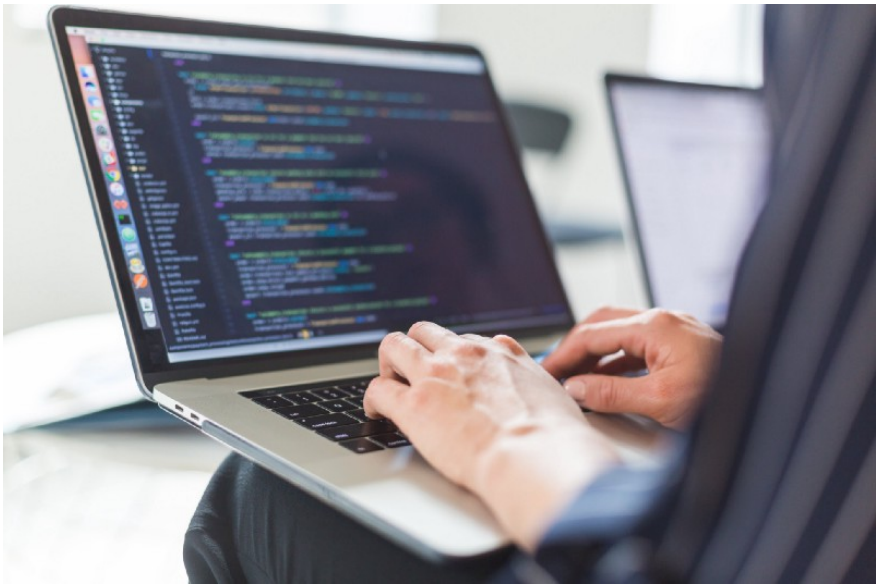
How to: Query DCore Blockchain Daemon API in C#

With a sample Windows Form app tutorial

Aaron Pemberton

Apr 7 · 12 min read ★



The usage of blockchain technologies continues to grow with adoption in a wide range of industries from FinTech to aviation, charities to entertainment. The widespread acknowledgement of its potential and possibilities is generating many new entrepreneurs with creative visions of developing projects that utilize blockchain technology to improve traditional industry methods or invent new, superior ones.

Attempting to communicate with a blockchain can be a daunting task for some, especially those with little programming experience. With the goal of assisting those who feel apprehensive, this tutorial will help

familiarize anyone with the process and hopefully allow them to feel comfortable enough to begin incorporating blockchain tech into their own project ideas. In this tutorial, we will cover some basic terminology and fundamental information, setting up a VPS to run DCore Daemon listening for API calls, and create a simple Windows Form app in C# using Microsoft Visual Studio which can be used to make queries to your server. The sample app should help to explain how to structure the API calls as well as visualize the response to better understand what steps you will need to take to obtain the information you seek.

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

DCore has two main types of APIs, Daemon API and Wallet API. The Daemon API primarily is for querying data from the blockchain such as account info, balances, transaction history, asset information, content information, subscription information, etc. The Wallet API has a lot of the same functionality along with the ability to sign in to an account and send transactions on the network, send and receive messages, upload content, etc. Both HTTP (and HTTPS) and websocket (both ws and wss) communication protocols can be used for API calls. The main difference being that the HTTP protocol doesn't hold session status. So you can only utilize the database set from the Daemon API using HTTP. The Wallet API requires running the CLI Wallet which holds session data, so either communication protocol can be used with the Wallet API. This tutorial will only cover database Daemon API calls via HTTP. A complete list of available DCore API calls can be found here https://docs.decent.ch/developer/modules.html.

. . .

Basic Terminology:

Blockchain: The most basic explanation of what is a blockchain is, it's an immutable record of time-stamped data maintained by a distributed network of computers. The data stored in "blocks" is secured and linked to successive blocks using cryptographic methods. Blockchain also allows for passing of verified information from point to point in a safe manner.

Consensus Method: The method in which new blocks are accepted by the network. Different types of methods include: Proof of Work (PoW), Proof of Stake (PoS), and Delegated Proof of Stake (DPoS).

Delegated Proof of Stake (DPoS): DCore utilized DPoS. In a DPoS network, a select number of participants are actively confirming new blocks and maintaining the security of the network. Active participants are referred to as "Witnesses" or "Miners" and are voted on by anyone owning the networks asset. DPoS has the advantage of higher transactions per second compared to PoW along with being more

environmentally conscientious as the amount of electricity required to maintain the network is a mere fraction of a PoW network.

Virtual Private Server (VPS): A VPS is server with it's own operating system and resources within a larger server. Renting a VPS allows for sandbox testing of applications without the overhead or need to set up a dedicated server.

Software Development Kit (SDK): DCore's SDKs provide a set of tools and code samples to aid developers in creating software on its platform. DCore's SDKs can be found here https://github.com/DECENTfoundation.

Application Programming Interface (API): API is an interface that allows software to interact with other software. This is what enables communication between your application and the blockchain. DCore has two types of APIs, Daemon API and Wallet API. This tutorial will focus on the Daemon API. For more information on DCore's APIs see the documentation here https://docs.decent.ch/API/index.html

Daemon: A daemon is a computer program that runs as a background process instead of being directly controlled by an active user.

JavaScript Object Notation (JSON): JSON is a lightweight format for storing and sending data.

. . .

VPS Setup:

For this tutorial, I will be using a VPS from vultr.com. Vultr allows low priced, easy to set up VPSs and hourly rate billing. Feel free to use any service you find adequate, though.

Begin by creating a new server deployment with Ubuntu 18.04 x64 operating system and a minimum of 25G of drive space. Once your VPS is running, connect to it using Putty or any SSH client you prefer.

DCore requires more RAM during installation than most VPSs provide, so create a swapfile with 10G of space.

```
1] sudo fallocate -l 10G /swapfile
2] sudo chmod 600 /swapfile
3] sudo mkswap /swapfile
4] sudo swapon /swapfile
```

Make the swapfile permanent by editing the fstab file.

```
1] sudo cp /etc/fstab /etc/fstab.bak
2] echo '/swapfile none swap sw 0 0' | sudo tee -a
/etc/fstab
```

Next, start the `screen` command. This will allow the terminal to continue even if you loose connection between your SSH client and VPS. For more information on using screen, see this website. https://linuxize.com/post/how-to-use-linux-screen/

The following procedure to build and install DCore is the same from DECENT's GitHub https://github.com/DECENTfoundation/DECENT-Network, repeated here for convenience along with a couple comments.

Install prerequisites

```
1] sudo apt-get update
2] sudo apt-get install build-essential autotools-dev
automake autoconf libtool make cmake g++ flex bison doxygen
unzip wget git qt5-default qttools5-dev qttools5-dev-tools
libreadline-dev libcrypto++-dev libgmp-dev libssl-dev
libcurl4-openssl-dev libboost-all-dev
3] . /etc/os-release
4] export ARCH=`dpkg --print-architecture`
5] wget -nv -P /tmp
https://github.com/DECENTfoundation/pbc/releases/download/0.
5.14/libpbc_0.5.14-${ID}${VERSION_ID}_${ARCH}.deb
https://github.com/DECENTfoundation/pbc/releases/download/0.
5.14/libpbc-dev_0.5.14-${ID}${VERSION_ID}_${ARCH}.deb
6] sudo dpkg -i /tmp/libpbc*
```

Download DCore sources

```
1] mkdir ~/dev
2] cd dev
3] git clone https://github.com/DECENTfoundation/DECENT-
Network.git
4] cd DECENT-Network
5] git submodule update --init --recursive
```

Build and Install DCore

```
1] mkdir -p ~/dev/DECENT-Network-build
2] cd ~/dev/DECENT-Network-build
3] cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release
~/dev/DECENT-Network
4] cmake --build . --target all -- -j -l 3.0
5] cmake --build . --target install
```

Note, if you receive errors during the `cmake --build . --target all -- -j -l 3.0` command, run it without arguments: `cmake --build . --target all` This will take longer to build but is helpful on servers with limited resources.

After the installation is complete, use this command to start the DCore Daemon

```
1] /usr/local/bin/decentd
```

In most cases, it will take a few hours to completely sync with the network. Once DCore is fully synced, close the daemon with ctrl+c. Next, allow listening on port 8090 in the Universal Firewall.

```
1] sudo ufw allow 8090
```

Then restart the daemon with the arguments --rpc-endpoint <your VPS ip>:8090. For example:

```
1] /usr/local/bin/decentd --rpc-endpoint 45.32.214.126:8090
```

Your VPS is now running the DCore Daemon listening for connections on port 8090.

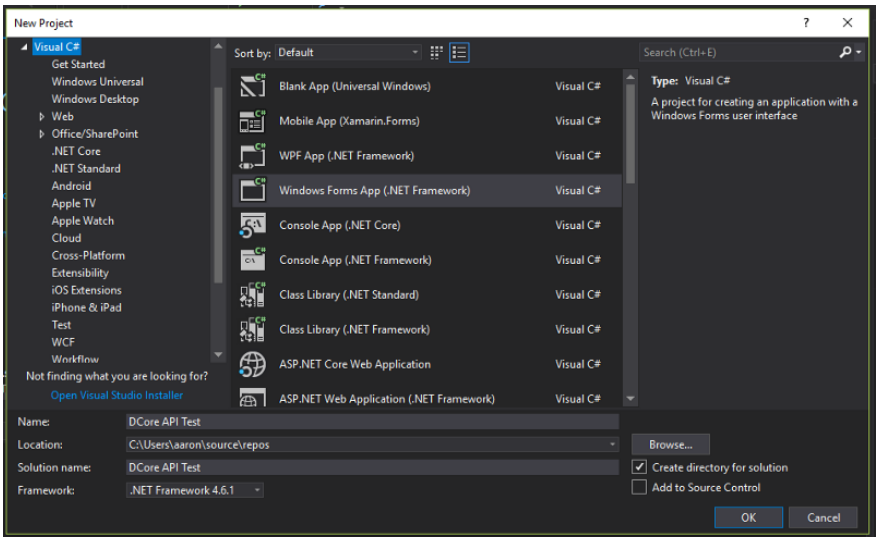
. . .

DCore API Test app:

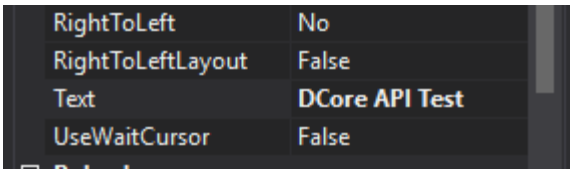
The complete Virtual Studio solution for this app as well as the built executable can be downloaded from GitHub at https://github.com/AaronPemberton/DCore-Test-App.

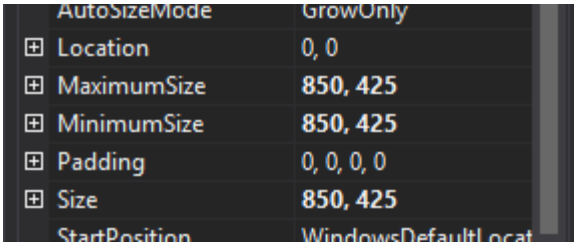
Form Layout:

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.



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 850, 425.





Insert the following into your blank form:

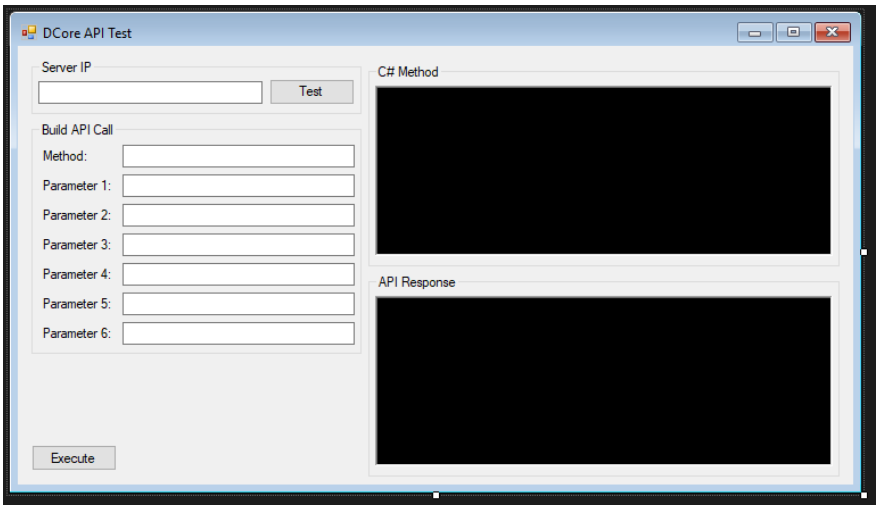
Add a GroupBox and change the Text to Server IP.
Inside the Server IP box, add a TextBox. Change the Name to textBox_IP.
Add a Button. Rename it button_Test and change the Text to Test.

And another GroupBox and change the Text to Build API
Add 7 Labels and change the Text of the first to Method: and the Text of the remainder to Parameter 1: through Parameter 6:.
Add 7 TextBoxes. Rename them textBox_Method and textBox_Parameter1 through textBox_Parameter6 corresponding with the adjacent label.

Add another GroupBox and change the text to C# Method.
Add a RichTextBox and change the Name to richTextBox_Method.
Change the RichTextBox BackColor to Desktop and ForeColor to Info.

Add another GroupBox and change the text to API Response.
Add a RichTextBox and change the Name to richTextBox_Response.
Change the RichTextBox BackColor to Desktop and ForeColor to Info.

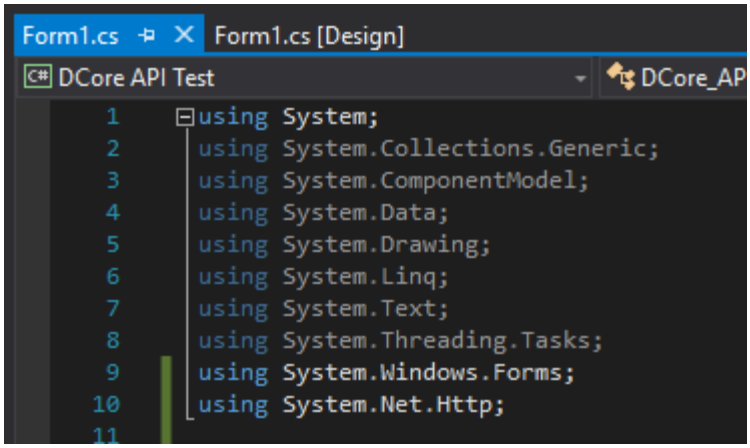
Finally, add a Button. Change the Name to button_Execute and the Text to Execute.



The form layout is now complete with a box to input the IP to connect to, boxes to input the method and parameter(s) to test, a section that will display the C# code for the method tested, and a box to show the response from the server. Next we will program these features.

Adding the Code:

Double-click the Test button. This will open the Form1.cs page. Near the top of the page, add a reference to the System.Net.Http namespace.



Next, add the following code to the button_Test_Click method. Note, because we are using the await operator to wait on the response, you will need to change this method to an async method.

```
22          private async void button_Test_Click(object sender, EventArgs e)
23          {
24              using (var httpClient = new HttpClient())
25              {
26                  try
27                  {
28                      var result = await httpClient.GetAsync(textBox_IP.Text);
29                      int StatusCode = (int)result.StatusCode;
30                      richTextBox_Response.Text = "Status Code: " + StatusCode;
31                  }
32                  catch
33                  {
34                      richTextBox_Response.Text = "ERROR! Could not connect to specified URL.";
35                  }
36              }
37
38          }
```

This will take the web address entered into textBox_IP and attempt to retrieve the status code from it. If successful, it will display the returned status code in the Response rich text box. Otherwise, it will display the error message.

Next, make a list of the parameter TextBoxes so that it can be looped through. To do this, initialize a new List<TextBox> and add the parameter TextBoxes to it with the following code.

```
13  namespace DCore_API_Test
14  {
15      public partial class Form1 : Form
16      {
17          List<TextBox> paramboxes = new List<TextBox>();
18
19          public Form1()
20          {
21              InitializeComponent();
22              paramboxes.Add(textBox_Parameter1);
23              paramboxes.Add(textBox_Parameter2);
24              paramboxes.Add(textBox_Parameter3);
25              paramboxes.Add(textBox_Parameter4);
26              paramboxes.Add(textBox_Parameter5);
27              paramboxes.Add(textBox_Parameter6);
28
29          }
30
```

Switch back to the Form1.cs [Design] tab and double-click on the Execute button. When this button is clicked, we want to check if the Method textbox was accidentally left empty and to build a list of the parameters given, if any. Add a call to the method displayCode , sending the parameters list. You will get a warning as this method does not exist. This will be added in the next section. To do this, add the following code to the button_Execute_Click method.

```
50
51      private void button_Execute_Click(object sender, EventArgs e)
52      {
53
54          if (string.IsNullOrWhiteSpace(textBox_Method.Text))
55          {
56              richTextBox_Response.Text = "ERROR! No Method to test!";
57          }
58          else
59          {
60              List<string> parameters = new List<string>();
61              for (int i = 0; i < 6; i++)
62              {
63                  if (!string.IsNullOrWhiteSpace(paramboxes[i].Text))
64                  {
65                      parameters.Add(paramboxes[i].Text);
66                  }
67              }
68
69              displayCode(parameters);
70          }
71      }
```

Now create a new Method named displayCode that accepts a List. The string of parameters (values) will be used not only to display the API call but also in the method executing the API call. Set it to an empty string (“”) to begin with, that way if there aren’t any parameters needed, this section of the API call will remain empty. The next portion checks if there are any parameters given and if so it creates the value string in the format needed to be used in the API call. For example: “params”:[“1.2.15”,[“1.3.0”,”1.3.51"]]

```
73      private void displayCode(List<string> paramList)
74      {
75          string values = "";
76          if (paramList.Count != 0)
77          {
78              int x = 0;
79              values = ",\"params\":[";
80              foreach (string item in paramList)
81              {
82                  if (!item.Contains(','))
83                  {
84                      values = values + "\"" + item.Trim() + "\"]";
85                  }
86                  else
87                  {
88                      string[] temp = item.Split(',');
89                      values = values + "[";
90                      foreach (string subitem in temp)
91                      {
92                          values = values + "\"" + subitem.Trim() + "\",";
93                      }
94                      values = values + "]";
95                      values = values.Remove(values.Length - 2, 1);
96                  }
97                  x++;
98                  if (x != paramList.Count)
99                  {
100                     values = values + ",";
101                 }
102             }
103         }
```

Still inside the displayCode Method, generate the example code to be displayed in the C# Method box by adding the following code. And finish the displayCode Method with a call to the callAPI method, sending it the values string.

```
richTextBox_Method.Clear();
string tab = "  ";
richTextBox_Method.AppendText("private async Task<string> APICall_" + textBox_Method.Text + "()\n");
richTextBox_Method.AppendText("{\n");
richTextBox_Method.AppendText(tab + "using (var httpClient = new HttpClient())\n");
richTextBox_Method.AppendText(tab + "{\n");
richTextBox_Method.AppendText(tab + tab + "try\n");
richTextBox_Method.AppendText(tab + tab + "{\n");
richTextBox_Method.AppendText(tab + tab + tab + "using  (var request = new HttpRequestMessage(new HttpMethod(\"POST\"), \"");
richTextBox_Method.AppendText(textBox_IP.Text + "\"))\n");
richTextBox_Method.AppendText(tab + tab + tab + "{\n");
string format = values.Replace("\"", "\\\"");
richTextBox_Method.AppendText(tab + tab + tab + "string content = \"{\\\"jsonrpc\\\":\\\"2.0\\\",\\\"id\\\":1,\\\"method\\\":\\\"");
richTextBox_Method.AppendText(textBox_Method.Text + "\\\"" + format + "}\";\n");
richTextBox_Method.AppendText(tab + tab + tab + "request.Content = new StringContent(content, Encoding.UTF8, \"application/x-www-form-urlencoded\");\n");
richTextBox_Method.AppendText(tab + tab + tab + "var response = await httpClient.SendAsync(request);\n");
richTextBox_Method.AppendText(tab + tab + tab + "response.EnsureSuccessStatusCode();\n");
richTextBox_Method.AppendText(tab + tab + tab + "string responseBody = await response.Content.ReadAsStringAsync();\n");
richTextBox_Method.AppendText(tab + tab + tab + "return responseBody;\n");
richTextBox_Method.AppendText(tab + tab + tab + "}\n");
richTextBox_Method.AppendText(tab + tab + "}\n");
richTextBox_Method.AppendText(tab + tab + "catch\n");
richTextBox_Method.AppendText(tab + tab + "{\n");
richTextBox_Method.AppendText(tab + tab + tab + "return \"Error\";\n");
richTextBox_Method.AppendText(tab + tab + "}\n");
richTextBox_Method.AppendText(tab + "}\n");
richTextBox_Method.AppendText("}\n");

callAPI(values);
```

Create a new Method called callAPI that accepts a string. This method will make the API call to the DCore server with the given query method and parameters. The response will be in JSON format. To make the response more legible, we will send the response to another method called FormatJson, which we will create next.

```
private async void callAPI(string parameterString)
{
    using (var httpClient = new HttpClient())
    {
        try
        {
            using (var request = new HttpRequestMessage(new HttpMethod("POST"), textBox_IP.Text))
            {
                string content = "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"" + textBox_Method.Text + "\"" + parameterString + "}";
                request.Content = new StringContent(content, Encoding.UTF8, "application/x-www-form-urlencoded");
                var response = await httpClient.SendAsync(request);
                response.EnsureSuccessStatusCode();
                string responseBody = await response.Content.ReadAsStringAsync();
                richTextBox_Response.Text = FormatJson(responseBody);
            }
        }
        catch
        {
            richTextBox_Response.AppendText("An unknown error occured");
        }
    }
}
```

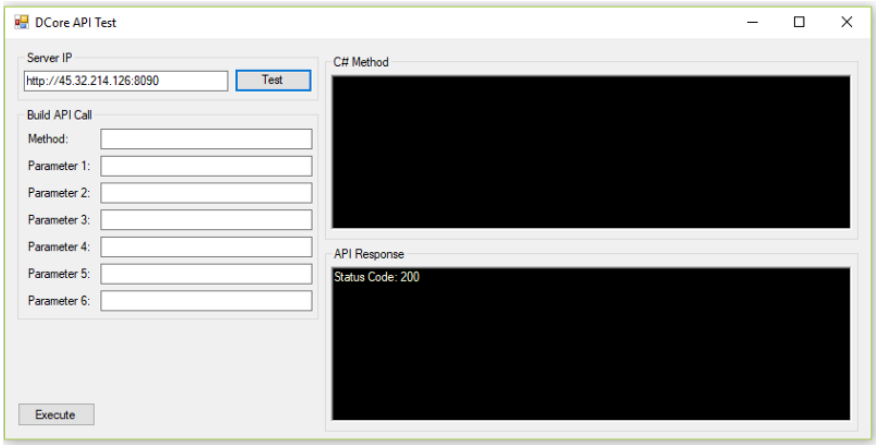
To format the JSON string into a more human readable text, add the following code, provided by Vince Panuccio in a response on StackOverflow.

```
private const string INDENT_STRING = "    ";
static string FormatJson(string json)
{

    int indentation = 0;
    int quoteCount = 0;
    var result =
        from ch in json
        let quotes = ch == '"' ? quoteCount++ : quoteCount
        let lineBreak = ch == ',' && quotes % 2 == 0 ? ch + Environment.NewLine + String.Concat(Enumerable.Repeat(INDENT_STRING, indentation)) : null
        let openChar = ch == '{' || ch == '[' ? ch + Environment.NewLine + String.Concat(Enumerable.Repeat(INDENT_STRING, ++indentation)) : ch.ToString()
        let closeChar = ch == '}' || ch == ']' ? Environment.NewLine + String.Concat(Enumerable.Repeat(INDENT_STRING, --indentation)) + ch : ch.ToString()
        select lineBreak == null
                    ? openChar.Length > 1
                        ? openChar
                        : closeChar
                    : lineBreak;

    return String.Concat(result);
}
```

Set the build type to Release and build the project. Navigate to the folder where you saved the project and locate the \DCore API Test\bin\Release\ folder. Run the DCore API Test.exe application. Note, if you gave the project a different name other than DCore API Test, this .exe file will have that name instead.

Using the DCore API Test app:

Your app is now ready to send API calls to your DCore server set up previously. Enter your server IP address (including http:// or https://) into the Server IP field. Click the Test button and you should get a Status Code: 200 response.



The DCore Database API information can be found at https://docs.decent.ch/developer/group___database_a_p_i.html. Use that as a guide to understand what parameters are required for each method. As an example, try querying the get_account_by_name method. As you can see from the API documentation, the method name is get_account_by_name, it requires one parameter which is a single account name, and it will return the account info for that account.

optional< account_object > graphene::app::database_api::get_account_by_name (string name) const

Get an account by name.

Parameters

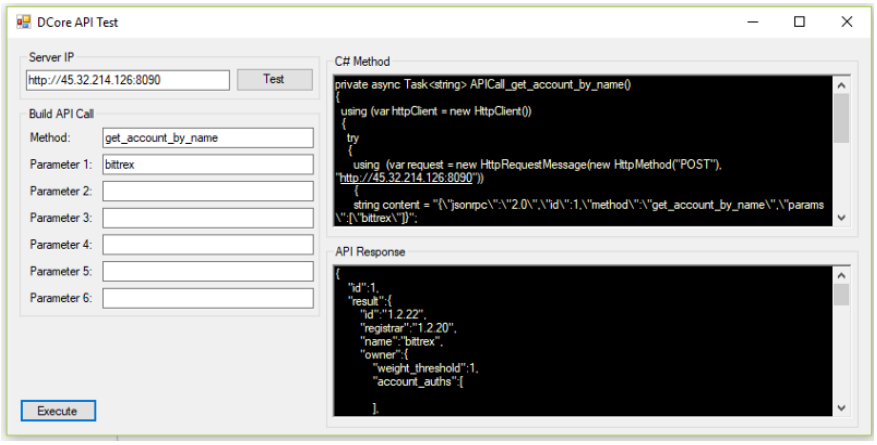
name name of the account to retrieve

Returns

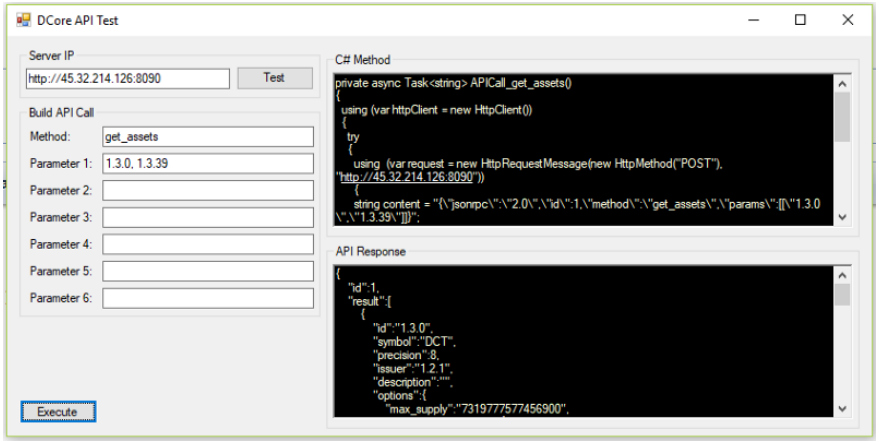
the `account_object` corresponding to the provided `name`, or `null` if no matching content was found

Definition at line **669** of file **database_api.cpp**.

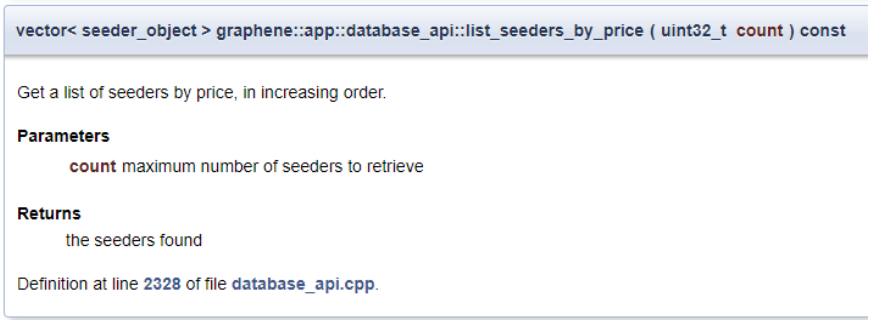
Enter get_account_by_name in the Method field and bittrex in the 1st parameter field then click Execute. The C# Method box will automatically populate with the C# code for a new method with these parameters. This code can be copied straight into any program you are creating and will return the JSON response when called. The API Response box displays the response from your DCore server. In this example, you can scroll through and see all relevant info about this account including its ID number (1.2.22).

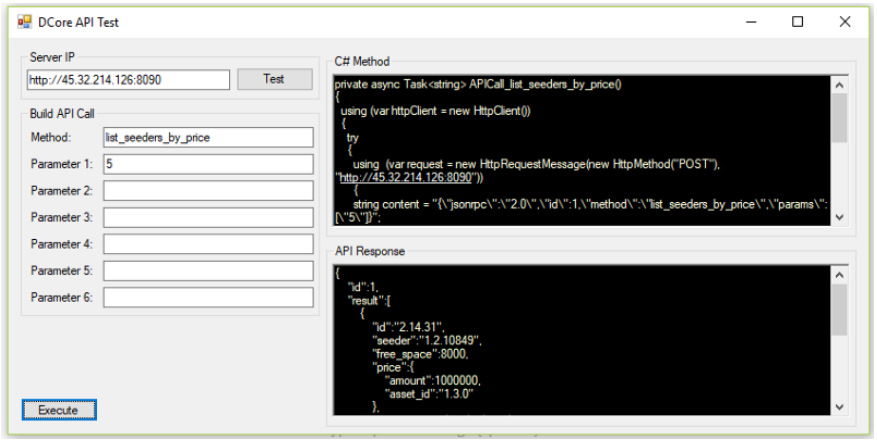


Now try the get_assets method. As you can see from the API documentation, this method requires a list of asset ID's. In the 1st parameter field, enter 1.3.0, 1.3.39 then click Execute. Again, the reusable code is generated in the C# Method box and by scrolling through the API Response box, we can see that asset ID 1.3.0 is the core asset DCT and asset ID 1.3.39 is the custom, user issued asset ALX.



As stated previously in this article, DCore enables content distribution using IPFS. The content submitted is stored by “seeders” on the network who get paid per MB of content verifiably stored per day. A list of available seeders can be obtained with an API call as well. As you can see from the documentation, list_seeders_by_price requires one parameter, the number of results to show. Enter list_seeders_by_price into the Method field and 3 into the 1st parameter field then click Execute. Again, the reusable code is generated and the API Response displays the info about the available seeders found.





. . .

Conclusion:

By querying the DCore Daemon Database API, you can gather a multitude of information. From account balances, asset information, network statistics and more. I hope that this tutorial will assist you in getting started on your own great project ideas. Thanks for reading and happy coding!