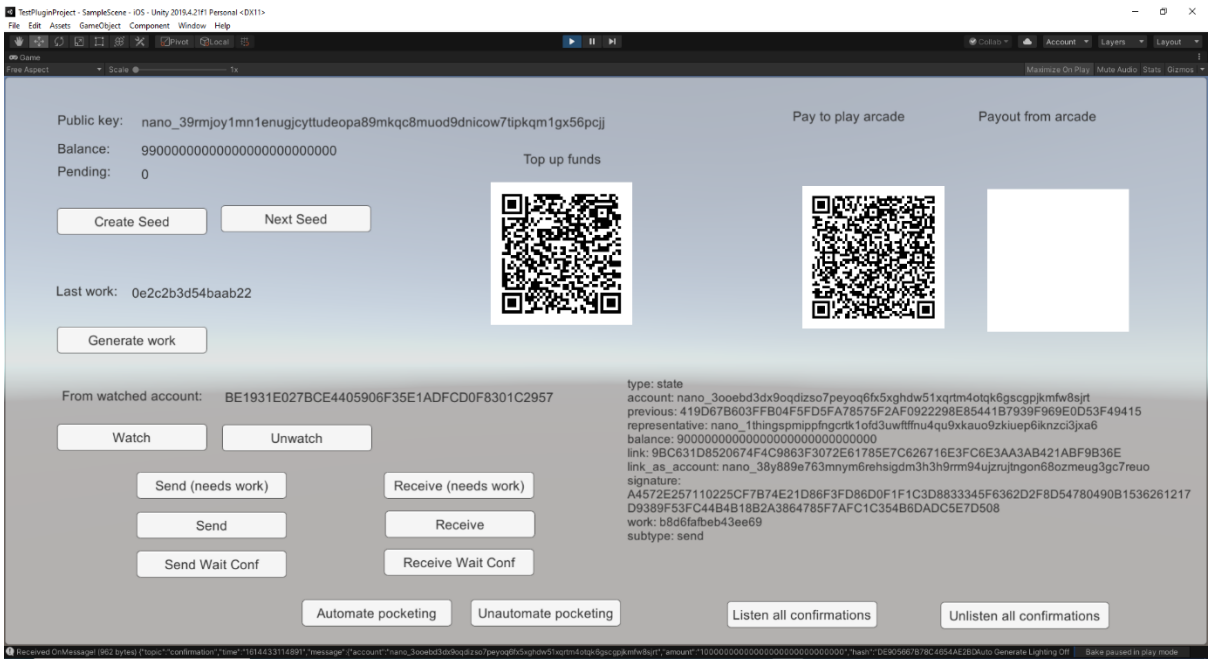
NanoPlugin

Plugin for integrating the Nano cryptocurrency into Unity.

Features include: functions for processing blocks, creating seeds + reading/saving them to disk in password-protected encrypted files. Generating qr codes, listening for payments to single accounts, private key payouts and many more.

Quickly testing

The easiest way to get started is to open up the scene in Scenes/SampleScene.unity. This provides a demo level to explore all the available functionality and connects to a publically available nodejs server.



A description of the various files

NanoUtils.cs contains various generic functions such as creating seeds, encrypting/decrypting them using AES with a password, converting to accounts, converting between Raw and Nano and various other things.

NanoWebsocket.cs maintains the websocket connection to the proxies. **NanoManager.cs** is where all other functionality is located

NanoAmount.cs is a helper class for storing and manipulating the raw units in Nano RPC.cs connects to a server forward proxy for the nano node

NanoDemo.cs contains content for the sample scene which illustrates all the functionality available.

TestUtils.cs contains test for various things.

To set up using the Nano plugin copy the Nano folder across to your project. The Scripts/Scenes folders are not required for deployment. A simple example of setting up the necessary functions with the public servers is:

```
using NanoPlugin;


public class NanoDemo : MonoBehaviour
{
  void Start()
  {
    // Initialize the NanoManager & NanoWebsocket
    nanoManager = gameObject.AddComponent<NanoManager>();
    nanoManager.rpcURL = "http://95.216.164.23:28103"; // Modify url to RPC server host:port
    nanoManager.defaultRep =
"nano_387tj8fjeo6r35ry5tjppympp8dct4d1ogpis7uaxsw8ywsrgp6shfge7two";


    nanoWebsocket = gameObject.AddComponent<NanoWebSocket>();
    nanoWebsocket.url = "ws://95.216.164.23:28104"; // Modify url to websocket server host:port
    nanoManager.Websocket = nanoWebsocket;
  }


  private NanoWebSocket websocket;
  private NanoManager nanoManager;
};
```

Various other functions available

Generate a private key

```
byte[] privateKey = NanoUtils.GeneratePrivateKey();
```

Convert a byte[] to hex string

```
string hexPrivateKey = NanoUtils.ByteArrayToHexString(privateKey);
```

Convert a hex string to byte[]

```
byte[] privateKey = NanoUtils.HexStringToByteArray(hexPrivateKey);
```

Get the nano_ address from the seed

```
string address = NanoUtils.PrivateKeyToAddress(privateKey);
```

Get the public key hex string from the seed

```
string publicKey = NanoUtils.PrivateKeyToPublicKeyHexString(privateKey);
```

Convert a nano_ address and public key hex string

```
string publicKey = NanoUtils.AddressToPublicKeyHexString(address);
```

Convert a public key hex string to nano_ address

```
string address = NanoUtils.PublicKeyToAddress(publicKey);
```

Individual node functions

These are low level building block functions which are not always necessary, it is recommened to use the utility functions mentioned below which encapsulates most of this.

Get account information

```
// First we get the frontier

yield return nanoManager.AccountInfo(address, (accountInfo) =>

{

  var previous = accountInfo.frontier;

  var rep = accountInfo.representative;

  if (previous != null)

    // account exists

});
```

Get pending blocks

```
List<PendingBlock> pendingBlocks = null;

yield return nanoManager.PendingBlocks(address, (responsePendingBlocks) =>

{

  pendingBlocks = responsePendingBlocks;

});

if (pendingBlocks != null && pendingBlocks.Count > 0)

....
```

Create & sign a block

```
var block = nanoManager.CreateBlock(address, NanoUtils.HexStringToByteArray(privateKey),
newBalance, link, previous, rep, work);
```

Process a block to the network

```
nanoManager.Process(block, BlockType.send, (hash) =>

{

  if (hash != null)

    // Block is processed

}
```

Create QR Code

```
var texture2D = NanoUtils.GenerateQRCodeTextureWithAmount(250, address, numRawPayToPlay,
50);
```

```
var sprite = Sprite.Create(texture2D, new Rect(0.0f, 0.0f, texture2D.width, texture2D.height), new Vector2(0.5f, 0.5f), 100.0f);
```

Utility node functions

Send nano, waiting for confirmation from the network

```
IEnumerator SendWaitConfHandler()
{
  yield return nanoManager.SendWaitConf(toAddress, amount, privateKey, (error, hash) =>
  {
    if (!error)
    {
      Debug.Log("Send wait confirmed!!");
    }
    else
    {
      Debug.Log("Error with SendWaitConf");
    }
  });
}
```

Send nano, without waiting for confirmation from the network

```
private IEnumerator SendHandler()
{
  yield return nanoManager.Send(toAddress, amount, privateKey, (error, hash) =>
  {
    if (!error)
    {
      Debug.Log("Send confirmed!!");
    }
    else
    {
      Debug.Log("Error with Send");
    }
  }); }
```

Receive nano, waiting for confirmation from the network

```
yield return nanoManager.ReceiveWaitConf(address, pendingBlock, privateKey, (error, hash) => { }
```

Receive nano, without waiting for confirmation from the network

```
yield return nanoManager.Receive(address, pendingBlock, privateKey, (error, hash) => { }
```

Automatically pocket nano

```
nanoManager.AutomatePocketing(address, privateKey, (block) =>

 {

   // block is the block received for the

 });
```

Listen to all confirmations (useful for visualisers)

```
// Register a confirmation listener, can have multiple of these

nanoManager.AddConfirmationListener((websocketConfirmationResponse) =>

{

 var block = websocketConfirmationResponse.message.block;

 // Do something, show a ball etc

});


// Set up the pipelining for the above confirmation listener

nanoManager.ListenAllConfirmations();
```

Note: Make sure that config.js on the server has listen_all = true

Recommendation setups for:

Arcade machines

Listen to payment - Have 1 seed per arcade machine, start at first index then increment each time a payment is needed. This only checks pending blocks, don't have anything else pocketing these funds automatically. Every time a new payment is needed, move to the next index. Only 1 payment can be listening at any 1 time!
Create a QR code for the account/amount required. Then listen for the payment. For payouts do a similar process with showing a QR Code (use the variant taking a private key), and listen for payout.

Single player

Create seed for player and store it encrypted with password (also check for local seed files if they want to open them)
Loop through seed files, save seed & send and wait for confirmation.

Multiplayer

Process seed (as above), create & sign block locally and hand off to server.
Server does validation (checks block is valid) then does appropriate action and send response to client.

To run your own test servers (recommended for production)

Requires running a nano node, as well as npm and nodejs being installed.

1. Run the nano_node binary after enabling rpc and websocket in config-node.toml file. nano_node --daemon
2. cd TestServer
3. Modify the config.js settings to be applicable for your system.
4. npm install
5. node server.js

A nano node is required to be running which the websocket & http server (for RPC requests) will communicate with. Websocket & RPC should be enabled in the node-config.toml nano file.

A http server (for RPC requests) is definitely needed for communicating with the nano node via JSON-RPC, a test nodejs script is supplied for this called server.js. A websocket server to receive notifications from the nano network is highly recommended to make the most use of the plugin functionality. Another test server called websocket_node.js listening for confirmations for blocks in real-time on the nano network. server_work_callback.js communicate with dpow (distributed POW) for work generation, a work_peer in node-config.toml should be set up for this, all these scripts are found in the ./TestServer subdirectory. Running server.js will also run websocket_node.js & server_work_callback,js. The websocket script makes 2 connections to the node, 1 is filtered output and 1 gets all websocket events (usual for visualisers). If you only need filtered output (recommended!) then disable allow_listen_all=false in config.js, this is the default.

Limitations

- The test servers should not be used in production due to lack of security/ddos protection. Likely canditates for a future version are the NanoRPCProxy.