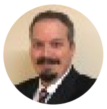
How to: Access CLI Wallet over Websocket in C#

With a sample Windows Form app tutorial


Aaron Pemberton
Jun 6 · 13 min read ★

There are several methods with which an app can be made to interact with a blockchain and perform functions, such as asset transfer transactions. This guide will focus on connecting to the CLI Wallet, uploading a user's wallet file, and creating a transaction with the user's account over websockets. Advanced topics, such as data encryption and SSL certificates, will not be covered in this guide as its main purpose is only to explain the basic concepts in communicating with the CLI Wallet over websocket.

For this tutorial, I will be utilizing DECENT's DCore blockchain (testnet), a VPS from Vultr.com, and Visual Studios to create a simple

Windows form app.

Let's begin with a brief introduction to DECENT and their blockchain platform DCore. Founded in 2015, DECENT is one of the first blockchain companies. DECENT has developed their own blockchain protocol, DCore, a platform that empowers users to create or migrate applications into a blockchain environment. Cooperating closely with top investment funds and incubators, DECENT is dedicated to building an ecosystem upon its proprietary blockchain technology to help developers and businesses adapt to a decentralized future, especially within the media and entertainment industries. DCore utilizes a Delegated Proof of Stake (DPoS) consensus method which enables achieving current speeds of more than 2000 transactions per second (TPS). DCore enables large file storage and distribution with the native integration of file sharing systems IPFS and CDN. DECENT provides multiple software development kits (SDK) in JVM, Swift, TypeScript, PHP, Java and JavaScript to further aid in developing on their platform.

DCore's range of features include custom token generation, content distribution, revenue sharing, encrypted messaging and more. Whether your project wants to incorporate a blockchain based in-game monetary system or a mobile, encrypted messaging service or almost anything you can envision, DECENT's DCore is more than capable.

For more information on DECENT, DCore or the SDK's, please visit their website at https://decent.ch/ and GitHib at https://github.com/DECENTfoundation

VPS Setup:

For detailed instructions on how to build DCore on a VPS, please see my previous Medium article here.

Dcore Testnet Setup:

For this tutorial, I will be using the DCore testnet. Full information on the testnet can be found here: https://docs.decent.ch/Testnets/index.html

First, create a new folder on your VPS to be used as a data directory for the Testnet. This tutorial will use the same naming convention as the DCore testnet instructions, though you can locate and name the folder as you see fit.

```
1] cd
2] mkdir /home/my-testnet-folder
```

Next, download the testnet genesis file to your testnet folder.

```
1] cd /home/my-testnet-folder
2] wget https://docs.decent.ch/assets/genesis-public-
testnet.json
```

Navigate to the folder where you installed the DCore daemon and CLI Wallet. Start the DCore daemon with parameters to specify the data directory you created, the genesis.json file and ip:port of one of the testnet nodes.

```
1] cd /usr/local/bin
2] ./decentd --data-dir=/home/my-testnet-
folder/public_testnet --genesis-json=/home/my-testnet-
folder/genesis-public-testnet.json --seed-
node=testnet.dcore.io:40000
```

This will start the DCore daemon using the testnet. While it is syncing the blockchain with the network, set up wscat next.

Wscat Setup:

To handle transferring the user's wallet file to the VPS, we will be using wscat to handle a separate websocket. Open a new terminal and follow the steps below to install node.js, npm and wscat on your VPS.

```
1] sudo apt-get install nodejs-dev node-gyp libssl1.0-dev
2] sudo apt-get install npm
3] npm install -g wscat
```

Navigate to the /usr/local/bin directory and create a new sub-directory which will contain the uploaded wallet files.

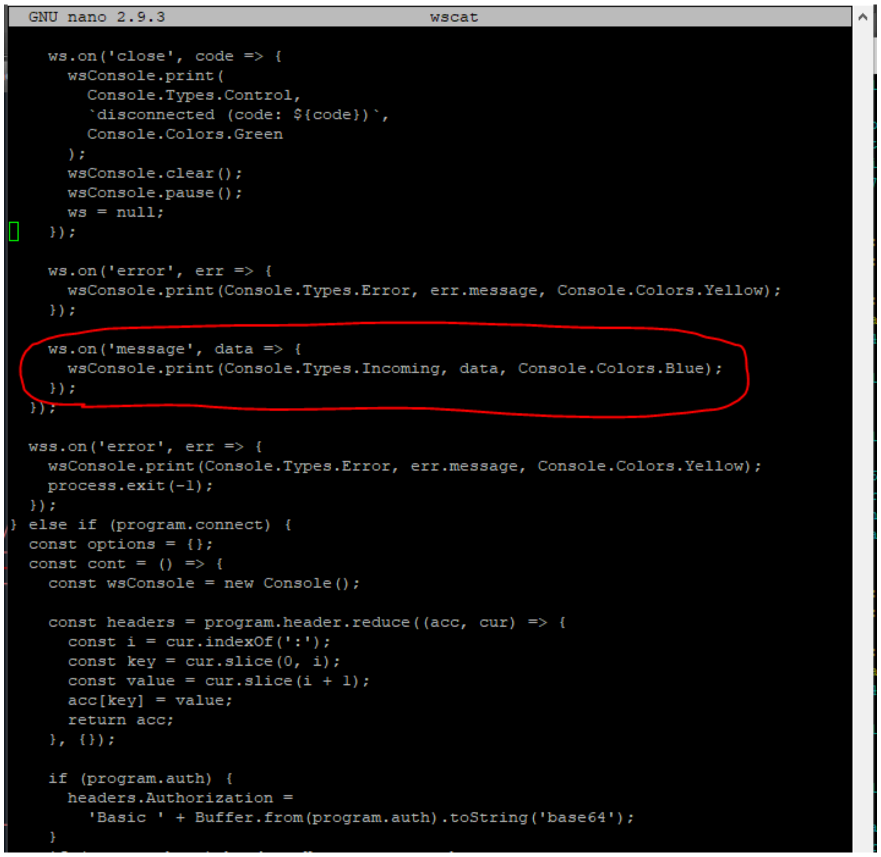
```
1] cd /usr/local/bin
2] mkdir downloads
```

Modify the wscat file to handle the requests we will be sending it by opening it up in the editor.

```
1] sudo nano wscat
```

Searching for the lines:

```
ws.on(‘message’, err=> {
  wsConsole.print(Console.Types.Incoming, data,
Console.Colors.Blue);
});
```



```
GNU nano 2.9.3                                  wscat

    ws.on('close', code => {
      wsConsole.print(
        Console.Types.Control,
        `disconnected (code: ${code})`,
        Console.Colors.Green
      );
      wsConsole.clear();
      wsConsole.pause();
      ws = null;
    });

    ws.on('error', err => {
      wsConsole.print(Console.Types.Error, err.message, Console.Colors.Yellow);
    });

    ws.on('message', data => {
      wsConsole.print(Console.Types.Incoming, data, Console.Colors.Blue);
    });
  });

  wss.on('error', err => {
    wsConsole.print(Console.Types.Error, err.message, Console.Colors.Yellow);
    process.exit(-1);
  });
} else if (program.connect) {
  const options = {};
  const cont = () => {
    const wsConsole = new Console();

    const headers = program.header.reduce((acc, cur) => {
      const i = cur.indexOf(':');
      const key = cur.slice(0, i);
      const value = cur.slice(i + 1);
      acc[key] = value;
      return acc;
    }, {});

    if (program.auth) {
      headers.Authorization =
        'Basic ' + Buffer.from(program.auth).toString('base64');
    }
```

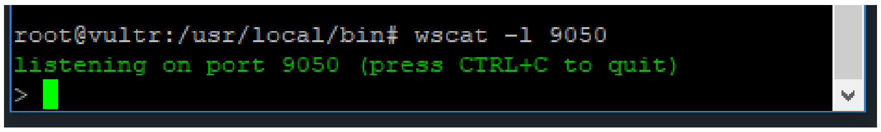
Modify the On Message function as follows:

```
ws.on('message', data => {
  var newWallet = data.includes("NEW-W-info");
  var delWallet = data.includes("DELETE-W-info");
   if (newWallet){
     var msg = data;
     var msgSplit = msg.split("?&?&?&");
     var fs = require("fs");
     fs.writeFile('downloads/' + msgSplit[2] + '-
wallet.json', msgSplit[1], (err) => {
     if (err) console.log(err);
     console.log("Successfully Written to File.");
     ws.send("Successfully Written to File.");
     });
   }
   if (delWallet){
     var msg = data;
     var msgSplit = msg.split("?&?&?&");
     var fs = require("fs");
     fs.unlinkSync('downloads/' + msgSplit[1] + '-
wallet.json',(err)=> {
     if (err) console.log(err);
     });
     console.log("Successfully Deleted File.");
     ws.send("Successfully Deleted File.");
   }
  });
});
```

When the wscat websocket receives a message, this event is triggered. The modifications made check if the received message is a new wallet command or a delete wallet command (message structure will be described in more detail later in this tutorial). If it is a new wallet command, the file is created in the /usr/local/bin/downloads folder and the wallet's json string saved into it. If it is a delete wallet command, it deletes the file and returns a successful response. Any other message received is simply ignored.

Close and save the wscat file then start wscat listening on port 9050 with the following command:

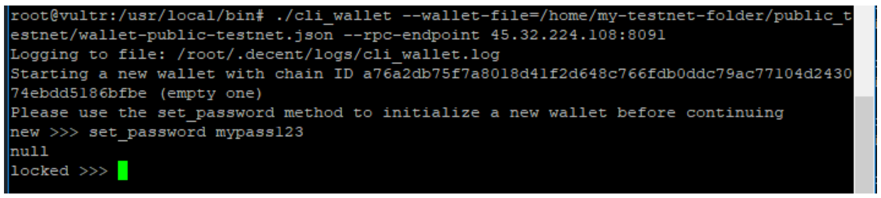
```
wscat -l 9050
```



```
root@vultr:/usr/local/bin# wscat -l 9050
listening on port 9050 (press CTRL+C to quit)
>
```

Finally, start the CLI wallet, listening for websocket connections, by opening a new terminal and the following commands (replacing your_IP with your VPS' IP address). Then use the set_password method in the CLI wallet to create a new password for the initial account.

```
1] cd /usr/local/bin
2] ./cli_wallet --wallet-file=/home/my-testnet-
folder/public_testnet/wallet-public-testnet.json --rpc-
endpoint your_IP:8091
```



```
root@vultr:/usr/local/bin# ./cli_wallet --wallet-file=/home/my-testnet-folder/public_t
estnet/wallet-public-testnet.json --rpc-endpoint 45.32.224.108:8091
Logging to file: /root/.decent/logs/cli_wallet.log
Starting a new wallet with chain ID a76a2db75f7a8018d41f2d648c766fdb0ddc79ac77104d2430
74ebdd5186bfbe (empty one)
Please use the set_password method to initialize a new wallet before continuing
new >>> set_password mypass123
null
locked >>>
```

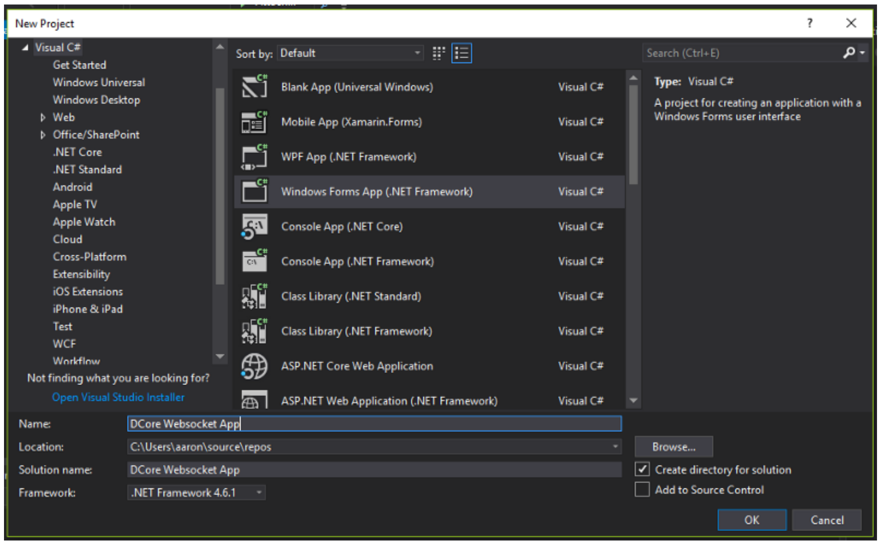
The VPS setup is now complete with the DCore daemon running on the testnet, the CLI wallet listening for websocket connections on port 8091 and wscat listening for connections on port 9050. Next, we will create a simple Windows Form App to utilize the CLI wallet on the VPS over websockets.

Window Form App:

The complete Virtual Studio solution for this app as well as the built executable can be downloaded from GitHub at https://github.com/AaronPemberton/DCore-Websocket-Tutorial-app

Form Layout:

Open Visual Studio and begin a new project. Select Visual C# and choose Windows Form App (.NET Framework). You can give the project any name you like. Use the .NET Framework 4.6.1.



Blank App (Universal Windows)	Visual C#
Mobile App (Xamarin.Forms)	Visual C#
WPF App (.NET Framework)	Visual C#
Windows Forms App (.NET Framework)	Visual C#
Console App (.NET Core)	Visual C#
Console App (.NET Framework)	Visual C#
Class Library (.NET Standard)	Visual C#
Class Library (.NET Framework)	Visual C#
ASP.NET Core Web Application	Visual C#
ASP.NET Web Application (.NET Framework)	Visual C#

Type: Visual C#

A project for creating an application with a Windows Forms user interface

Name: DCore Websocket App

Location: C:\Users\aaron\source\repos

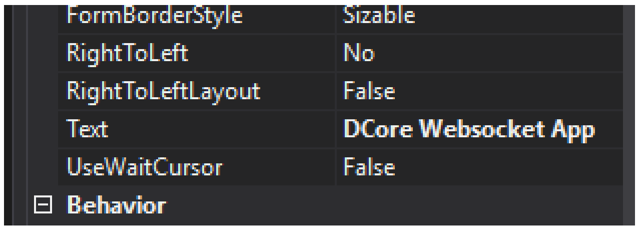
Solution name: DCore Websocket App

Framework: .NET Framework 4.6.1

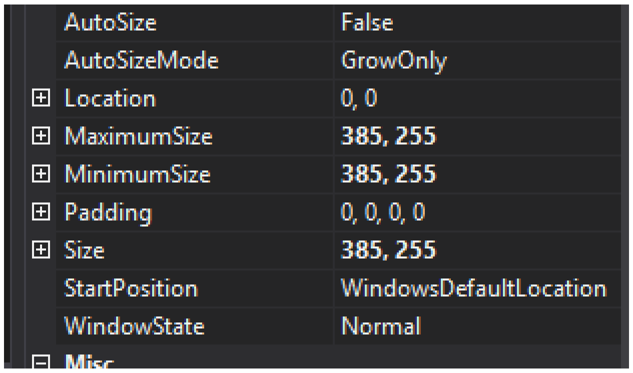
✓ Create directory for solution

☐ Add to Source Control

In the Form1.cs [Design] tab, select the blank form. In the Properties pallet, change the Text to the name of your app and set the MaximumSize and MinimumSize to 385, 255.



FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	**DCore Websocket App**
UseWaitCursor	False
⊟ **Behavior**	



AutoSize	False
AutoSizeMode	GrowOnly
⊞ Location	0, 0
⊞ MaximumSize	**385, 255**
⊞ MinimumSize	**385, 255**
⊞ Padding	0, 0, 0, 0
⊞ Size	**385, 255**
StartPosition	WindowsDefaultLocation
WindowState	Normal
⊟ **Misc**	

Insert the following into your blank form:

Add a Label and change the text to Wallet File:. Add a TextBox, change the name to textBox_walletFile and set ReadOnly to True. Add a Button, name it button_Select and change the text to Select.

Add a Label and change the text to Wallet Password:. Add a TextBox, change the name to textBox_Password and set the PasswordChar to an *. Add a Button, change the name to button_Show and change the text to Show.

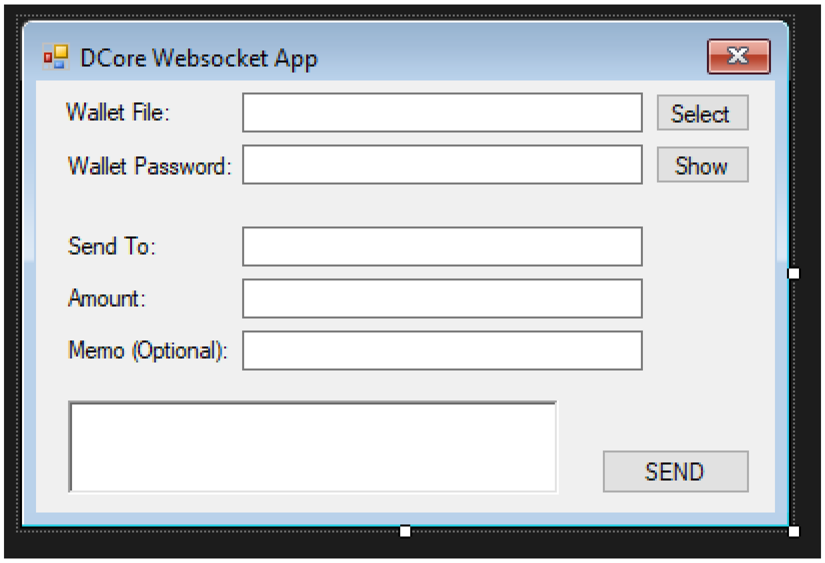
Add a Label and change the text to Send To:. Add a TextBox and change the name to textBox_sendTo.

Add a Label and change the text to Amount:. Add a TextBox and change the name to textBox_Amount.

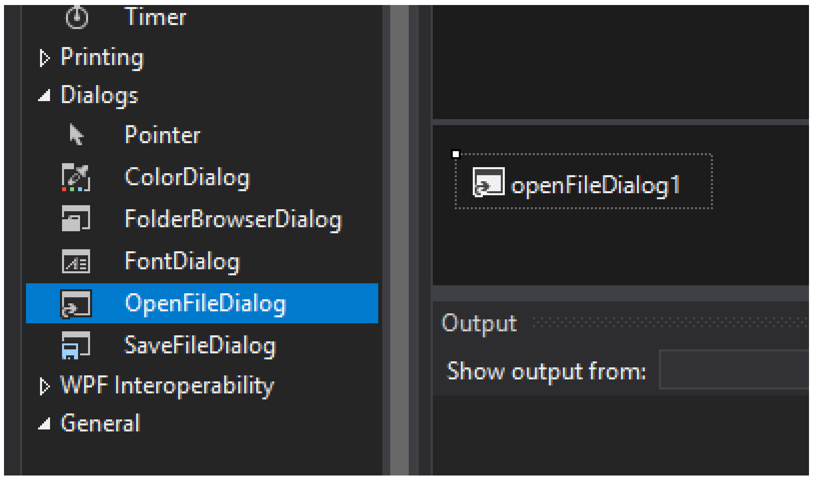
Add a Label and change the text to Memo (Optional):. Add a TextBox and change the name to textBox_Memo.

Add a RichTextBox, change the size to 245, 45 and change the name to richTextBox_Status.

Add a Button, change the name to button_Send and change the text to SEND.

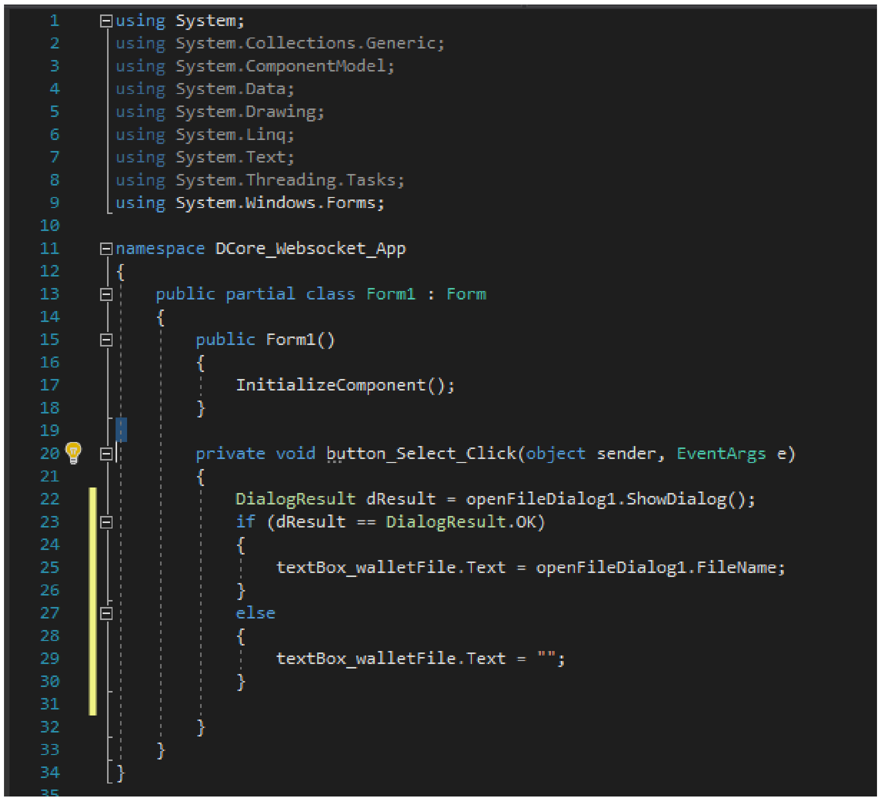


Select OpenFileDialog from the Toolbox, drag and drop it onto the form to create openFileDialog1.



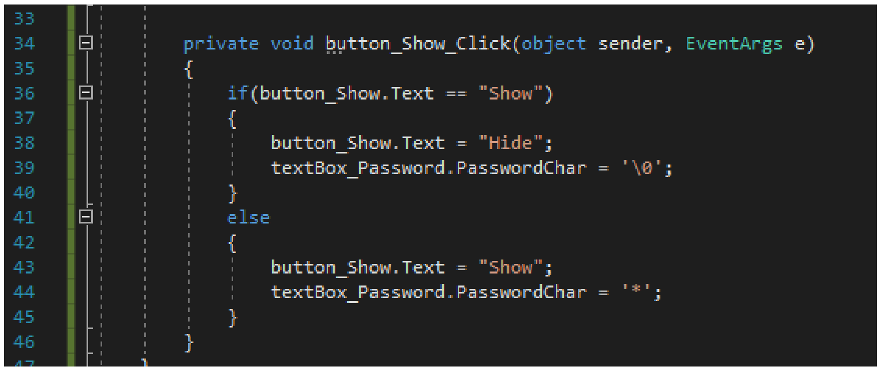
Adding the Code:

Double-click the Select button. This will open the Form1.cs page. When the user clicks the Select button, we want a dialog box to open in which the user can navigate to and select their wallet.json file. Once selected, we will save the path to file in the textBox_walletFile Textbox or blank out the Textbox if the user cancels the dialog box. To do this, add the following code inside the button_Select_Click method.



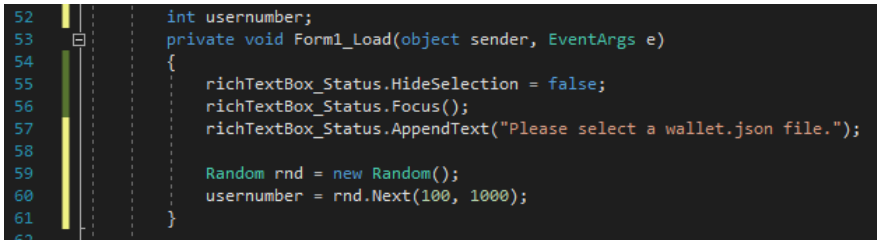
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace DCore_Websocket_App
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button_Select_Click(object sender, EventArgs e)
        {
            DialogResult dResult = openFileDialog1.ShowDialog();
            if (dResult == DialogResult.OK)
            {
                textBox_walletFile.Text = openFileDialog1.FileName;
            }
            else
            {
                textBox_walletFile.Text = "";
            }

        }
    }
}
```

Next, return to the Form1.cs [Design] tab and double-click the Show button. Earlier in the tutorial, we set the default for the textBox_Password Textbox to hide any input by replacing each character with an *. We will use this button to show and hide the password text, if the user needs to see what they have typed in. To do this, add the following code in the button_Show_Click method.



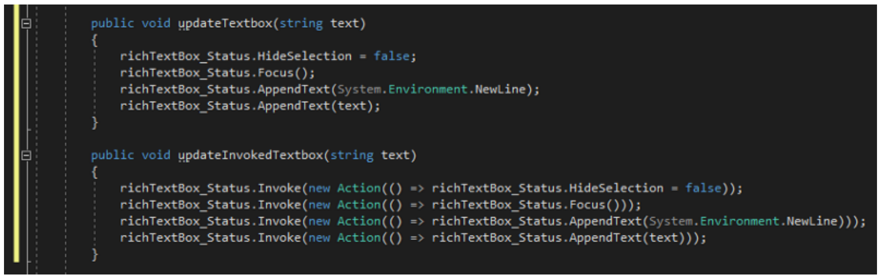
```
private void button_Show_Click(object sender, EventArgs e)
{
    if(button_Show.Text == "Show")
    {
        button_Show.Text = "Hide";
        textBox_Password.PasswordChar = '\0';
    }
    else
    {
        button_Show.Text = "Show";
        textBox_Password.PasswordChar = '*';
    }
}
```

Return back to the Form1.cs [Design] tab and double-click the form to generate the Form1_Load method. Add the following code to set the start-up text for the Status RichTextbox and to generate a unique, random user number.



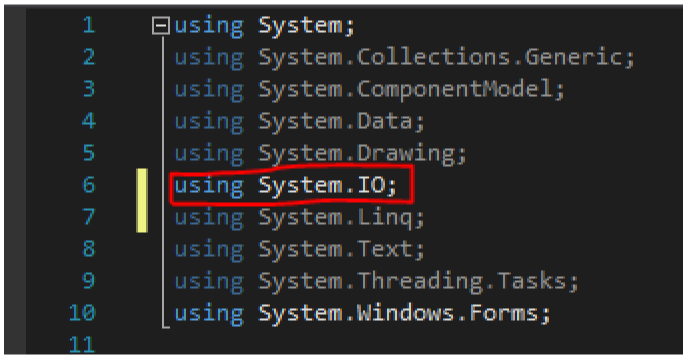
```
int usernumber;
private void Form1_Load(object sender, EventArgs e)
{
    richTextBox_Status.HideSelection = false;
    richTextBox_Status.Focus();
    richTextBox_Status.AppendText("Please select a wallet.json file.");

    Random rnd = new Random();
    usernumber = rnd.Next(100, 1000);
}
```

Next create two methods. One to update the richTextbox_status text when called from the current thread. And another to update it when called from another thread.

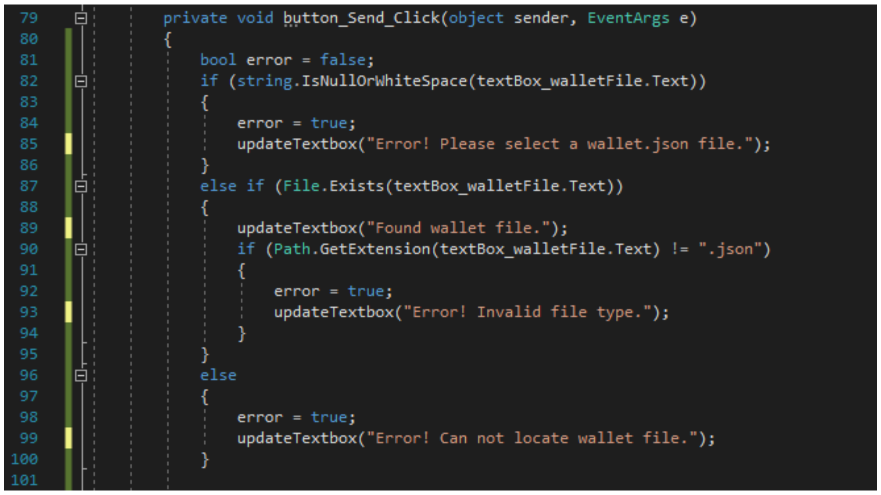


```
public void updateTextbox(string text)
{
    richTextBox_Status.HideSelection = false;
    richTextBox_Status.Focus();
    richTextBox_Status.AppendText(System.Environment.NewLine);
    richTextBox_Status.AppendText(text);
}

public void updateInvokedTextbox(string text)
{
    richTextBox_Status.Invoke(new Action(() => richTextBox_Status.HideSelection = false));
    richTextBox_Status.Invoke(new Action(() => richTextBox_Status.Focus()));
    richTextBox_Status.Invoke(new Action(() => richTextBox_Status.AppendText(System.Environment.NewLine)));
    richTextBox_Status.Invoke(new Action(() => richTextBox_Status.AppendText(text)));
}
```

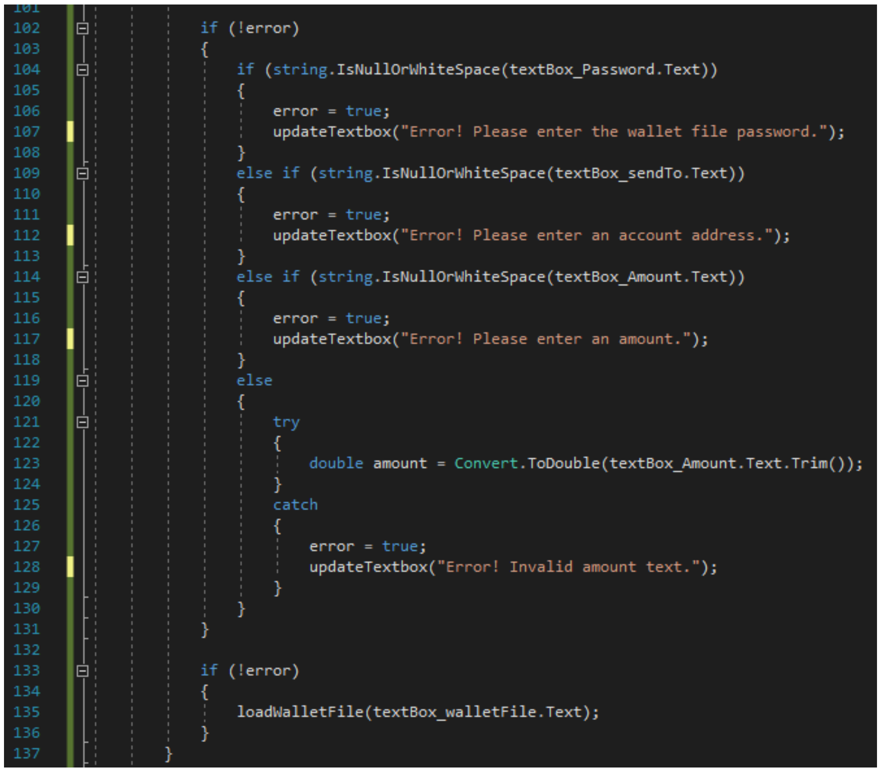
On the Form1.cs [Design] tab once more, double-click the Send button to generate the button_Send_Click method. Once the Send button is clicked, the first thing we want to do is verify that all of the mandatory Textboxes are filled out. Do this by adding a using System.IO statement near the top of the Form1.cs page and the following code to the button_Send_Click method.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```



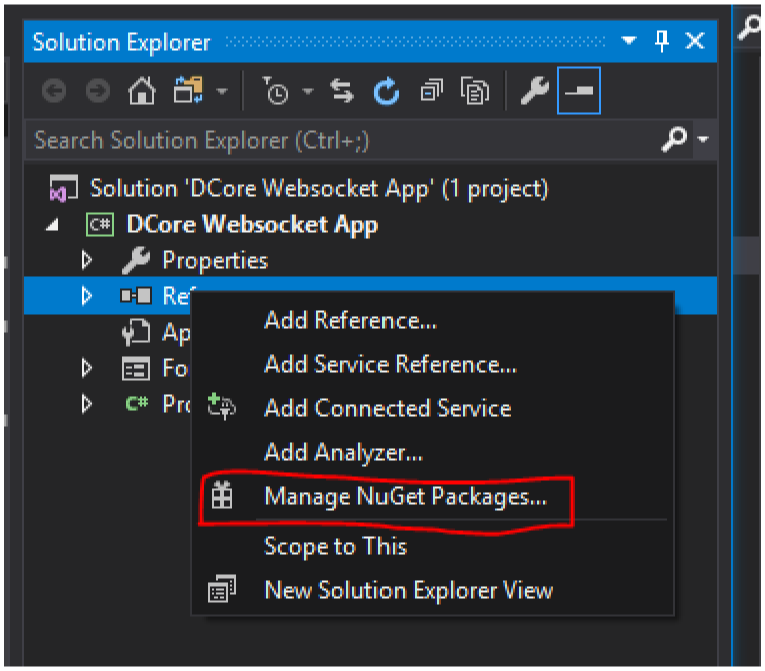
```
private void button_Send_Click(object sender, EventArgs e)
{
    bool error = false;
    if (string.IsNullOrWhiteSpace(textBox_walletFile.Text))
    {
        error = true;
        updateTextbox("Error! Please select a wallet.json file.");
    }
    else if (File.Exists(textBox_walletFile.Text))
    {
        updateTextbox("Found wallet file.");
        if (Path.GetExtension(textBox_walletFile.Text) != ".json")
        {
            error = true;
            updateTextbox("Error! Invalid file type.");
        }
    }
    else
    {
        error = true;
        updateTextbox("Error! Can not locate wallet file.");
    }
```



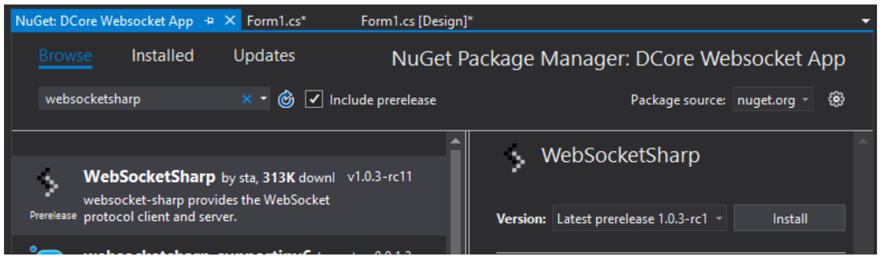
```
                if (!error)
                {
                    if (string.IsNullOrWhiteSpace(textBox_Password.Text))
                    {
                        error = true;
                        updateTextbox("Error! Please enter the wallet file password.");
                    }
                    else if (string.IsNullOrWhiteSpace(textBox_sendTo.Text))
                    {
                        error = true;
                        updateTextbox("Error! Please enter an account address.");
                    }
                    else if (string.IsNullOrWhiteSpace(textBox_Amount.Text))
                    {
                        error = true;
                        updateTextbox("Error! Please enter an amount.");
                    }
                    else
                    {
                        try
                        {
                            double amount = Convert.ToDouble(textBox_Amount.Text.Trim());
                        }
                        catch
                        {
                            error = true;
                            updateTextbox("Error! Invalid amount text.");
                        }
                    }
                }

                if (!error)
                {
                    loadWalletFile(textBox_walletFile.Text);
                }
            }
```

Finish off the button_Send_Click method with one last check if there were errors, if not send the textBox_walletFile text to the loadWalletFile method. Note: You will get a warning stating loadWalletFile does not exist. Ignore it as we will be generating it in the next few steps.

For this tutorial, we will be using WebSocketSharp NuGet package by sta. To install the package, in the Solution Explorer, right-click on References and select Manage NuGet Packages…



Select the Browse tab. In the search field enter websocketsharp and check the Include prerelease box. Select WebSocketSharp by sta and click Install.

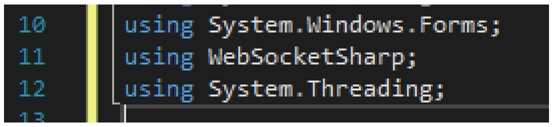

NuGet: DCore Websocket App
Form1.cs*
Form1.cs [Design]*
Browse
Installed
Updates
NuGet Package Manager: DCore Websocket App
Install

websocketsharp ☑ Include prerelease Package source: nuget.org

WebSocketSharp by sta, 313K downl v1.0.3-rc11
websocket-sharp provides the WebSocket protocol client and server.
Prerelease

WebSocketSharp
Version: Latest prerelease 1.0.3-rc1

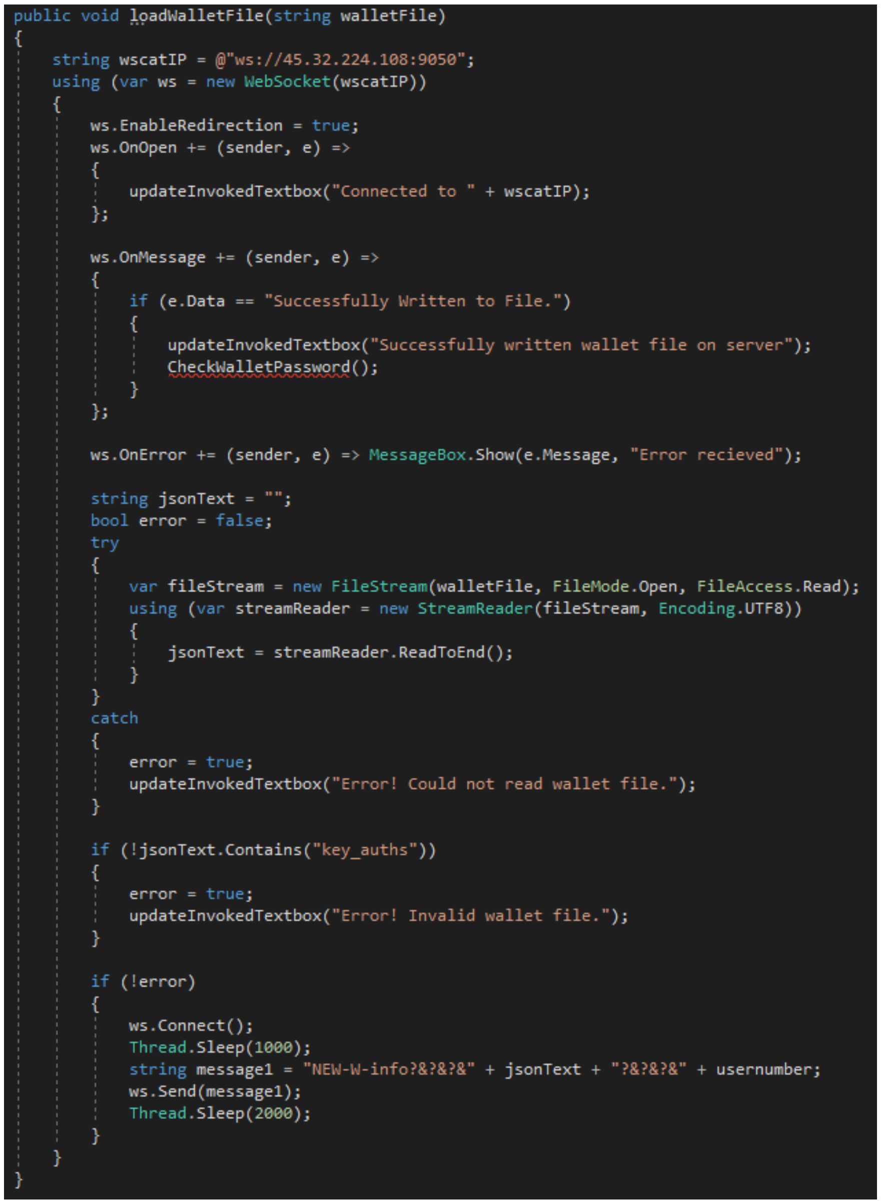
Then add a using statement near the top of the Form1.cs file for WebSocketSharp and for System.Threading.



```
using System.Windows.Forms;
using WebSocketSharp;
using System.Threading;
```

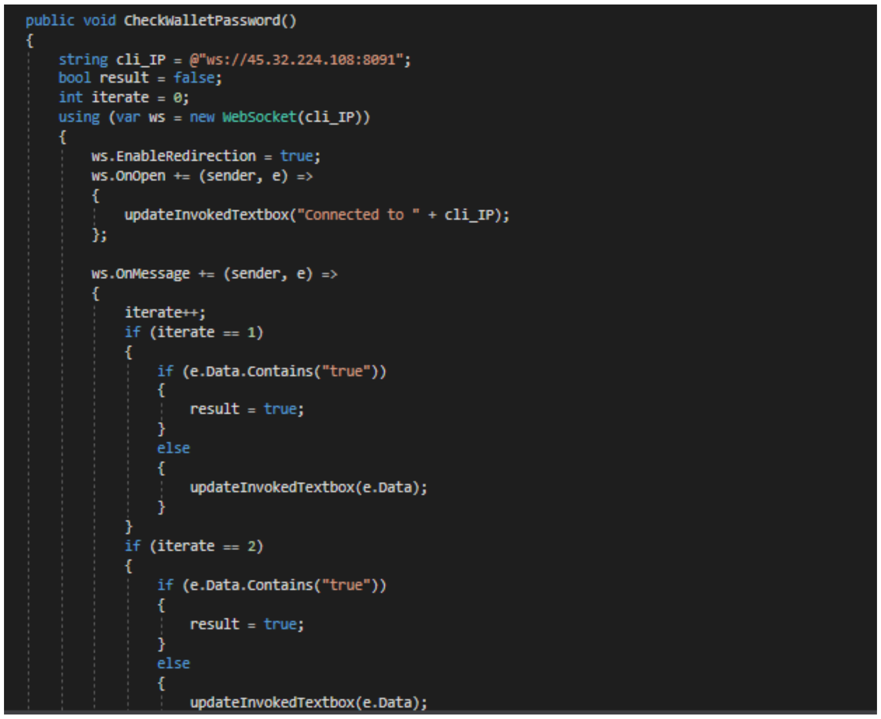
Create a new method named loadWalletFile that accepts a string. Create a string to contain the IP:Port address of your wscat websocket on the VPS. Initialize a new instance of a Websocket and set the OnOpen event listener to update the richTextBox_Status text to notify of the connection. Set the OnMessage event listener to check if the message is the succeed message from wscat and if so, update the Status text and execute the CheckWalletPassword method. This method will be generated in upcoming steps. Set the OnError event listener to display a message box with the error message.

Create an empty string to hold the wallet file text. Using FileStream, try to read the contents of the wallet file. If successful, check if it is a valid DCore wallet file by searching for the string “key_auths”. If there are no errors, connect to the wscat websocket and send it the wallet file text. Note: the wallet file text is prepended with NEW-W-info which is our signal to wscat that we want to create a new wallet file on the server. The ?&?&?& is a separator that is visually easy to see for this tutorial and serves as the string by which wscat splits the message into an array. The wallet file text is appended with the user number generated when the form loaded. This number will be used for the wallet file name on the server.

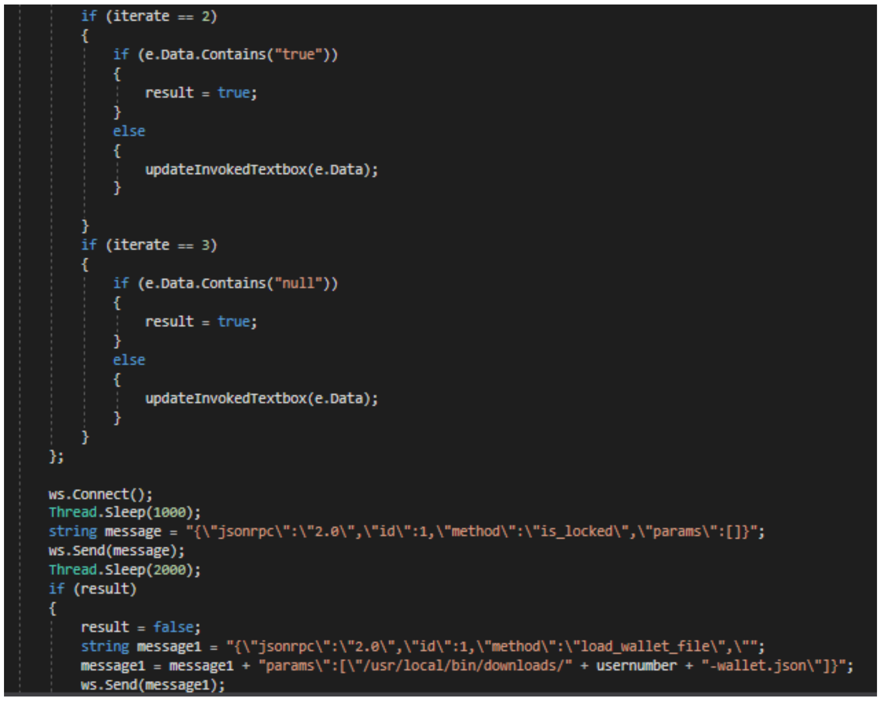


```
public void loadWalletFile(string walletFile)
{
    string wscatIP = @"ws://45.32.224.108:9050";
    using (var ws = new WebSocket(wscatIP))
    {
        ws.EnableRedirection = true;
        ws.OnOpen += (sender, e) =>
        {
            updateInvokedTextbox("Connected to " + wscatIP);
        };

        ws.OnMessage += (sender, e) =>
        {
            if (e.Data == "Successfully Written to File.")
            {
                updateInvokedTextbox("Successfully written wallet file on server");
                CheckWalletPassword();
            }
        };

        ws.OnError += (sender, e) => MessageBox.Show(e.Message, "Error recieved");

        string jsonText = "";
        bool error = false;
        try
        {
            var fileStream = new FileStream(walletFile, FileMode.Open, FileAccess.Read);
            using (var streamReader = new StreamReader(fileStream, Encoding.UTF8))
            {
                jsonText = streamReader.ReadToEnd();
            }
        }
        catch
        {
            error = true;
            updateInvokedTextbox("Error! Could not read wallet file.");
        }

        if (!jsonText.Contains("key_auths"))
        {
            error = true;
            updateInvokedTextbox("Error! Invalid wallet file.");
        }

        if (!error)
        {
            ws.Connect();
            Thread.Sleep(1000);
            string message1 = "NEW-W-info?&?&?&" + jsonText + "?&?&?&" + usernumber;
            ws.Send(message1);
            Thread.Sleep(2000);
        }
    }
}
```

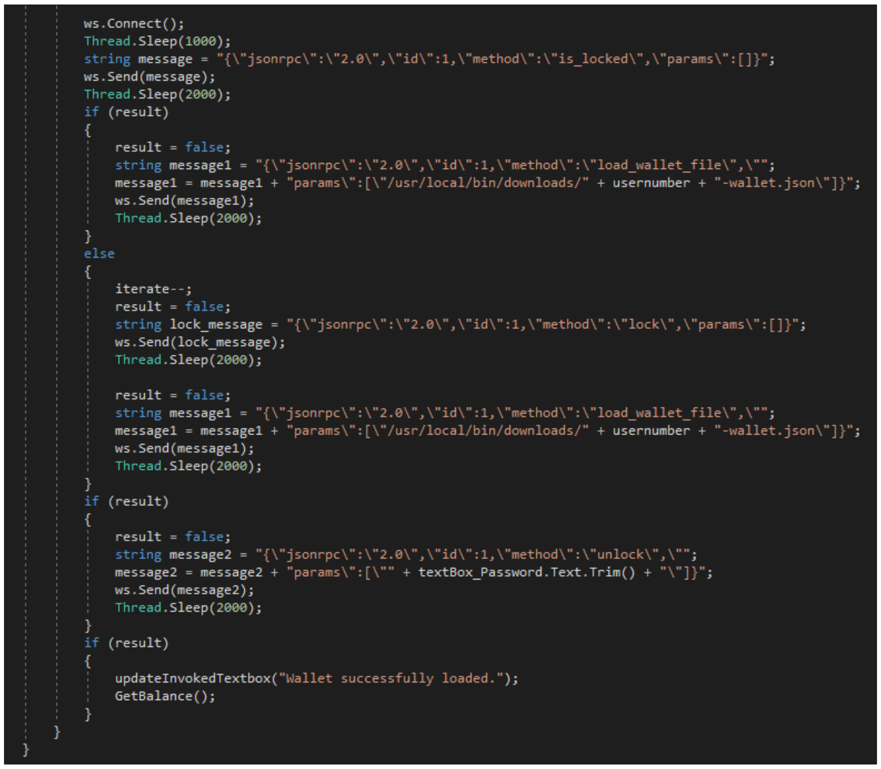
Next we will create the CheckWalletPassword method. This method will connect to the CLI wallet, load the wallet file saved on the server and attempt to unlock the wallet using the provided wallet password. Create a string to contain the IP:Port address of your CLI Wallet websocket on the VPS. Create a bool variable to be used to check if the websocket responses are correct. Create an integer variable to iterate through the separate API calls. The first call checks if the CLI Wallet is in use by verifying if it is locked. If the wallet is unlocked, it tries to lock the wallet prior to attempting to load the wallet file saved on the server. If successfully loaded, it then attempts to unlock the wallet with the password entered in the form field. If the wallet unlocks, we then call the GetBalance method.



```
public void CheckWalletPassword()
{
    string cli_IP = @"ws://45.32.224.108:8091";
    bool result = false;
    int iterate = 0;
    using (var ws = new WebSocket(cli_IP))
    {
        ws.EnableRedirection = true;
        ws.OnOpen += (sender, e) =>
        {
            updateInvokedTextbox("Connected to " + cli_IP);
        };

        ws.OnMessage += (sender, e) =>
        {
            iterate++;
            if (iterate == 1)
            {
                if (e.Data.Contains("true"))
                {
                    result = true;
                }
                else
                {
                    updateInvokedTextbox(e.Data);
                }
            }
            if (iterate == 2)
            {
                if (e.Data.Contains("true"))
                {
                    result = true;
                }
                else
                {
                    updateInvokedTextbox(e.Data);
```

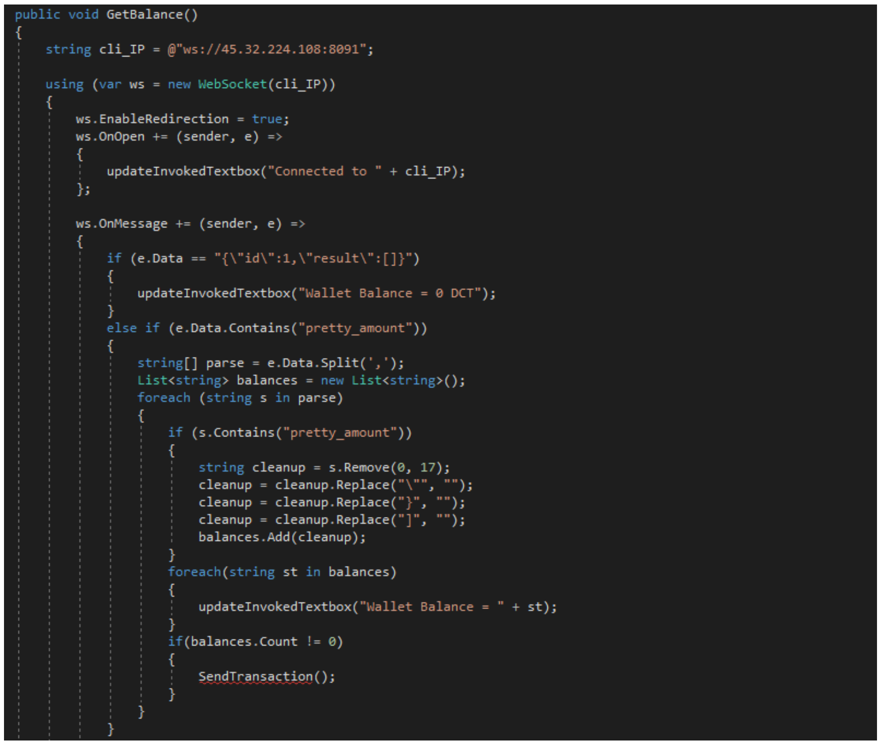


```
            if (iterate == 2)
            {
                if (e.Data.Contains("true"))
                {
                    result = true;
                }
                else
                {
                    updateInvokedTextbox(e.Data);
                }

            }
            if (iterate == 3)
            {
                if (e.Data.Contains("null"))
                {
                    result = true;
                }
                else
                {
                    updateInvokedTextbox(e.Data);
                }
            }
        };

        ws.Connect();
        Thread.Sleep(1000);
        string message = "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"is_locked\",\"params\":[]}";
        ws.Send(message);
        Thread.Sleep(2000);
        if (result)
        {
            result = false;
            string message1 = "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"load_wallet_file\",\"";
            message1 = message1 + "params\":[\"/usr/local/bin/downloads/" + usernumber + "-wallet.json\"]}";
            ws.Send(message1);
```

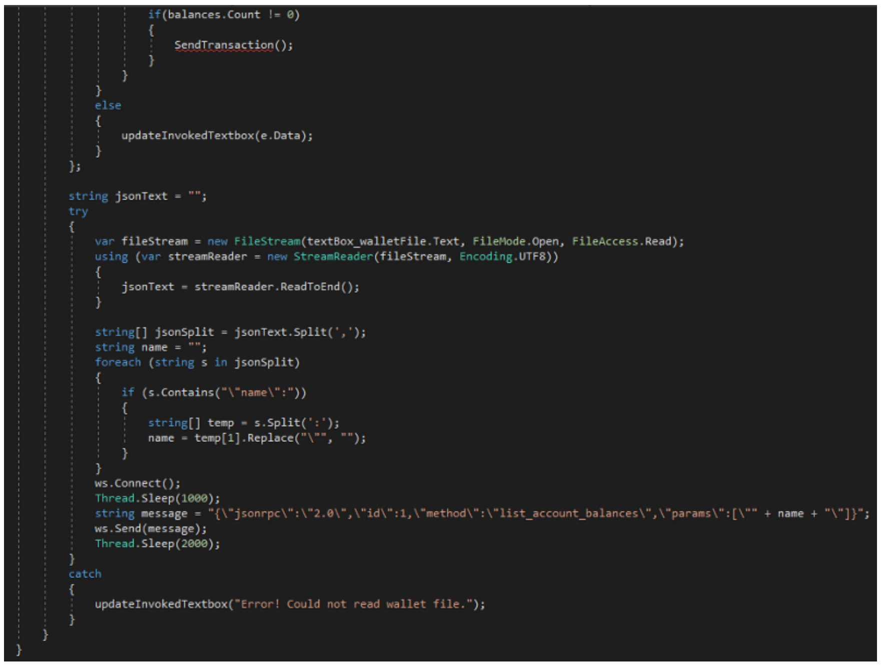


```
            ws.Connect();
            Thread.Sleep(1000);
            string message = "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"is_locked\",\"params\":[]}";
            ws.Send(message);
            Thread.Sleep(2000);
            if (result)
            {
                result = false;
                string message1 = "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"load_wallet_file\",\"";
                message1 = message1 + "params\":[\"/usr/local/bin/downloads/" + usernumber + "-wallet.json\"]}";
                ws.Send(message1);
                Thread.Sleep(2000);
            }
            else
            {
                iterate--;
                result = false;
                string lock_message = "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"lock\",\"params\":[]}";
                ws.Send(lock_message);
                Thread.Sleep(2000);

                result = false;
                string message1 = "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"load_wallet_file\",\"";
                message1 = message1 + "params\":[\"/usr/local/bin/downloads/" + usernumber + "-wallet.json\"]}";
                ws.Send(message1);
                Thread.Sleep(2000);
            }
            if (result)
            {
                result = false;
                string message2 = "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"unlock\",\"";
                message2 = message2 + "params\":[\"" + textBox_Password.Text.Trim() + "\"]}";
                ws.Send(message2);
                Thread.Sleep(2000);
            }
            if (result)
            {
                updateInvokedTextbox("Wallet successfully loaded.");
                GetBalance();
            }
        }
    }
```

The GetBalance method obtains the account name from the wallet file then sends the list_account_balances request to the CLI Wallet for that account name. The message received is then parsed for the appropriate information to display in the Status RichTextbox. If the account does hold a balance, then the SendTransaction method is called to attempt to process the transaction.

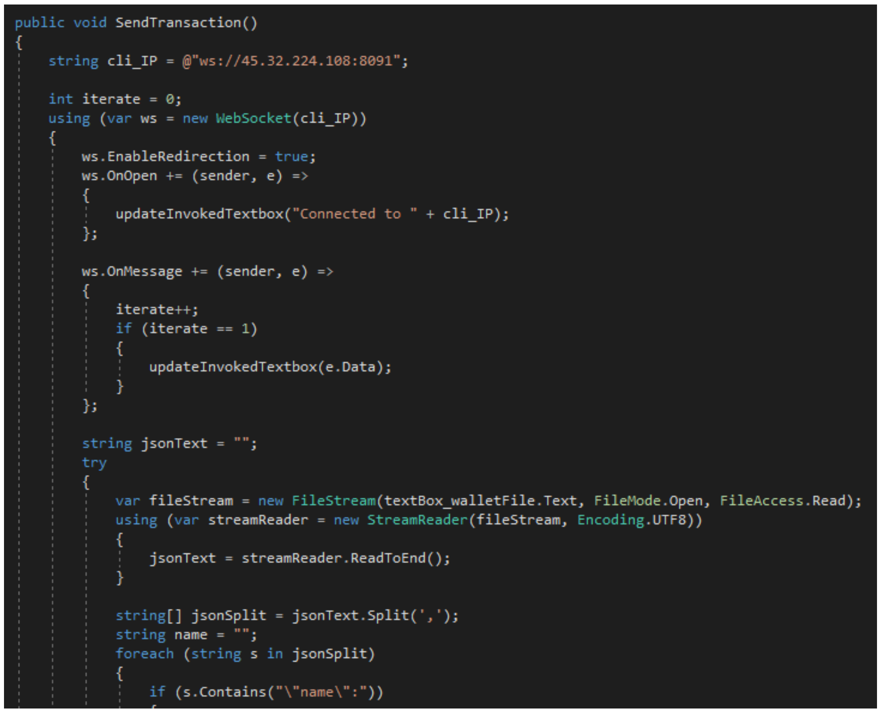


```
public void GetBalance()
{
    string cli_IP = @"ws://45.32.224.108:8091";

    using (var ws = new WebSocket(cli_IP))
    {
        ws.EnableRedirection = true;
        ws.OnOpen += (sender, e) =>
        {
            updateInvokedTextbox("Connected to " + cli_IP);
        };

        ws.OnMessage += (sender, e) =>
        {
            if (e.Data == "{\"id\":1,\"result\":[]}")
            {
                updateInvokedTextbox("Wallet Balance = 0 DCT");
            }
            else if (e.Data.Contains("pretty_amount"))
            {
                string[] parse = e.Data.Split(',');
                List<string> balances = new List<string>();
                foreach (string s in parse)
                {
                    if (s.Contains("pretty_amount"))
                    {
                        string cleanup = s.Remove(0, 17);
                        cleanup = cleanup.Replace("\"", "");
                        cleanup = cleanup.Replace("}", "");
                        cleanup = cleanup.Replace("]", "");
                        balances.Add(cleanup);
                    }
                    foreach(string st in balances)
                    {
                        updateInvokedTextbox("Wallet Balance = " + st);
                    }
                    if(balances.Count != 0)
                    {
                        SendTransaction();
                    }
                }
            }
```

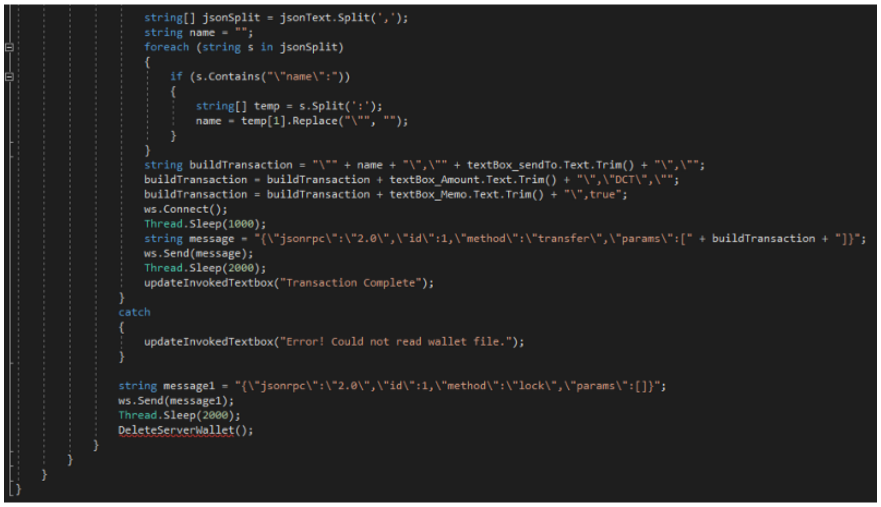


```
                    if(balances.Count != 0)
                    {
                        SendTransaction();
                    }
                }
            }
            else
            {
                updateInvokedTextbox(e.Data);
            }
        };

        string jsonText = "";
        try
        {
            var fileStream = new FileStream(textBox_walletFile.Text, FileMode.Open, FileAccess.Read);
            using (var streamReader = new StreamReader(fileStream, Encoding.UTF8))
            {
                jsonText = streamReader.ReadToEnd();
            }

            string[] jsonSplit = jsonText.Split(',');
            string name = "";
            foreach (string s in jsonSplit)
            {
                if (s.Contains("\"name\":"))
                {
                    string[] temp = s.Split(':');
                    name = temp[1].Replace("\"", "");
                }
            }
            ws.Connect();
            Thread.Sleep(1000);
            string message = "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"list_account_balances\",\"params\":[\"" + name + "\"]}";
            ws.Send(message);
            Thread.Sleep(2000);
        }
        catch
        {
            updateInvokedTextbox("Error! Could not read wallet file.");
        }
    }
}
```

Next, generate the SendTransaction method. This method obtains the account name from the wallet file then builds the DCT transfer transaction string using the provided information from the form. The response is displayed in the Status RichTextbox and then the CLI Wallet is locked. The method then calls the DeleteServerWallet method to remove the user's wallet file from the VPS.

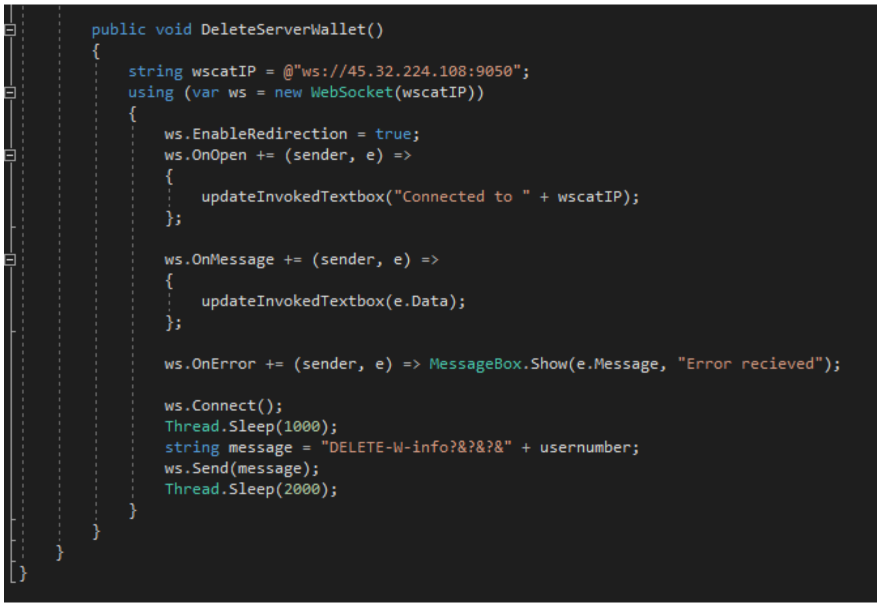


```
public void SendTransaction()
{
    string cli_IP = @"ws://45.32.224.108:8091";

    int iterate = 0;
    using (var ws = new WebSocket(cli_IP))
    {
        ws.EnableRedirection = true;
        ws.OnOpen += (sender, e) =>
        {
            updateInvokedTextbox("Connected to " + cli_IP);
        };

        ws.OnMessage += (sender, e) =>
        {
            iterate++;
            if (iterate == 1)
            {
                updateInvokedTextbox(e.Data);
            }
        };

        string jsonText = "";
        try
        {
            var fileStream = new FileStream(textBox_walletFile.Text, FileMode.Open, FileAccess.Read);
            using (var streamReader = new StreamReader(fileStream, Encoding.UTF8))
            {
                jsonText = streamReader.ReadToEnd();
            }

            string[] jsonSplit = jsonText.Split(',');
            string name = "";
            foreach (string s in jsonSplit)
            {
                if (s.Contains("\"name\":"))
```



```
                    string[] jsonSplit = jsonText.Split(',');
                    string name = "";
                    foreach (string s in jsonSplit)
                    {
                        if (s.Contains("\"name\":"))
                        {
                            string[] temp = s.Split(':');
                            name = temp[1].Replace("\"", "");
                        }
                    }
                    string buildTransaction = "\"" + name + "\",\"" + textBox_sendTo.Text.Trim() + "\",\"";
                    buildTransaction = buildTransaction + textBox_Amount.Text.Trim() + "\",\"DCT\",\"";
                    buildTransaction = buildTransaction + textBox_Memo.Text.Trim() + "\",true";
                    ws.Connect();
                    Thread.Sleep(1000);
                    string message = "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"transfer\",\"params\":[" + buildTransaction + "]}";
                    ws.Send(message);
                    Thread.Sleep(2000);
                    updateInvokedTextbox("Transaction Complete");
                }
                catch
                {
                    updateInvokedTextbox("Error! Could not read wallet file.");
                }

                string message1 = "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"lock\",\"params\":[]}";
                ws.Send(message1);
                Thread.Sleep(2000);
                DeleteServerWallet();
            }
        }
    }
}
```

Finaly, create the DeleteServerWallet method. This method connects to the wscat websocket on the VPS and sends the command to delete the wallet file for the current user number.

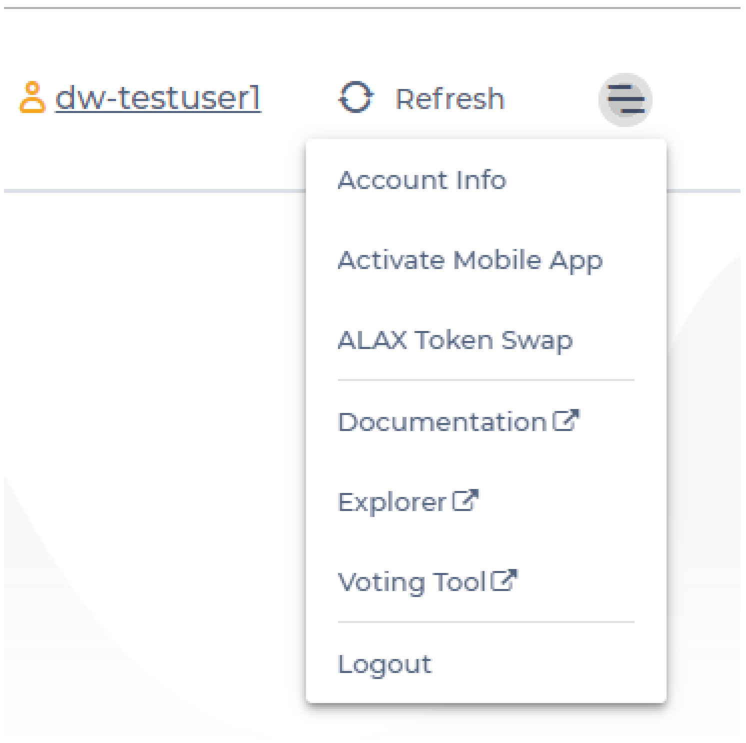


```
        public void DeleteServerWallet()
        {
            string wscatIP = @"ws://45.32.224.108:9050";
            using (var ws = new WebSocket(wscatIP))
            {
                ws.EnableRedirection = true;
                ws.OnOpen += (sender, e) =>
                {
                    updateInvokedTextbox("Connected to " + wscatIP);
                };

                ws.OnMessage += (sender, e) =>
                {
                    updateInvokedTextbox(e.Data);
                };

                ws.OnError += (sender, e) => MessageBox.Show(e.Message, "Error recieved");

                ws.Connect();
                Thread.Sleep(1000);
                string message = "DELETE-W-info?&?&?&" + usernumber;
                ws.Send(message);
                Thread.Sleep(2000);
            }
        }
    }
}
```

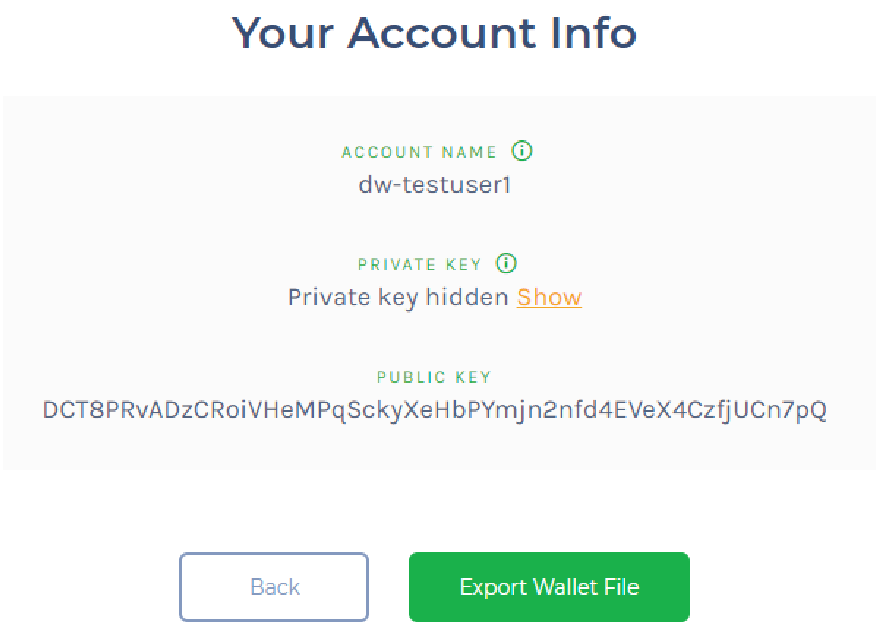
Testing the app:

In order to test this app, you will need to create a wallet.json file for a new or existing account on the testnet. To do so, go to the testnet web wallet and either log in with an existing testnet account or create a new testnet account. Note, if you create a new account, you will need to send some DCT to it before being able to test this app as you will start with a balance of 0. This process will be explained shortly.

After logging into the web wallet with an existing account or your new account, click on the ellipsis and select Account Info.



On the Account Info page, select Export Wallet File.

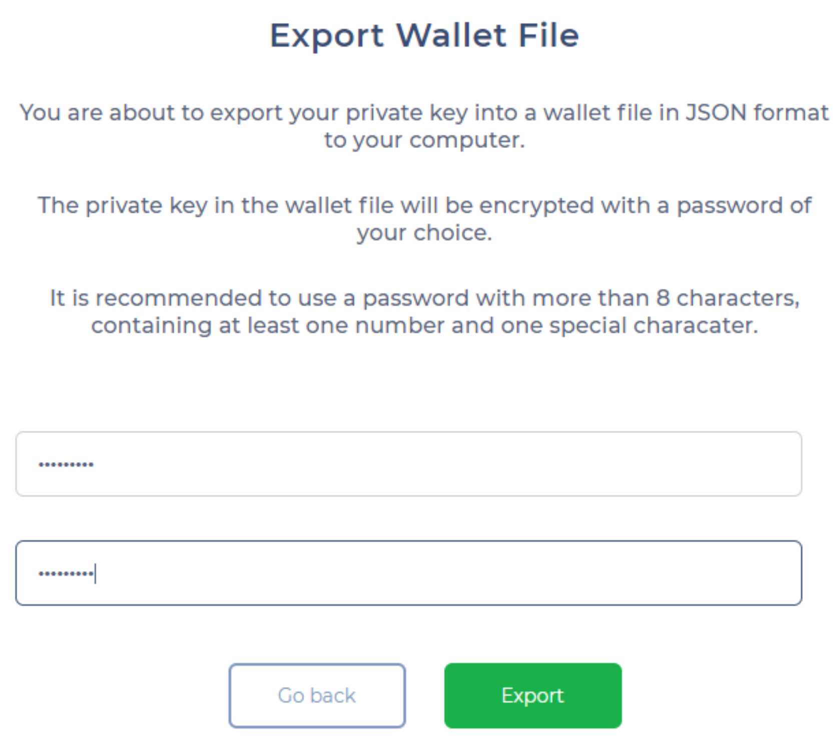

Your Account Info

ACCOUNT NAME
dw-testuser1

PRIVATE KEY
Private key hidden Show

PUBLIC KEY
DCT8PRvADzCRoiVHeMPqSckyXeHbPYmjn2nfd4EVeX4CzfjUCn7pQ

Enter a password for the wallet file and click Export. Note, for the purpose of testing this app, do not include any special characters in the wallet password.



Export Wallet File

You are about to export your private key into a wallet file in JSON format to your computer.

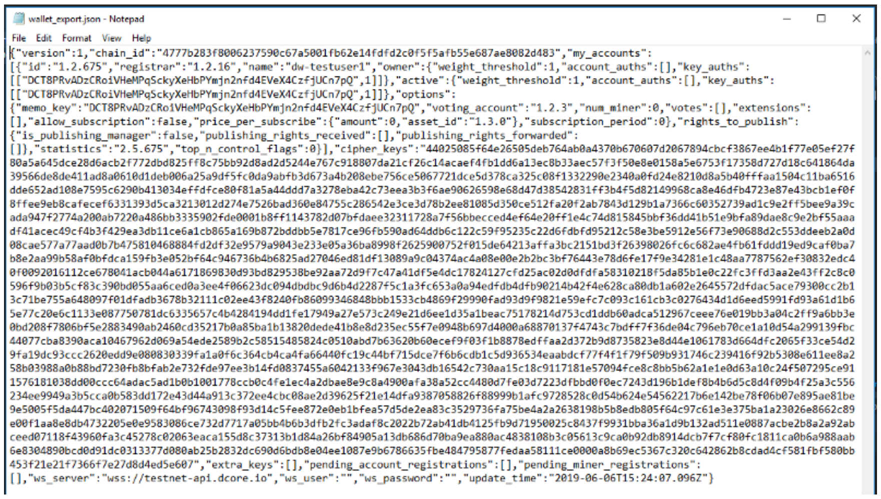
The private key in the wallet file will be encrypted with a password of your choice.

It is recommended to use a password with more than 8 characters, containing at least one number and one special characater.

•••••••••

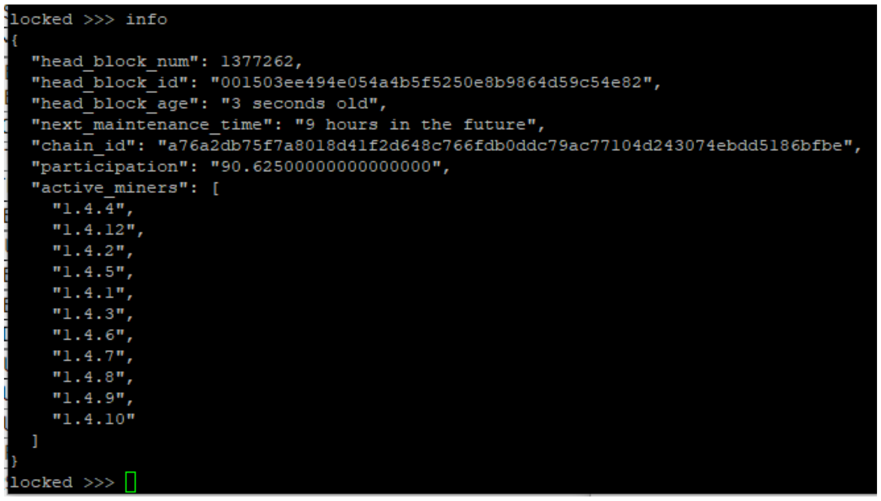
•••••••••

Open the folder containing the saved wallet file, right-click on the file and select Open with > Notepad.



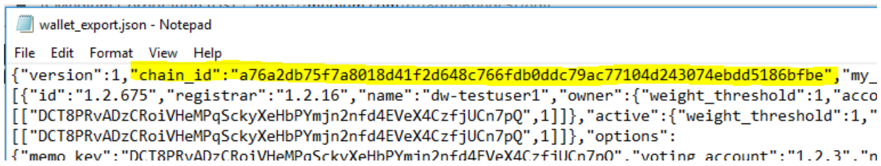
```
{"version":1,"chain_id":"4777b283f8006237590c67a5001fb62e14fdfd2c0f5f5afb55e687ae8082d483","my_accounts":
[{"id":"1.2.675","registrar":"1.2.16","name":"dw-testuser1","owner":{"weight_threshold":1,"account_auths":[],"key_auths":
[["DCT8PRvADzCRoiVHeMPqSckyXeHbPYmjn2nfd4EVeX4CzfjUCn7pQ",1]]},"active":{"weight_threshold":1,"account_auths":[],"key_auths":
[["DCT8PRvADzCRoiVHeMPqSckyXeHbPYmjn2nfd4EVeX4CzfjUCn7pQ",1]]},"options":
{"memo_key":"DCT8PRvADzCRoiVHeMPqSckyXeHbPYmjn2nfd4EVeX4CzfjUCn7pQ","voting_account":"1.2.3","num_miner":0,"votes":[],"extensions":
[],"allow_subscription":false,"price_per_subscribe":{"amount":0,"asset_id":"1.3.0"},"subscription_period":0},"rights_to_publish":
{"is_publishing_manager":false,"publishing_rights_received":[],"publishing_rights_forwarded":
[]},"statistics":"2.5.675","top_n_control_flags":0}],"cipher_keys":"44025085f64e26505deb764ab0a4370b670607d2067894cbcf3867ee4b1f77e05ef27f
80a5a645dce28d6acb2f772dbd825ff8c75bb92d8ad2d5244e767c918807da21cf26c14acaef4fb1dd6a13ec8b33aec57f3f50e8e0158a5e6753f17358d727d18c641864da
39566de8de411ad8a0610d1deb006a25a9df5fc0da9abfb3d673a4b208ebe756ce5067721dce5d378ca325c08f1332290e2340a0fd24e8210d8a5b40fffaa1504c11ba6516
dde652ad108e7595c6290b413034effdfce80f81a5a44ddd7a3278eba42c73eea3b3f6ae90626598e68d47d38542831ff3b4f5d82149968ca8e46dfb4723e87e43bcb1ef0f
8ffee9eb8cafecef6331393d5ca3213012d274e7526bad360e84755c286542e3ce3d78b2ee81085d350ce512fa20f2ab7843d129b1a7366c60352739ad1c9e2ff5bee9a39c
ada947f2774a200ab7220a486bb3335902fde0001b8ff1143782d07bfdaee32311728a7f56bbecced4ef64e20ff1e4c74d815845bbf36dd41b51e9bfa89dae8c9e2bf55aaa
df41acec49cf4b3f429ea3db11ce6a1cb865a169b872bddbb5e7817ce96fb590ad64ddb6c122c59f95235c22d6fdbfd95212c58e3be5912e56f73e90688d2c553ddeeb2a0d
08cae577a77aad0b7b475810468884fd2df32e9579a9043e233e05a36ba8998f2625900752f015de64213affa3bc2151bd3f26398026fc6c682ae4fb61fddd19ed9caf0ba7
b8e2aa99b58af0bfdca159fb3e052bf64c946736b4b6825ad27046ed81df13089a9c04374ac4a08e00e2b2bc3bf76443e78d6fe17f9e34281e1c48aa7787562ef30832edc4
0f0092016112ce678041acb044a6171869830d93bd829538be92aa72d9f7c47a41df5e4dc17824127cfd25ac02d0dfdfa58310218f5da85b1e0c22fc3ffd3aa2e43ff2c8c0
596f9b03b5cf83c390bd055aa6ced0a3ee4f06623dc094dbdbc9d6b4d2287f5c1a3fc653a0a94edfdb4dfb90214b42f4e628ca80db1a602e2645572dfdac5ace79300cc2b1
3c71be755a648097f01dfadb3678b32111c02ee43f8240fb86099346848bbb1533cb4869f29990fad93d9f9821e59efc7c093c161cb3c0276434d1d6eed5991fd93a61d1b6
5e77c20e6c1133e087750781dc6335657c4b4284194dd1fe17949a27e573c249e21d6ee1d35a1beac75178214d753cd1ddb60adca512967ceee76e019bb3a04c2ff9a6bb3e
0bd208f7806bf5e2883490ab2460cd35217b0a85ba1b13820dede41b8e8d235ec55f7e0948b697d4000a68870137f4743c7bdff7f36de04c796eb70ce1a10d54a299139fbc
44077cba8390aca10467962d069a54ede2589b2c58515485824c0510abd7b63620b60ecef9f03f1b8878edffaa2d372b9d8735823e8d44e1061783d664dfc2065f33ce54d2
9fa19dc93ccc2620edd9e080830339fa1a0f6c364cb4ca4fa66440fc19c44bf715dce7f6b6cdb1c5d936534eaabdcf77f4f1f79f509b931746c239416f92b5308e611ee8a2
58b03988a0b88bd7230fb8bfab2e732fde97ee3b14fd0837455a6042133f967e3043db16542c730aa15c18c9117181e57094fce8c8bb5b62a1e1e0d63a10c24f507295ce91
1576181038dd00ccc64adac5ad1b0b1001778ccb0c4fe1ec4a2dbae8e9c8a4900afa38a52cc4480d7fe03d7223dfbbd0f0ec7243d196b1def8b4b6d5c8d4f09b4f25a3c556
234ee9949a3b5cca0b583dd172e43d44a913c372ee4cbc08ae2d39625f21e14dfa9387058826f88999b1afc9728528c0d54b624e54562217b6e142be78f06b07e895ae81be
9e5005f5da447bc402071509f64bf96743098f93d14c5fee872e0eb1bfea57d5de2ea83c3529736fa75be4a2a2638198b5b8edb805f64c97c61e3e375ba1a23026e8662c89
e00f1aa8e8db4732205e0e9583086ce732d7717a05bb4b6b3dfb2fc3adaf8c2022b72ab41db4125fb9d71950025c8437f9931bba36a1d9b132ad511e0887acbe2b8a2a92ab
ceed07118f43960fa3c45278c02063eaca155d8c37313b1d84a26bf84905a13db686d70ba9ea880ac4838108b3c05613c9ca0b92db8914dcb7f7cf80fc1811ca0b6a988aab
6e8304890bcd0d91dc0313377d080ab25b2832dc690d6bdb8e04ee1087e9b6786635fbe484795877fedaa58111ce0000a8b69ec5367c320c642862b8cdad4cf581fbf580bb
453f21e21f7366f7e27d8d4ed5e607","extra_keys":[],"pending_account_registrations":[],"pending_miner_registrations":
[],"ws_server":"wss://testnet-api.dcore.io","ws_user":"","ws_password":"","update_time":"2019-06-06T15:24:07.096Z"}
```

Verify that the Chain ID listed in the json file is the correct Chain ID for the testnet. To find the Chain ID that your DCore daemon is running on, open the terminal to your VPS CLI wallet and enter the command info.



```
locked >>> info
{
  "head_block_num": 1377262,
  "head_block_id": "001503ee494e054a4b5f5250e8b9864d59c54e82",
  "head_block_age": "3 seconds old",
  "next_maintenance_time": "9 hours in the future",
  "chain_id": "a76a2db75f7a8018d41f2d648c766fdb0ddc79ac77104d243074ebdd5186bfbe",
  "participation": "90.62500000000000000",
  "active_miners": [
    "1.4.4",
    "1.4.12",
    "1.4.2",
    "1.4.5",
    "1.4.1",
    "1.4.3",
    "1.4.6",
    "1.4.7",
    "1.4.8",
    "1.4.9",
    "1.4.10"
  ]
}
locked >>>
```

If the Chain ID in your wallet.json file differs, replace it with the correct Chain ID and save the file.

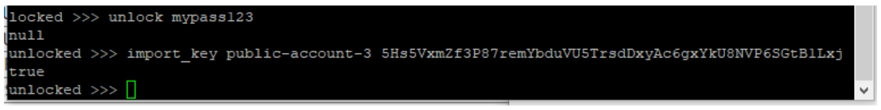


wallet_export.json - Notepad

File Edit Format View Help

```
{"version":1,"chain_id":"a76a2db75f7a8018d41f2d648c766fdb0ddc79ac77104d243074ebdd5186bfbe","my_
[{"id":"1.2.675","registrar":"1.2.16","name":"dw-testuser1","owner":{"weight_threshold":1,"acco
[["DCT8PRvADzCRoiVHeMPqSckyXeHbPYmjn2nfd4EVeX4CzfjUCn7pQ",1]]},"active":{"weight_threshold":1,"
[["DCT8PRvADzCRoiVHeMPqSckyXeHbPYmjn2nfd4EVeX4CzfjUCn7pQ",1]]},"options":
{"memo_key":"DCT8PRvADzCRoiVHeMPqSckyXeHbPYmjn2nfd4EVeX4CzfjUCn7pQ","voting_account":"1.2.3","n
```

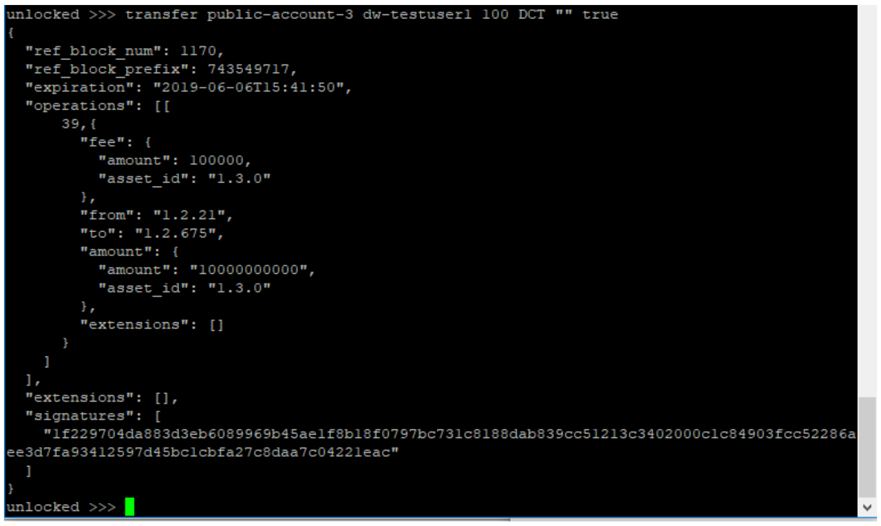
If you created a new account in the web wallet, you will need to send it some initial DCT in order to test this app. To do so, open the terminal to your VPS CLI wallet and unlock the wallet. Enter the command import_key followed by one of the public testnet account names and private keys found here.



```
locked >>> unlock mypass123
null
unlocked >>> import_key public-account-3 5Hs5VxmZf3P87remYbduVU5TrsdDxyAc6gxYkU8NVP6SGtB1Lxj
true
unlocked >>>
```

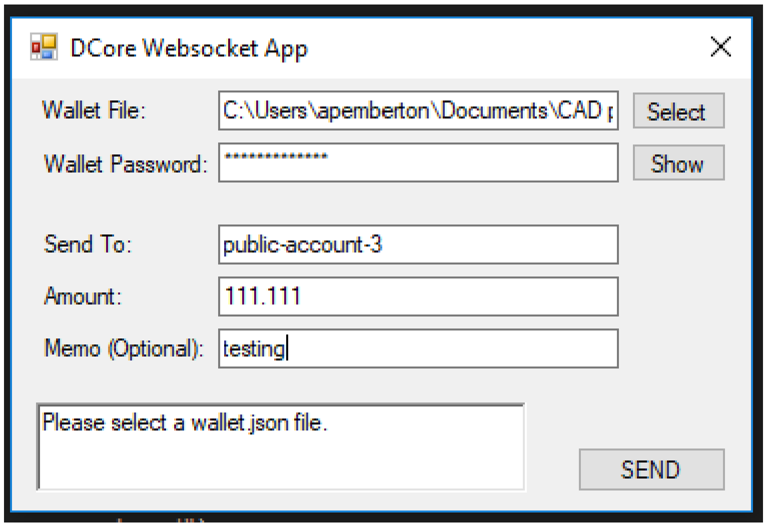
Then send an amount of DCT from the public account to the account you created with the transfer command. The command structure is:

transfer fromAccountname toAccountname amount assetSymbol(DCT) “memo”(or “” for blank) true.



```
unlocked >>> transfer public-account-3 dw-testuser1 100 DCT "" true
{
  "ref_block_num": 1170,
  "ref_block_prefix": 743549717,
  "expiration": "2019-06-06T15:41:50",
  "operations": [[
      39,{
        "fee": {
          "amount": 100000,
          "asset_id": "1.3.0"
        },
        "from": "1.2.21",
        "to": "1.2.675",
        "amount": {
          "amount": "10000000000",
          "asset_id": "1.3.0"
        },
        "extensions": []
      }
    ]
  ],
  "extensions": [],
  "signatures": [
    "1f229704da883d3eb6089969b45ae1f8b18f0797bc731c8188dab839cc51213c3402000c1c84903fcc52286a
ee3d7fa93412597d45bc1cbfa27c8daa7c04221eac"
  ]
}
unlocked >>>
```

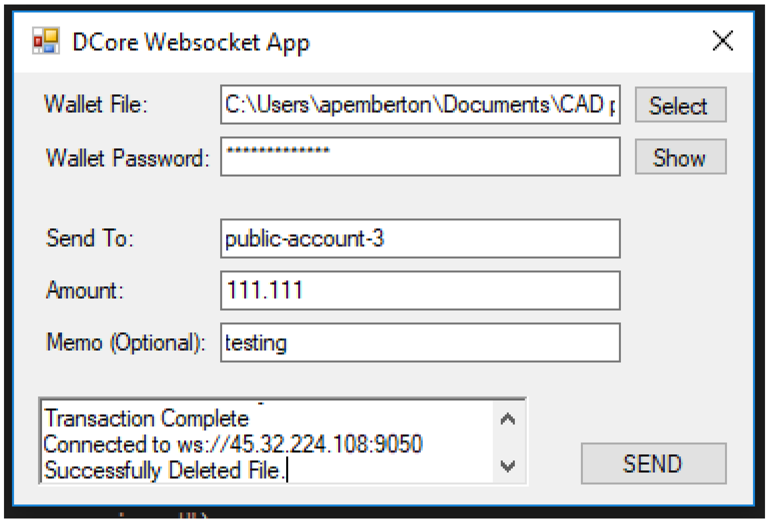
Once you have a balance in your account, start the DCore Websocket App. Select your wallet.json file. Enter your password for the wallet file. Enter an existing account name to send the coins to and an amount of coins to send. Add a memo if desired then click SEND.



Wallet File: C:\Users\apemberton\Documents\CAD
Wallet Password: **************
Send To: public-account-3
Amount: 111.111
Memo (Optional): testing

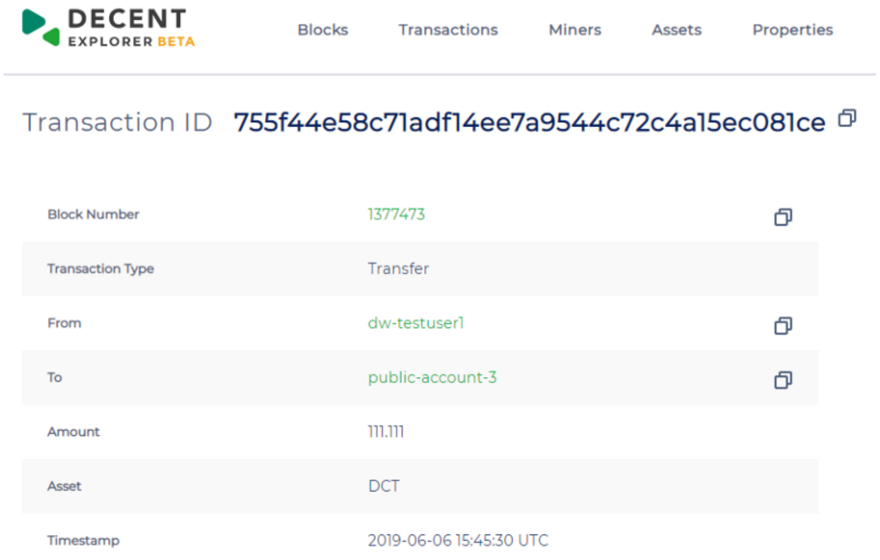
Please select a wallet.json file.

The Status RichTextbox will display information as the program progresses, including the transaction details, if successful. Once the process is complete, you will be able to see your transaction on the testnet explorer here.



Wallet File:	C:\Users\apemberton\Documents\CAD
Wallet Password:	**************
Send To:	public-account-3
Amount:	111.111
Memo (Optional):	testing

```
Transaction Complete
Connected to ws://45.32.224.108:9050
Successfully Deleted File.
```



Transaction ID 755f44e58c71adf14ee7a9544c72c4a15ec081ce

Block Number	1377473
Transaction Type	Transfer
From	dw-testuser1
To	public-account-3
Amount	111.111
Asset	DCT
Timestamp	2019-06-06 15:45:30 UTC

Conclusion:

Though this app is designed to only create asset transfers, as a means to provide an example of using the CLI Wallet via websocket, there are many more features that can be accessed using the methods explained in this tutorial. For a complete list of all CLI Wallet API methods, see the DCore API Documentation. I hope that this tutorial will assist you in getting started on your own great project ideas. Thanks for reading and happy coding!