component Descriptions

| **Mark** | **Name** | **Description** |
| --- | --- | --- |
| [A](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/A32.png) | HelloWorld.cs | Smart contract source (example) |
| [B](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/B32.png) | csc.exe | C# Compiler |
| [C](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/C32.png) | HelloWorld.dll | Smart contract compiled assembly (MSIL) |
| [D](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/D32.png) | neon.exe | NEO Transcompiler (debugger version) |
| [E](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/E32.png) | HelloWorld.avm | NEO VM script file (NEO byte code script) |
| [F](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/F32.png) | neo-gui (developer) | Smart contract deployment and testing tool |
| [V](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/V32.png) | Visual Studio | Microsoft .NET/C# IDE |
| [W](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/W32.png) | neo-debugger | Interactive source and byte code level debugger |

TODO

* 1. NEO Virtual Machine Environment

TODO

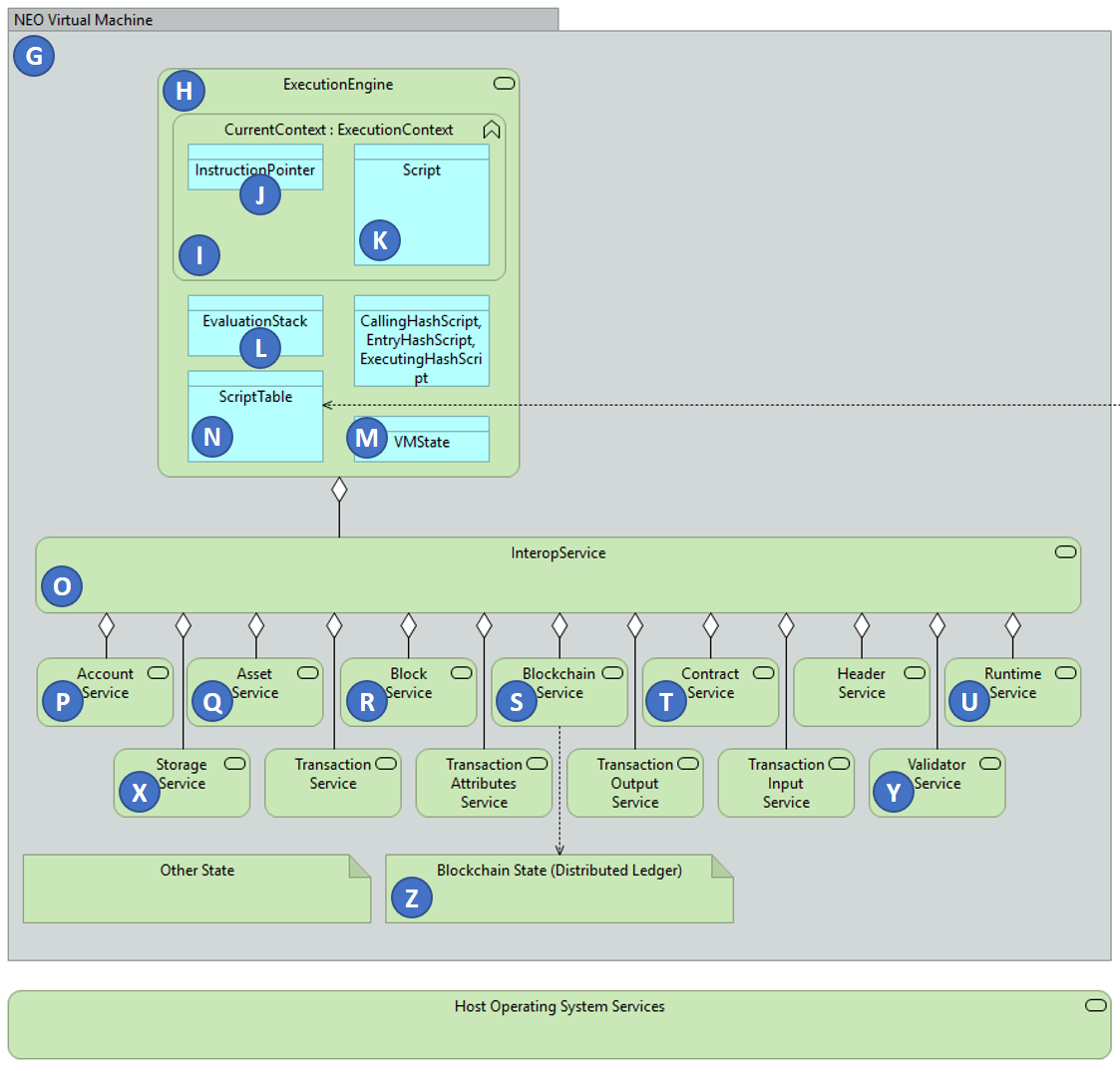


Figure . NEO Virtual Machine Architecture

TODO

Table . NEO Virtual Machine Component Descriptions

| **Mark** | **Name** | **Description** | **Namespaces/APIs Callable by a Smart Contract** |
| --- | --- | --- | --- |
| [G](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/G32.png) | NEO VM | Virtual machine major components and services | - |
| [H](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/H32.png) | ExecutionEngine | Execution engine components | Neo.SmartContract.Framework.Services.System.ExecutionEngineclass |
| [I](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/I32.png) | CurrentContext | Execution engine current context | - |
| [J](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/J32.png) | InstructionPointer | Execution engine current instruction pointer | - |
| [K](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/K32.png) | Script | Current smart contract script being executed | Neo.SmartContract.Framework.SmartContract class |
| [L](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/L32.png) | ExecutionStack | Execution engine data stack | - |
| [M](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/M32.png) | VMState | Virtual machine current state (HALT, FAULT, BREAK) | - |
| [N](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/N32.png) | ScriptTable | Virtual machine script table of callable scripts | - |
| [O](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/O32.png) | InteropService | Interop service layer for everything that isn't one of the basic execution engine capabilities (see above) | - |
| [P](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/P32.png) | Account Service | Account service | Neo.SmartContract.Framework.Services.Neo.Account class |
| [Q](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/Q32.png) | Asset Service | Asset service | Neo.SmartContract.Framework.Services.Neo.Asset class |
| [R](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/R32.png) | Block Service | Block service | Neo.SmartContract.Framework.Services.Neo.Block class |
| [S](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/S32.png) | Blockchain Service, Header Service | Blockchain and Header services | Neo.SmartContract.Framework.Services.Neo.Blockchain class, Neo.SmartContract.Framework.Services.Neo.Header class |
| [T](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/T32.png) | Contract Service | Contract service | Neo.SmartContract.Framework.Services.Neo.Contract class |
| [U](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/U32.png) | Runtime Service | Runtime service | Neo.SmartContract.Framework.Services.Neo.Runtime class |
| [X](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/X32.png) | Storage Service | Storage service | Neo.SmartContract.Framework.Services.Neo.Storage class |
| [Y](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/Y32.png) | Validator Service | Validator service | Neo.SmartContract.Framework.Services.Neo.Validator class |
| [Z](https://github.com/mwherman2000/neo-windocs/blob/master/images/balls/Z32.png) | Blockchain State | Distributed ledger | - |

1. About the Author
   1. Michael Herman

TODO