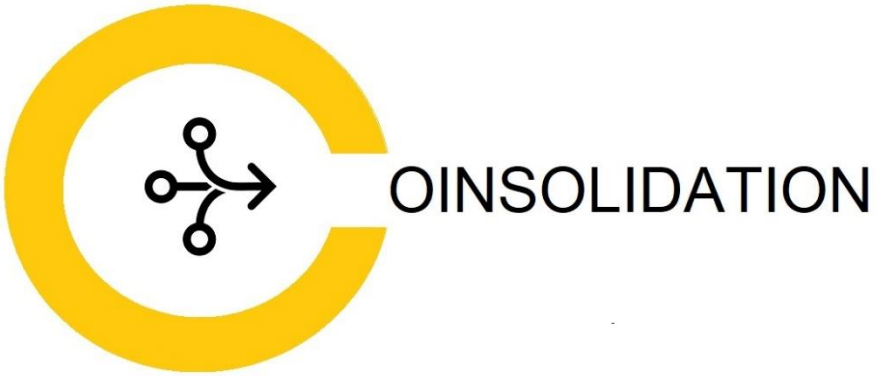


EXchange tensions

Configuration & Administration.

Ethereum Exchange Extension (EEE).

User's guide

version 1.0.0 Beta

November 2020.

Content

1. Introduction.

Ethereum is an open source platform, decentralized unlike other block chains, Ethereum can do much more. It is programmable, which means that developers can use it to create new types of applications. Its digital currency is called "Ether".

These decentralized applications (or "dapps") get the benefits of cryptomonics and blockchain technology. They are reliable and predictable, which means that once they are "loaded" into the Ethereum, they will always run on schedule. They can control digital assets to create new types of financial applications. They can be decentralized, which means that no one entity or person controls them.

Right now, thousands of developers around the world are creating applications in Ethereum and inventing new types of applications, many of which you can use today:

- Crypto-currency portfolios that allow you to make cheap, instant payments with ETH or other assets
- Financial applications that allow you to borrow, lend or invest your digital assets
- Decentralized markets, which allow you to exchange digital assets, or even exchange "predictions" about real-world events
- Games where you have assets in the game and can even win real money.

How does ether work?

Ether, like other crypto-currencies, uses a shared digital book where all transactions are recorded. It is publicly accessible, completely transparent and very difficult to modify afterwards.

This is called a **blockchain,** and is built through the **mining** process.

Miners are responsible for verifying groups of ether transactions to form "blocks" and coding them by solving complex algorithms. These algorithms can be more or less difficult as a way to maintain some constancy in the processing time of the blocks (about one every 14 seconds).

The new blocks are then linked to the previous block chain and the miner in question receives a **reward,** i.e. a fixed number of ether *tokens.* Normally this is 5 ether units, although this number can be seen to be variable if the crypt currency continues to rise.

How does Ethereum work?

The Ethereum *blockchain* is very similar to the bitcoin one, but its programming language allows developers to create software through which to manage transactions and automate certain results. This software is known as **intelligent contract**.

If a traditional contract describes the terms of a relationship, a smart contract makes sure those terms are met by writing them in code. These are programs that automatically execute the contract once the predefined conditions are met, eliminating the delay and cost that exists when executing an agreement manually.

To give a simple example, an Ethereum user could create a smart contract to send a set amount of ether to a friend on a certain date. They would write this code in the block string and when the contract is completed (i.e. when the agreed date is reached) the ether will be sent automatically.

This basic idea can be applied to more complex configurations, and its potential is probably unlimited, with projects that have already made remarkable progress in sectors such as insurance, real estate, financial services, legal services and micro patronage.

Smart contracts also have several additional benefits:

1. They eliminate the figure of the intermediary, offering the user total control and minimizing extra costs
2. They are registered, encrypted and duplicated in the public block chain, where all users can see the market activity
3. Eliminate the time and effort required in manual processes

Of course, smart contracts are still a very new system with many details to polish. The code is translated literally, so any errors during the creation of the contract could lead to unwanted results that cannot be changed.

DApps vs. smart contracts

Intelligent contracts share similarities with **DApps** (decentralized applications), but also separate them from some important differences.

Like smart contracts, a DApp is an interface that connects a user to a service from a provider through a decentralized peer network. But, while smart contracts need a fixed number of participants to be created, DApps have no limit to the number of users. Moreover, they are not just limited to financial applications such as smart contracts: a DApp can serve any purpose you can think of.

2. What is Blockly programming?

Blockly is a visual programming methodology based on the JavaScript programming language composed of a simple set of commands that we can combine as if they were the pieces of a puzzle. It is a very useful tool for those who want to learn to program in an intuitive and simple way or for those who already know how to program and want to see the potential of this type of programming.

Blockly is a form of programming where you don't need any background in any kind of computer language, this is because it is only joining graphic blocks as if we were playing lego or a puzzle, you just need to have some logic and that's it!

Anyone can create programs for cell phones (smartphones) without messing with those programming languages difficult to understand, just put together blocks in a graphical way in a simple, easy and fast to create.

3. What is an extension?

An extension is a module made in a given Java programming language in a file with the extension .AIX

In the Blockoin.org project we based on creating user-friendly extensions for the financial area with which they can get to make mobile applications in a few minutes without having a vast knowledge of programming.

4. What is BlockoinEthereum and BlockoinINFURA?

BlockoinEthereum and BlockoinINFURA are Software (extensions) of free use that includes the following technological solutions (algorithms) to be able to create being used in the Ethereum's network:

- CREATION OF NEW ACCOUNTS ON THE ETHEREUM PLATFORM.
- BALANCE SHEET CONSULTATION, ASSET TRANSFERS (CRYPTOMONEDA-ETHER AND TOKENS)
- CREATION OF NEW ERC20 AND CRYPTOMONEDA-TOKEN TOKENS.
- COMPILATION, CREATION, CONSULTATION AND PUBLICATION OF SMART CONTRACT
- CREATION OF OFFLINE AND ONLINE TRANSACTIONS.
- LOADING OF PRIMARY AND PUBLIC KEYS.
- EXCHANGE RATES & GAS STATION CONSULTATION.
- DEVELOPMENT, TESTING AND ETHEREUM CORE NETWORK ENVIRONMENTS (NETWORKS).
- INCLUDES THE 39 FUNCTIONALITIES OF INFURA.

5. Basic concepts applied in the Ethereum platform.

What is a blockchain?

The blockchain is generally associated with Bitcoin and other crypto-currencies, but these are just the tip of the iceberg since it is not only used for digital money, but can be used for any information that may have a value for users and/or companies. This technology, which has its origins in 1991, when Stuart Haber and W. Scott Stornetta described the first work on a chain of cryptographically secured blocks, was not noticed until 2008, when it became popular with the arrival of bitcoin. But currently its use is being demanded in other commercial applications and is projected to grow in the medium future in several markets, such as financial institutions or the Internet of Things IoT among other sectors.

The blockchain, better known by the term blockchain, is a single record, agreed upon and distributed over several nodes (electronic devices such as PCs, smartpohones, tablets, etc) in a network. In the case of crypto-currencies, we can think of it as the accounting book where each of the transactions is recorded.

Its operation can be complex to understand if we go deeper into the internal details of its implementation, but the basic idea is simple to follow.

It is stored in each block:

1.- a number of valid records or transactions,

2.- information concerning that block,

3 .- its link with the previous block and the next block through the hash of each block –un unique code that would be like the fingerprint of the block.

Therefore, **each block** has a **specific and unmovable place within the chain,** since each block contains information from the hash of the previous block. The entire string is stored on each node in the network that makes up the blockchain, so **an exact copy of the string is stored on all network participants**.

What is an address or account within the blockchain Ethereum platform?

It is a string of 42 characters in the Ethereum platform that represent a number in hexadecimal base, where the assets defined in the Ethereum will be deposited or sent. In other blockchain platforms the number of characters of the account or address can be different, for example:

0x5d2Acdb34c279Aa6d1e94a77F7b18aB938BFb2bB

What is a kryptomoney?

It is a digital or virtual currency designed to function as a medium of exchange. It uses cryptography (digital security) to secure and verify transactions, as well as to control the creation of new units of a particular crypto currency.

What is Ether?

Ether is the native digital currency of the Ethereum blockchain platform, a decentralized platform developed in 2013. An Ether is a unit that can be divided into smaller units called WEI and has the following equivalence.

1 Ether = 1,000,000,000,000,000,000 Wei. (In mathematical expression it is 10^{18}).

What units are handled when using an Ether?

1	ether
10^3	finney
10^6	szabo
10^9	Shannon o GWEI este se usa para el GasPrice.
10^{12}	babbage
10^{15}	lovelace
10^{18}	wei

What is a token?

Tokens are digital assets that can be used within the ecosystem of a given project.

The main distinction between tokens and crypto-currencies is that the former require another blockchain platform (not their own) to function. Ethereum is the most common platform for creating tokens, mainly due to its intelligent contract function. The tokens created on the Ethereum blockchain are generally known as ERC-20 tokens although there are other more specialized types of tokens such as the ERC-721 token used mainly for collectible assets (cards, use in video games, works of art, etc).

What is Gas?

Gas is the cost of performing an operation or set of operations on the Ethereum network. These operations can be several: from making a transaction to executing an intelligent contract or creating a decentralized application.

In other words, more simply, the Gas is the unit to measure the work done in the Ethereum.

As in the physical world, in Ethereum there are also jobs that cost more than others: if the operation we want to perform requires a greater use of resources by the nodes that form the platform, this will also increase the Gas and vice versa.

The Gas serves mainly to perform three functions on the Ethereum platform:

Assigns a cost to the execution of tasks; In Ethereum, performing tasks also has a cost. Depending on the difficulty of the task or the speed at which we want that task to be processed, the computational cost of that operation will be higher or lower accordingly and the number of Gas will increase or decrease in proportion.

2.- Secures the system; The Ethereum system is a safe system and this is largely possible thanks to the Gas.

By requiring a commission to be paid for each transaction performed, the platform ensures that no unusable transactions are processed on the network. This helps to make the blockchain lighter, as it will not add lots of useless Megabytes of information to the blockchain.

In addition, with the Gas, the system is also protected against 'spam' and the infinite use of loops: instructions to perform repetitive tasks by code.

For example, if the Gas did not exist, nothing would prevent a task from being repeated infinitely, collapsing the system and making it unusable.

3.- Rewarding the miners; The miners are independent external systems distributed around the world that are in charge of executing the transactions within the Ethereum platform. When we make a transaction or execute a smart contract, we "pay" a certain amount of Gas.

This gas serves to "pay" the miners for the resources they have used (hardware, electricity and time) and also add a reward for their work. Therefore, we could say that the Gas also helps to maintain the balance of the platform.

We talk about "paying" Gas -in quotes- because that is what operations cost. In other words, you "pay" for the expense that it costs Ethereum to process the transactions.

What is Gas Price?

A Gas unit corresponds to the execution of an instruction, such as a computer step.

So how do the miners "cash in" on the gas they acquire as a reward?

The gas itself is not worth anything, and therefore cannot be charged. In order to "charge" for this used gas, it is necessary to give value to these consumed resources, that is, to give a monetary value to the work done by the miners. The amount of Gas used in a transaction or a smart contract has an equivalent price in Ether. This price is called the 'Gas price', it is normally assigned by the miners and is variable depending on how busy the Ethereum blockchain is. It is normally handled in GWEI units.

What is Gas Limit?

This data indicates the maximum value of Gas that a transaction can consume to be valid.

Normally, the software used to make transactions on the Ethereum network automatically calculates an estimate of the amount of Gas needed to carry out the transaction and shows it to us immediately.

What is Gas Station?

It is the place where the values handled by the miners are referred to in order to decide in consensus which is the GasPrice needed to carry out the different types of transactions in the Ethereum blockchain.

Depending on how much of GasPrice is selected, the priority time given to the execution of the transaction will be weighted, normally three types of GasPrice are handled: TRADER: ASAP, FAST < 2 minutes, STANDARD < 5 minutes. These are average values and can vary the time depending on the workload (tractions) of the Ethereum's main network.

https://ethgasstation.info/

What is an Exchange?

A Kryptonian Exchange is the meeting point where exchanges of currencies are made in exchange for fiat money or other cryptosystems. In these online exchange houses the market price is generated which marks the value of the cryptomonies based on supply and demand.

What is Exchange Rates?

These are the rates of the value of an ether in the currencies of circulation of each country. For example, on the day of the creation of this manual, an ether has a value in US dollars of $430.94

What is a Smart Contract?

In ethereum a Smart contract is a program in programming language called Solidity that is within the blockchain Ethereum, which has instructions to be executed automatically based on pre-established rules, this property makes an evolution in all existing blockchain as you can create many Smart contract adapted to different private and public sectors making more efficient operations of companies or where appropriate adapt to systems or programs of all kinds.

What is "nonce"?

It is an incremental "hexadecimal number" counter that keeps track of transactions in each direction or account created in Ethereum.

What is a transaction?

It is the execution or transfer of some type of non tangible asset that can be given a pre-established value within the Ethereum system and that can later be changed to a tangible value for a company or person.

What is txHash?

It is a hexadecimal number that helps to track the result in detail of each transaction.

What types of transactions are there?

You have two types, one is the transaction "offline" this creates without the need to have connection to the main network of Ethereum can be stored until you choose to connect to the network of Ethereum and release the transaction, have the advantage of security because the entire transaction is processed offline which prevents any anomaly that could be in the network connection. The other transaction is the "online" one which always needs to be connected to the internet with the security advantages and disadvantages that the connection brings.

What is INFURA.io?

Infura.io is a platform that provides a set of tools and infrastructures that allow developers to easily take their blockchain application from testing, to scaling, with simple and reliable access to Ethereum and IPFS.

What is an address in the Ethereum network?

An address or account is composed of three parts, the address, the public key and the private key, these two keys are a string of numbers and characters in hexadecimal format that are used to send and receive (active) or ether (digital currency).

The primary key should never be shared with anyone as it is the one that authorizes the release of the balance (signs the transactions) held in the account.

The public key is known to the public and is shared with anyone as it is the reference to confirm that the transaction is correct both in terms of value and to whom it is sent.

Examples:

{

"private": "429a043ea6393b358d3542ff2aab9338b9c0ed928e35ec0aed630b93adb14a1c",

"public": "049b4b7e72701a09d3ee09165bba460f2549494a9d9fd7a95aaac57c2827eac162fd9e105b2461cd6594ca8ca6a8daf10fe982f918be1b0060c87db9cfbcd289a8",

"address": "88ab6dcecc3603c7042f4334fc06db8e8d7062d5"

}

What is Coinsolidation.org?

It is a platform in consolidating crypto-currencies, we created the first hybrid addresses in the centralized and decentralized financial world, using Blockly technology which is a form of visual programming without the need for advanced knowledge of programming based on functional extensions. See our WhitePaper at www.coinsolidation.org

What types of networks for testing and main network in the Ethereum blockchain?

https://mainnet.infura.io

Main network, production network where all transactions are made in real time.

https://ropsten.infura.io

Ropsten test network allows blockchain developments to test their work in a live environment, but without the need for real ETH tokens, with an algorithm called (*Proof-of-Work*).

https://kovan.infura.io

Kovan test network, which unlike its predecessor ropsten uses an algorithm called Proof of Authority.

https://rinkeby.infura.io

Test Network, uses an algorithm called Proof of Authority. One of the most widely used for testing.

https://goerli.infura.io

Goerli is a public test network for Ethereum that the POA (Proof of Authority) consensus engine supports with various clients. Spam-proof.

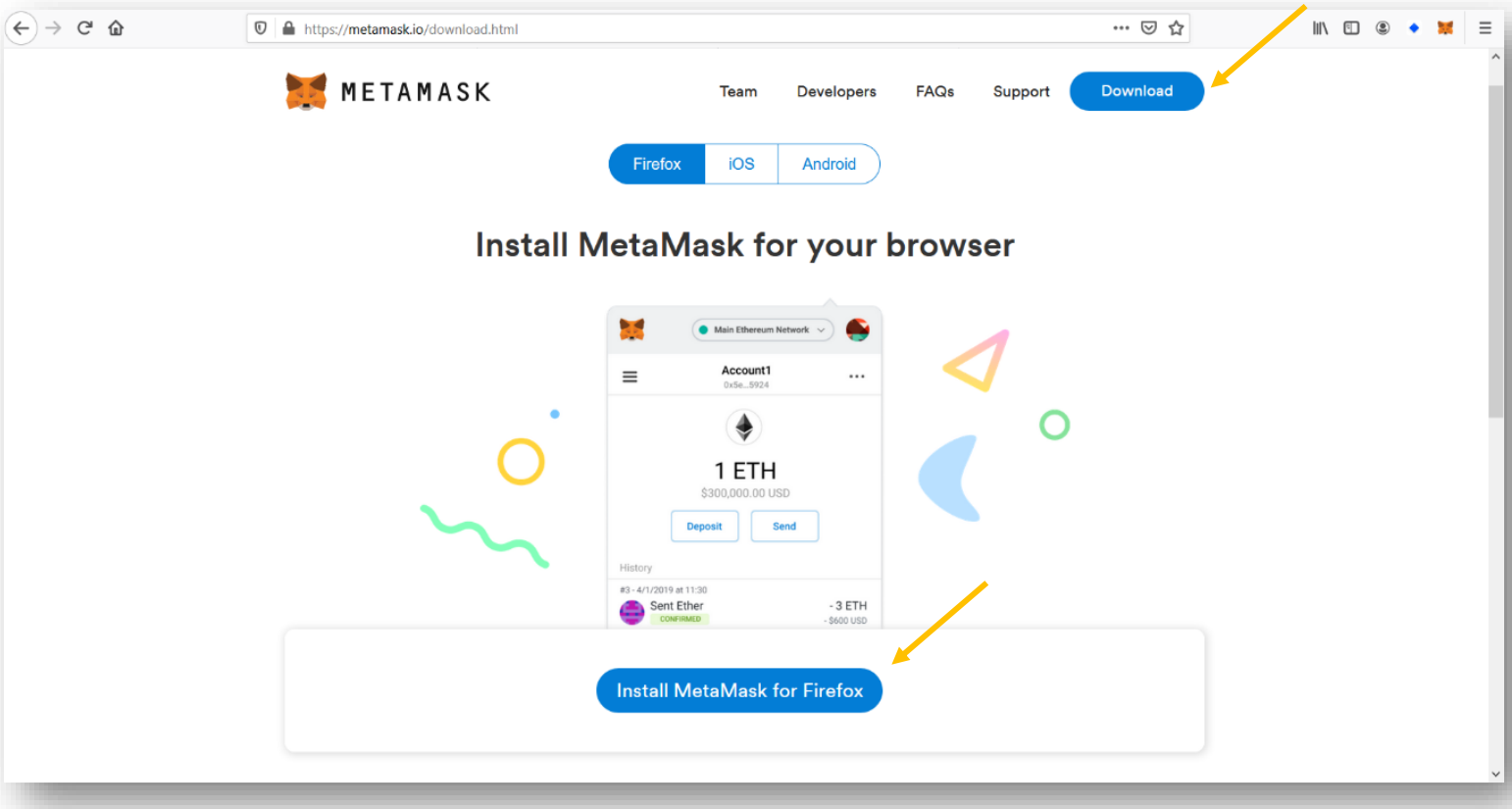
In total there are 5 networks based on the blockchain Ethereum, one production or main and four for testing, then we will use the network of Rinkeby to install our test environment.

6. Installation and configuration of extension and Ethereum testing environment.

We need a testing environment first. To learn how to use the different functionalities of the two extensions that will be used to create, send, publish, review and obtain data from all the transactions we make on the Ethereum platform.

The first thing we have to do is install our test environment. We will have to install the METAMASK application. This is a wallet to create, import Ethereum accounts and manage transactions from the browser in our internet browser that we will use is Mozilla and can also be used in Chrome.

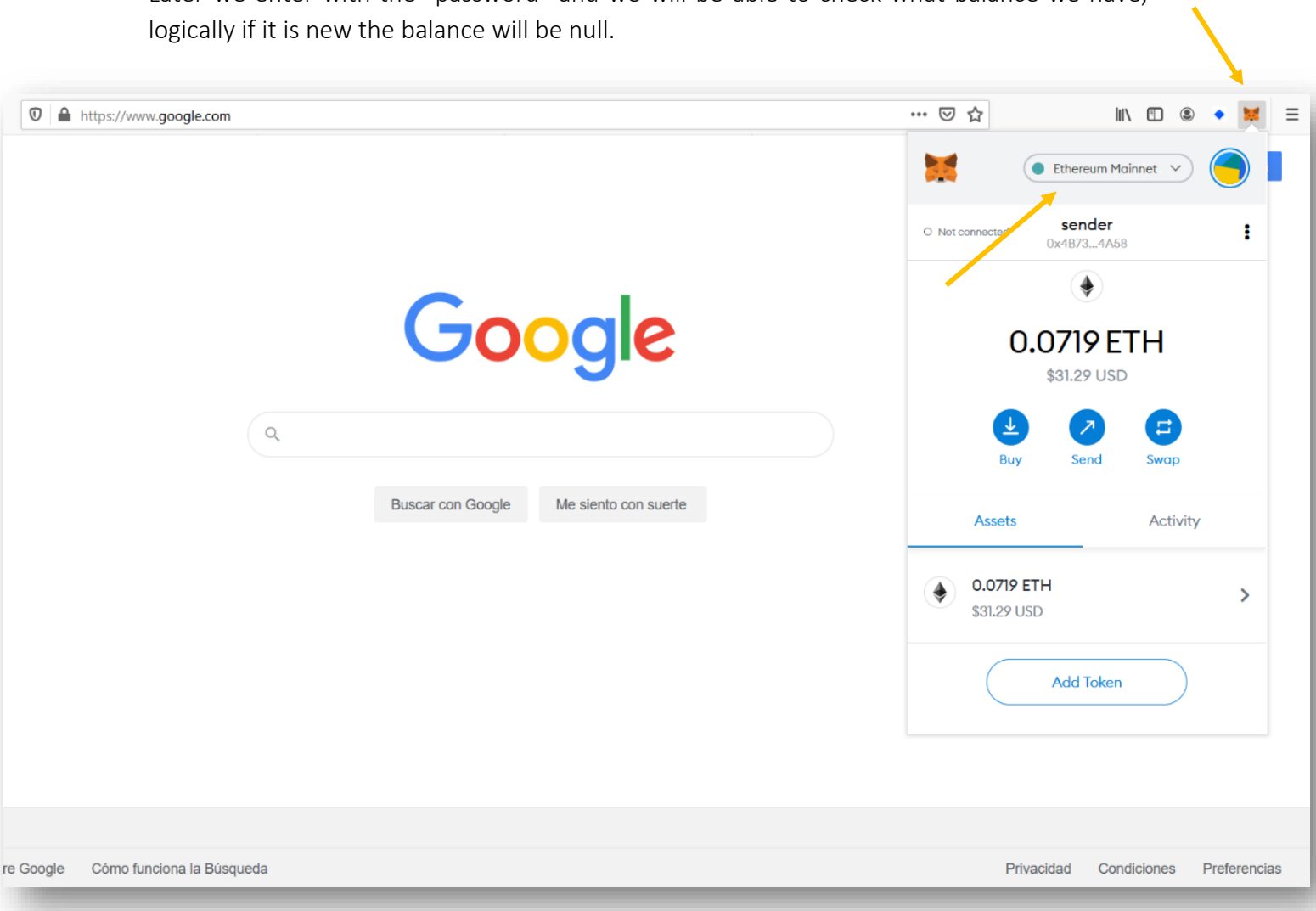
Go to the official website www.metamask.io and click on the "Down load" button.



Then click on the "Install MetaMask for Firefox" button and accept the default options. After the installation we will see an icon in the upper right side where we will see already installed the Metamask software.

Click on the Metamask icon and the option to import an existing account or create a new one will appear. In our case we will import an account that we already have running using the method of "restore through seed". If you do not have simply create a new account. In this case we will be asked for a "password" to be assigned in either case.

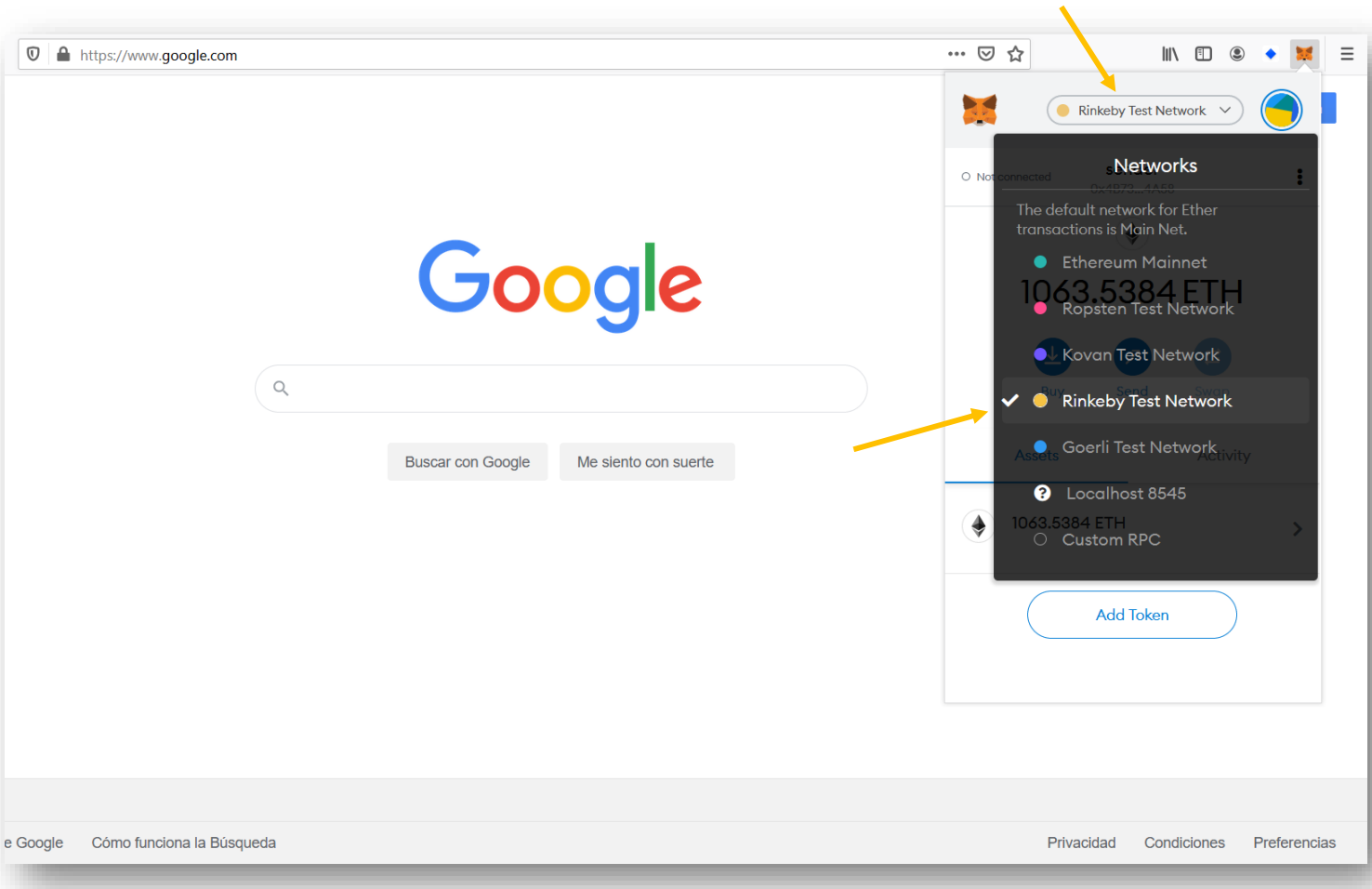
Later we enter with the "password" and we will be able to check what balance we have, logically if it is new the balance will be null.



Now we proceed to review how we are going to deposit some ethers (digital currencies) to our Rinkeby test account.

At the top we have the option to choose the type of network that we will use, by default when entering places you in the "Ethereum Mainnet", however, when you click we can

review all the optional networks that we can choose, in our case we selected the network of Rinkeby.



The next step is to deposit ethers to our Rinkery network referenced test account.

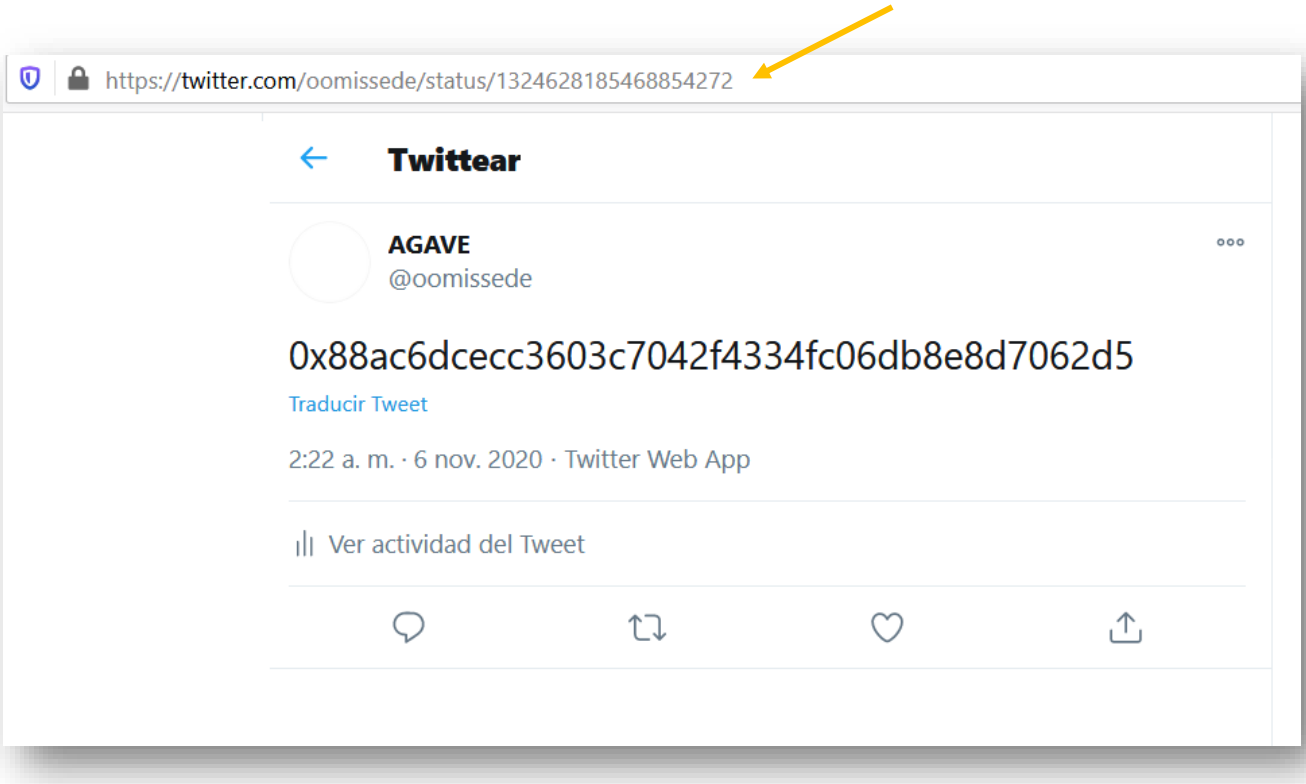
IMPORTANT NOTE: The ethers (digital currency) that we deposit in our account referenced to the Rinkery test network have no value in the crypto-currency market, they will only be used by us for software testing. Always check which network we are working on to avoid errors in transactions.

In order to get ether for testing we need to perform the following procedure.

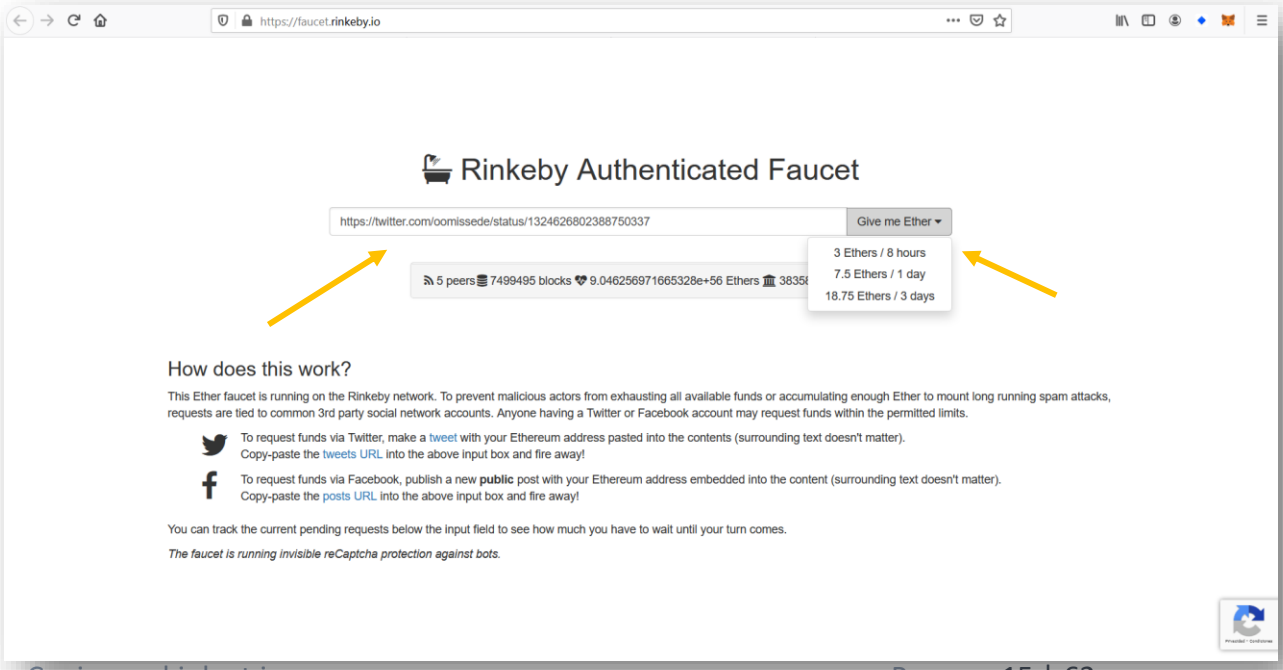
In any twitter account that you have we will have to enter twitter and create a comment that only includes the account and / or address we have in Metamask then we will have to copy the link of the comment since we have this we will go to the next link to fund our account.

https://faucet.rinkeby.io/

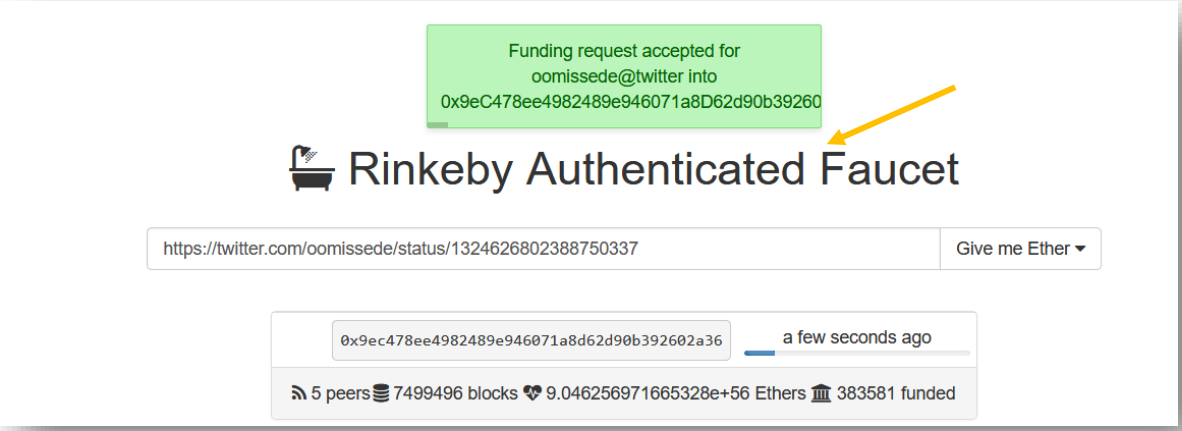
We created a twitter commentary and copied the link from the commentary.



In the site https://faucet.rinkeby.io/ we copy the twitter link and in the button "Give me Ether" we choose the desired amount.



Finally if everything is correct it will give us an announcement that the deposit was accepted and depending on the workload of the Rinkeby network system the deposit will be in a few minutes or may take longer.



Now that we have ethers in our account we can start testing on the Ethereum platform.

We have two extensions to interact with the Ethereum blockchain.

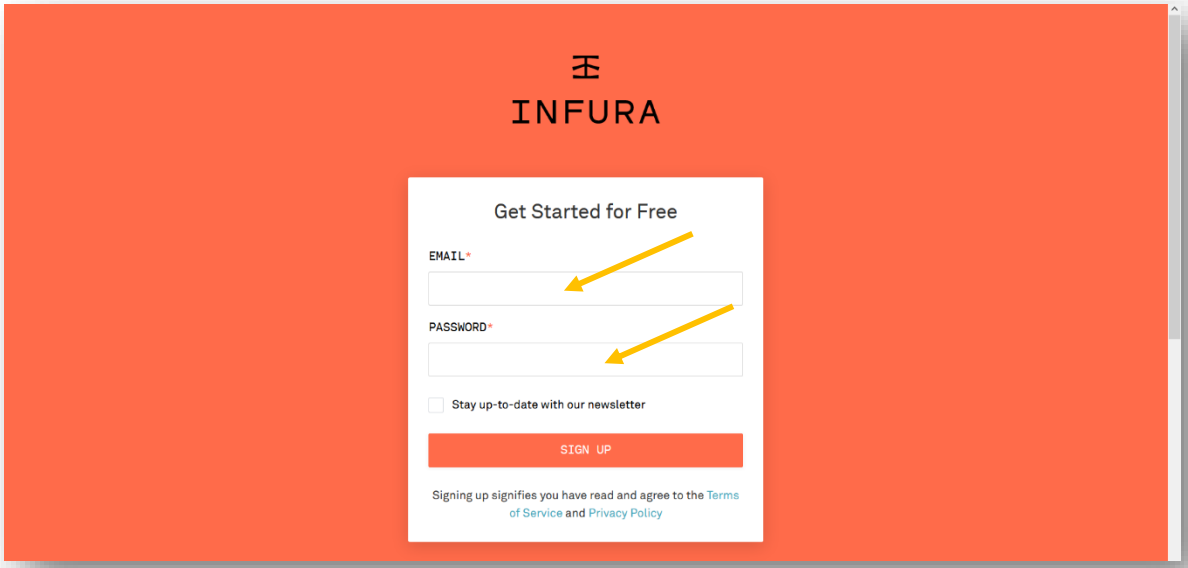
The co-solidation extension. BlockoinETHEREUM.aix contains the functions to perform the following functions

- Creation of new accounts (addresses) in blockchain Ethereum (privateKey, publicKey, Address)
- Storage of account data (addresses) in binary files.
- Import accounts (addresses) from binary files.
- Obtaining an account (address) through privateKey.
- Creation and sending of transactions between accounts (addresses) "online".
- Creation, signature and sending of offline PushRaw transactions.
- Consultation of transaction details Tx.
- Balance check on addresses and smart contracts.
- Compiler for Smart contracts.
- Creation, publication and execution of smart contracts.
- Creation, publication and execution of Token ERC20 (cryptocurrency token).
- Obtaining ABI code from smart contract.
- Verification of network connection.
- Consultation of ether's value in the crypt-currency market in any country of the world (native currency of the country) - Exchange Rates.
- GasPrice consultation.
- Consultation of "nonce" of a specific account.

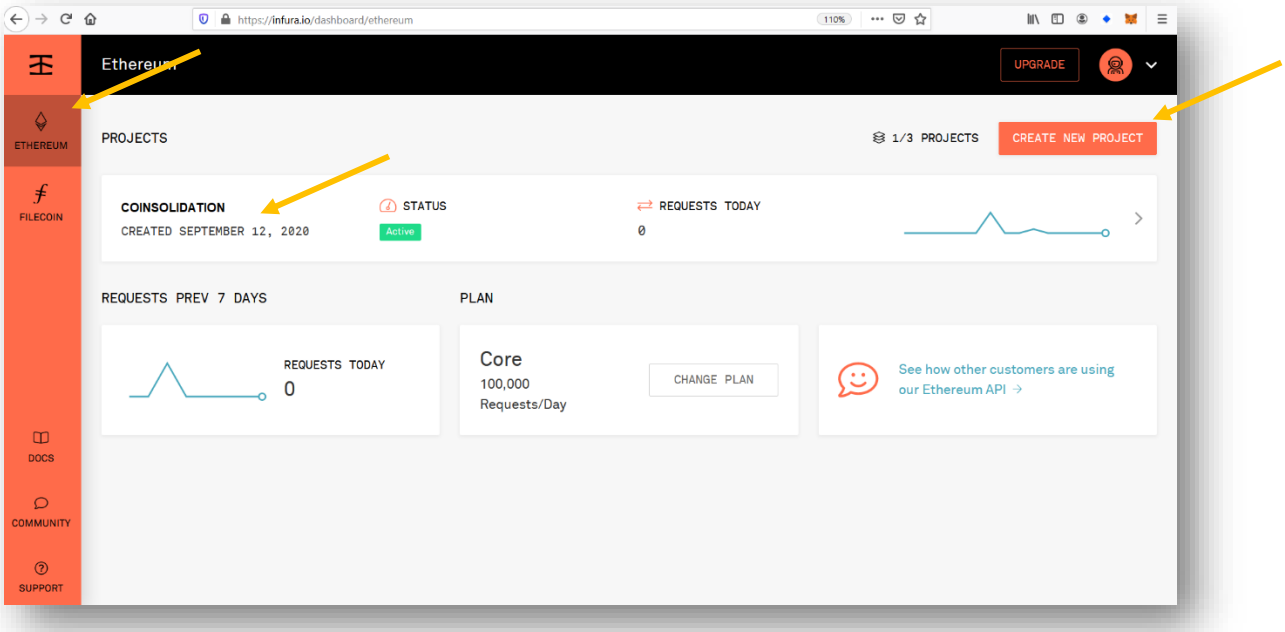
The **coinsoidation** extension. **BlockoinINFURA.aix** provides us with the 40 functionalities of the Infura.io platform. To see details, please consult the json-rpc documentation directly at https://infura.io/docs

In order to use the INFURA extension you need to create an account on the infura.io site since we need a KEY API to be able to send queries to the Ethereum network, as well as to be able to use the Ethereum testing networks.

How we open an account is simple as shown below.

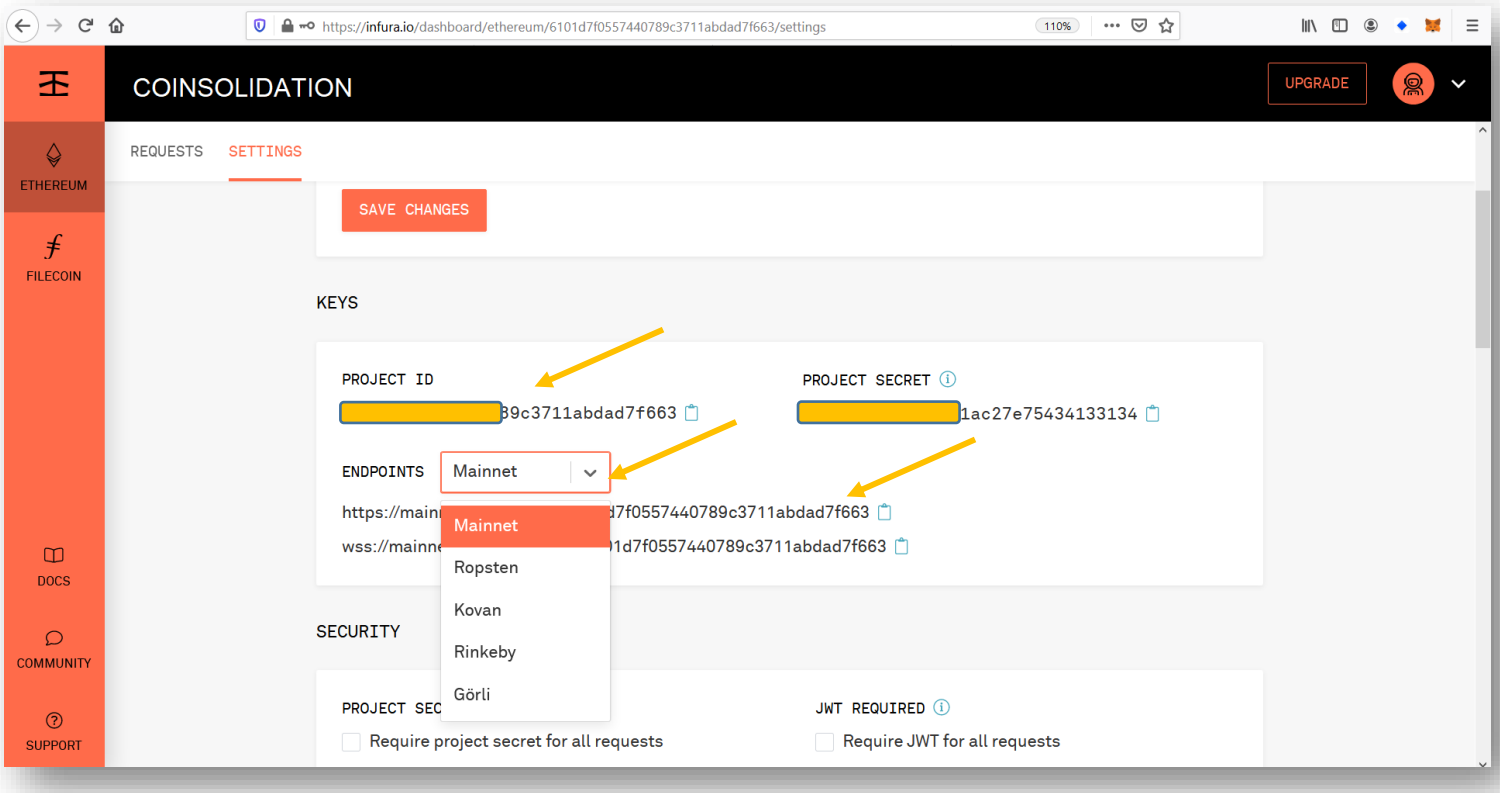


Once the account is created, we enter and we can have the KEY API of the different networks. We go to the top left and we click on Ethereum, then we will see the projects, we **create** a new project and we introduce ourselves to that project.

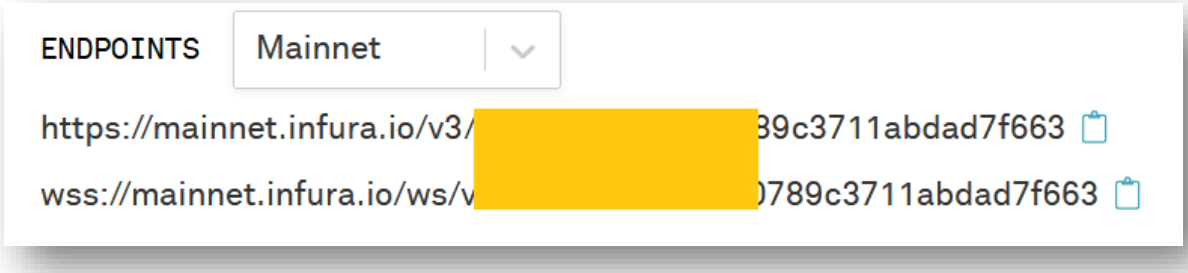


Within the project we will have the data of the KEY API and the different networks that we can use.

Within the project we can review the API KEY (PROJECT ID) and select which network we will work on or the complete links of the ENDPOINTS to choose.



For example, to use the Ethereum's main network we would take the following league:

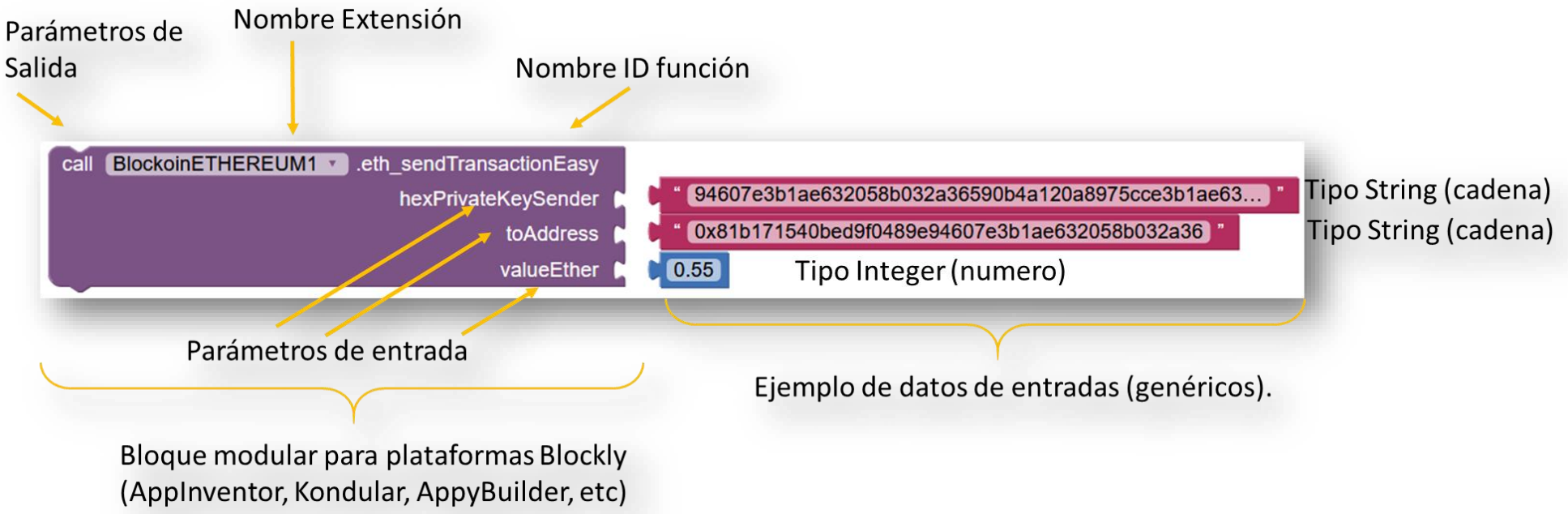


NOTE: The extensions have been tested on AppInventor, Kondular, Thunkable and AppyBuilder systems.

7. Definition and use of blocks (generic function)

We will start by explaining the distribution of the data that all the blocks will have, their syntax of use and configuration.

In the following example we can see a modular block and its input and output parameters, as well as the types of input data, these data can be of type String (string of characters) or Integer (integer or decimal). We show how it is used and configure it for its proper functioning.



Each module block will have its description and will be named in case it has any mandatory or optional dependency(s) of other blocks used as input parameters, the integration process will be announced.

8. Features and events of Exchange Ethereum Extension (EEE).

***Testing network that we will use **Rinkeby,** as urlNetwork, when you want to make real transactions you only have to change the urlNetwork network for **mainnet.**

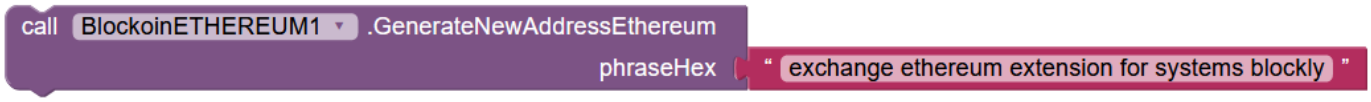
Block to check internet connection - (**CheckInternetConnection**).

Input parameters: Not applicable.

Output parameters: Returns "True" if you have connection or returns "False" if there is no connection.

Description: Block to check internet connection and send data (transactions).

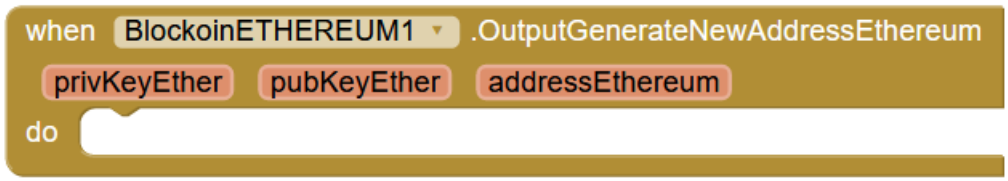
Block to generate a **new** "Offline" address - (**GenerateNewAddressEthereum**).



Input parameters: **phrasaHex <String>**.

Output parameters: Event (**OutputGenerateNewAddressEthereum**).

Outputs: PrivateKey<String>, PublicKey<String> , **addressEtehreum<String>.**



Description: Create a new ethereum address (account) based on a phrase or sequence of numbers. The new address can be created without a network or internet connection - "Offline".

Block to generate a new "Online" address - (**GenerateNewAddressEthereum**).

Input parameters: Not applicable.

Output parameters: Return in JSON data format; privateKey, publicKey, address.

Output example:

{

"**private**": "9ab4b2fba728a55643414f26adc04eea080740860cbe39b13fe4acb43dbb9f83",

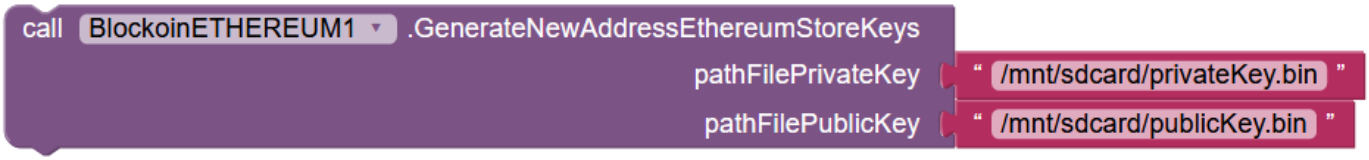
"**public**": "0421d559aa98d6fc77688ae9f19b3dc502c05f0b7f70bbe924cb712d6243a489695b1c1ee03993b3b6d9727797e780677a5469800b4d98374bdb910ed99fa2b5c8",

"**address**": "92a2f157d5aec3fa79f92995fea148616d82c5ef"

}

Description: Create a new ethereum address (account). It is necessary to have access or connection to the internet since the generation is through REST API service - "Online".

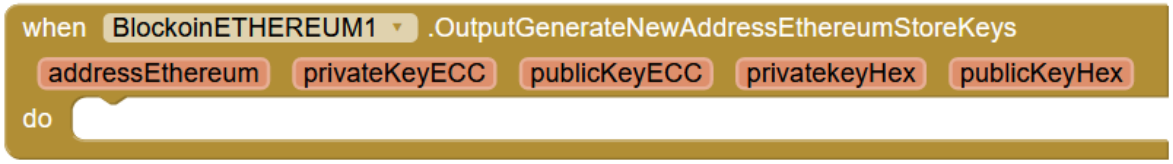
Block to generate a new "Offline" address and save the public and private keys in binary files - (**GenerateNewAddressEthereumStoreKeys**)



Input parameters: **pathFilePrivateKey<String>** , pathFilePublicKey<String>.

Output parameters: Event (**OutputGenerateNewAddressEthereumStoreKeys**).

Outputs: addressEthereum<String> , privateKeyECC<String> , **publicKeyECC<String> , privateKeyHex<String> ,** publicKeyHex<String>.



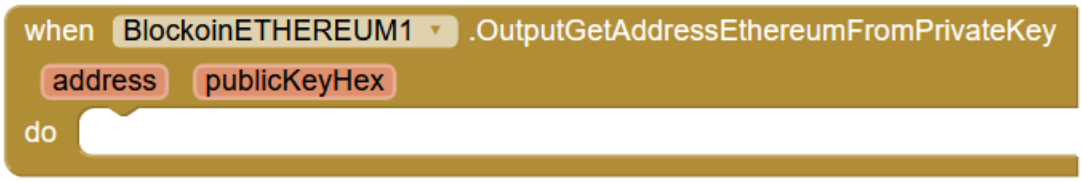
Description: Creates a new random ethereum address (account) and stores the public and private keys in binary files that will be used for importing and exporting account data. The new address can be created without a network or internet connection - "Offline".

Block to generate public key - (**GeneratePublicKeyHexFromPrivateKeyHex**).

Input parameters: **hexPrivateKey <String>**.

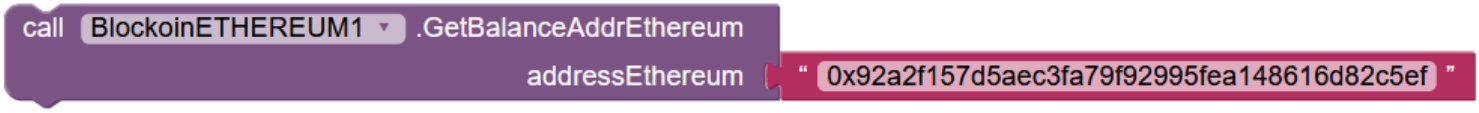
Output parameters: Event (**OutputGeneratePublicKeyHexFromPrivateKeyHex**).

Outputs: address<String> , publicKeyHex<String>.



Description: Creates the public key based on the entry of a private key.

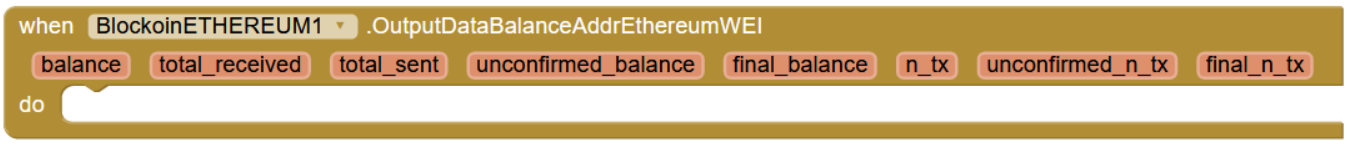
Block to obtain the balance of an address - (**GetBalanceAddrEthereum**).



Input parameters: **hexPrivateKey <String>**.

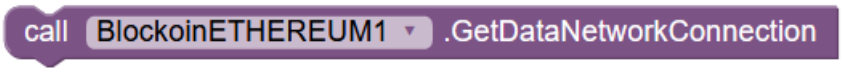
Output parameters: Event (**OutputGeneratePublicKeyHexFromPrivateKeyHex**).

Outputs: balance<String> , total_received<String> , **total_sent<String>** , **unconfirmed_balance <String>** , final_balance<String> , **n_tx<String>** , unconfirmed_n_tx<String> , **final_n_tx<String>.**



Description: Displays balance and detailed account (address) data.

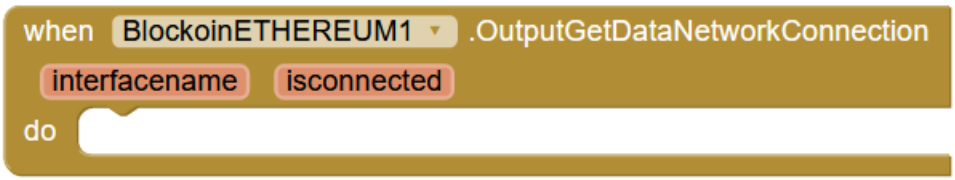
Block to check if your mobile has activated the network interface - (**GetDataNetworkConnection**).



Input parameters: Not applicable.

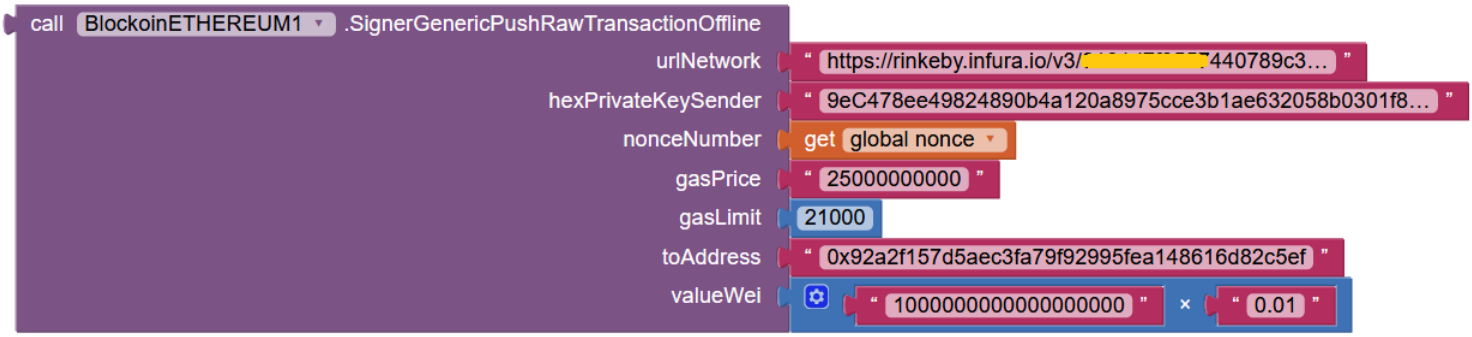
Output parameters: Event (**OutputGeneratePublicKeyHexFromPrivateKeyHex**).

Outputs: interfasename<String> , isconnected<String>.



Description: Displays the name of the mobile interface and delivers whether the interface is activated or not.

Block to sign transaction "Offline" - (**SignerGenericPushRawTransactionOffline**)



Required dependency(s): Block (**eth_getTransactionCount)**, Block (**eth_SendRawTransactionInfura**).

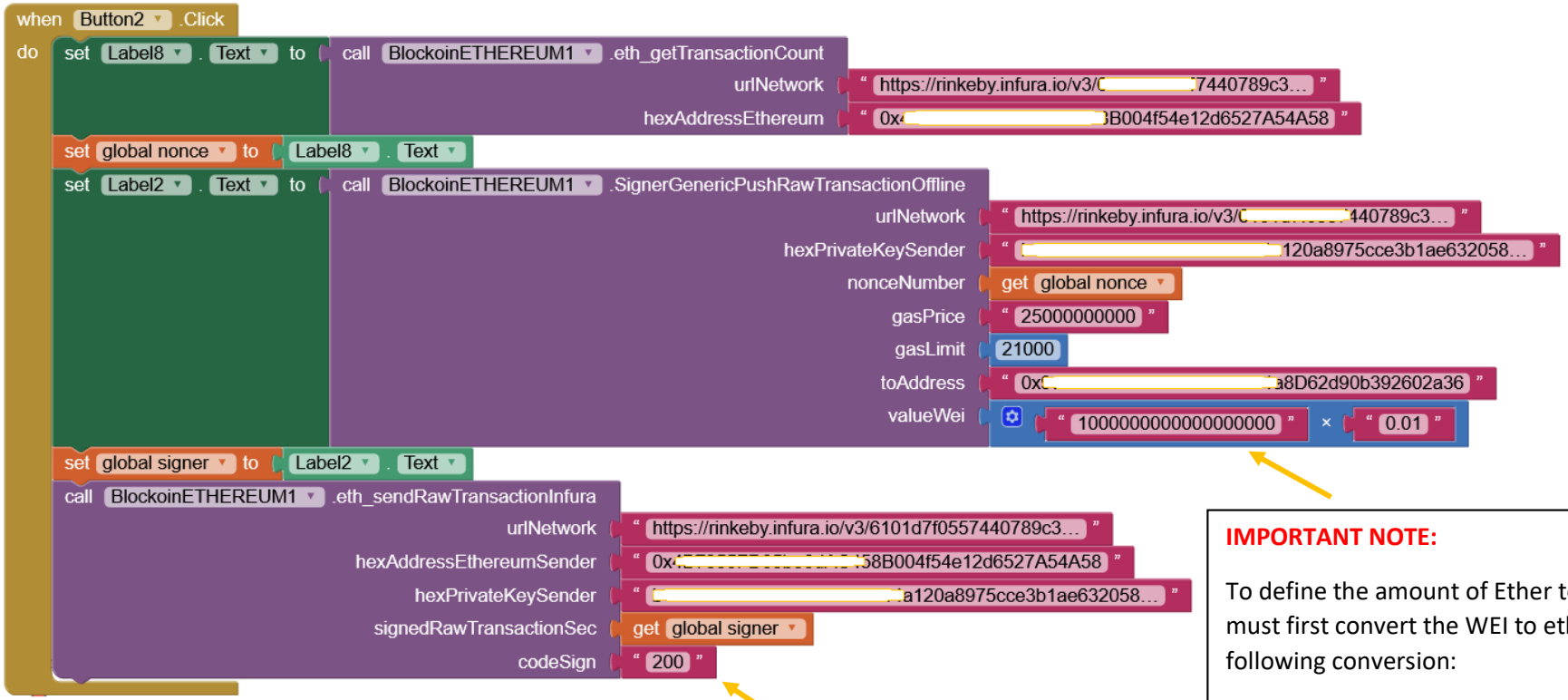
Parámetros de entrada: **urlNetwork** <String>, **hexPrivateKeySender** <String>, nonceNumber **<String>,** gasPrice <String**>, gasLimit** <Integer>, **toAddress** <String>, valueWEI **<Integer>.**

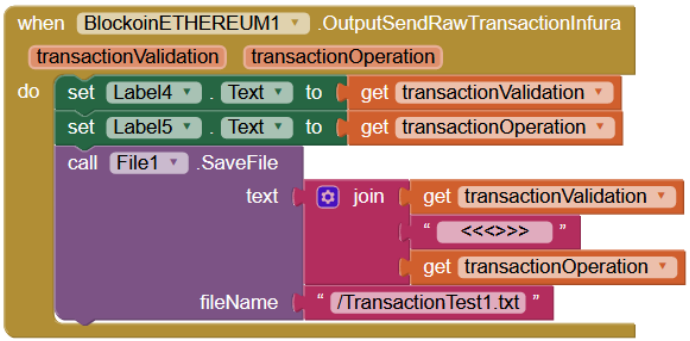
Output parameters:

Outputs: Signed transaction to be sent. <String>.

Description: Prepares a new transaction to be sent (encrypted and signed). This can be processed without a network or internet connection - "Offline".

Example of full use with block dependencies **(SignerGenericPushRawTransactionOffline).**





IMPORTANT NOTE:

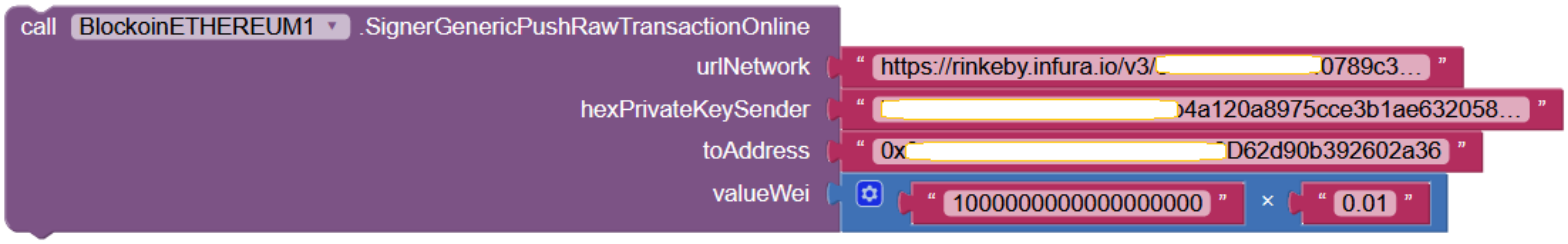
To define the amount of Ether to send, we must first convert the WEI to ethers. Use the following conversion:

1 Ether = 10000000000000000 WEI = ^{1018}

(10000000000000000) x Num. ether

In this function "Offline" the codeSign must be at 200 or empty, never put 400 or the transaction will not be made.

Block to sign "Online" transaction - (**SignerGenericPushRawTransactionOnline**).

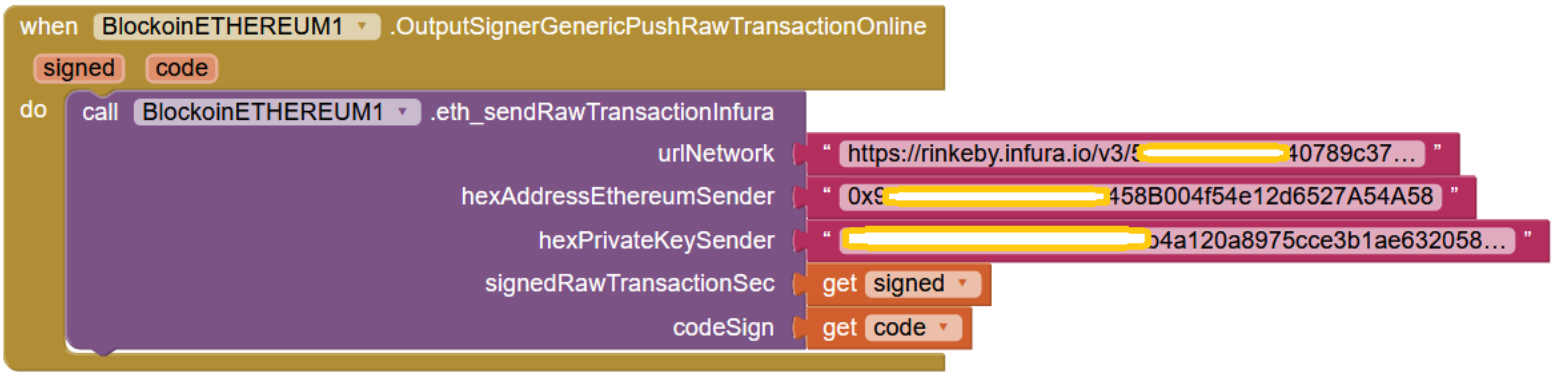


Mandatory dependency(s): Block (**eth_SendRawTransactionInfura**).

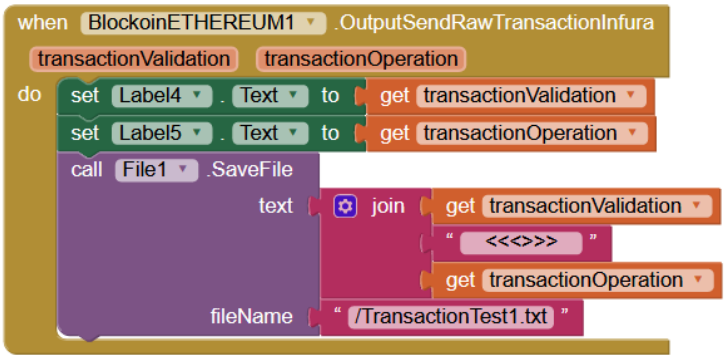
Input parameters: **urlNetwork <String>**, hexPrivateKeySender **<String>,** toAddress <String**>,** valueWEI <Integer**>.**

Output parameters: Events to use in the following order (**OutputSignerGenericPushRawTransactionOnline**) and (**OutputSendRawTransactionInfura**).

Outputs: signed<String> , code<String>.

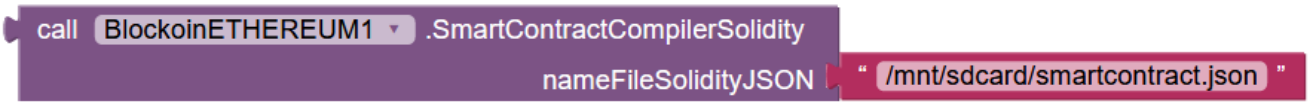


Outputs block eth_sendRawTransactionInfura: **transactionValidation<String>** , **transactionOperation<String**>.



Description: Prepares a new transaction to be sent (encrypted and signed). A network or internet connection is required - "Online".

Block to compile Smart contract "Online" - (**SmartContractCompilerSolidity**).



Input parameters: **nameFileSolidityJSON <String>**.

Output parameters: Shows the compiled smart contract, this function is useful to check if it is well written before publishing a smart contract in the Ethereum network.

Outputs: **Compiled code.**

The Smart contract must be in a JSON format file.

Example of basic Smart contract in Solidity language.

```
pragma solidity ^0.5.0;

deadly contract {
address owner;
function mortal() { owner = msg.sender; }
function kill() { if (msg.sender == owner) suicide(owner); }}
contract greeter is deadly {
string greeting;
function greeter(string _greeting) public { greeting = _greeting; }
function greet() constant returns (string) {return greeting;}
```

Example of previous Smart contract in JSON format with comments.

```
# Check solidity compilation via non-published test
# Using "greeter" contract solidity example, the "hello world" of
Ethereum.
```

File: smartcontract.json

```
{
 "solidity": "contract mortal {\n /* Define variable owner of the type
address*/\n     address owner;\n     /* this function is executed at
initialization and sets the owner of the contract */\n     function
mortal() { owner = msg.sender; }\n     /* Function to recover the funds
on the contract */\n     function kill() { if (msg.sender == owner)
suicide(owner); }\n}\n\ncontract greeter is mortal {\n     /* define
variable greeting of the type string */\n     string greeting;\n     /*
this runs when the contract is executed */\n     function greeter(string
_greeting) public {\n         greeting = _greeting;\n     }\n     /* main
```

```
function */\n     function greet() constant returns (string) {\n
return greeting;\n     }\n}",
"params: ["Hello Coinsolidation Test"]
}
```

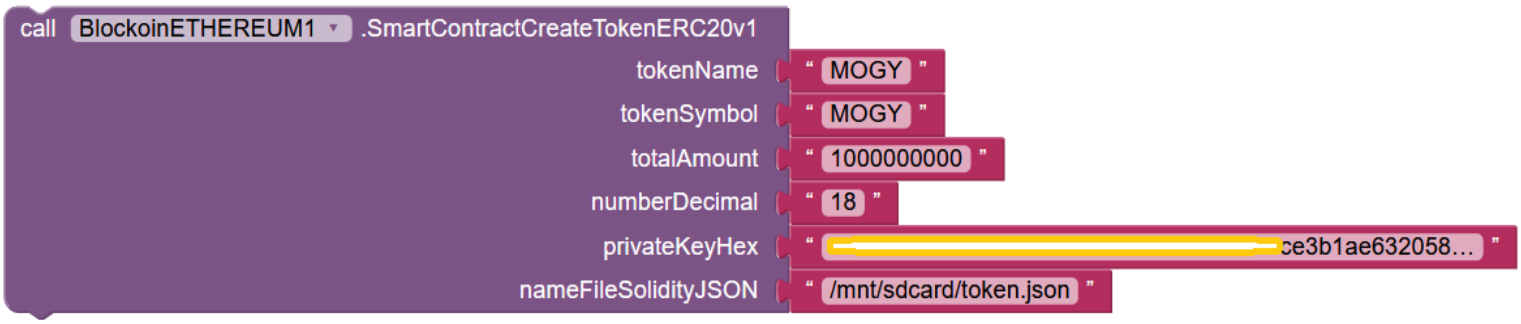
IMPORTANT NOTE: The JSON format must always have a line break at the end of each line.

Example of compiled Smartcontract output.

```
[
 {
     "name": "\u003cstdin\u003e:greeter",
     "solidity": "contract mortal {\n /* Define variable owner of the
type address*/\n     address owner;\n     /* this function is executed at
initialization and sets the owner of the contract */\n     function
mortal() { owner = msg.sender; }\n     /* Function to recover the funds
on the contract */\n     function kill() { if (msg.sender == owner)
suicide(owner); }\n}\n\ncontract greeter is mortal {\n     /* define
variable greeting of the type string */\n     string greeting;\n     /*
this runs when the contract is executed */\n     function greeter(string
_greeting) public {\n         greeting = _greeting;\n     }\n     /* main
function */\n     function greet() constant returns (string) {\n
return greeting;\n     }\n}",
     "bin":
"606060405260405161023e38038061023e8339810160405280510160008054600160a060
020a0319163317905580600160005090805190602001908280546001816001161561010000
203166002900490600052602060002090601f016020900481019282601f10609f57805160
ff19168380011785555b50608e9291505b8082111560cc5760008155830160 7d565b50505
061016e806100d06000396000f35b828001600101855582156076579182015b8281111560
765782518260005055916020019190600101906 0b0565b5090566060405260e060020a6
00035046341c0e1b5811461002657806 3cfae321714610068575b005b6100246000543373
ffffffffffffffffffffffffffffffffffffffff90811691161415610137576000 5473fff
ffffffffffffffffffffffffffffffffffffff16ff5b6100c960006060908152600180546 0
a06020601f60026000196101008688161502019094169390930492830181900402810160 4
052608082815292919082828015610164578060 1f10610139576101008083540402835291
602001916101645656 5b604051808060200182810382528381815181526020019150805190 6
020019080838382906000600460208460 1f0104600f02600301f150905090810190601f16
801561012957808203805160018360200361010 00a031916815260200191505b509250505
06
     "abi": [
       {
         "constant": false,
         "inputs": [],
         "name": "kill",
         "outputs": [],
         "type": "function"
       },
       {
         "constant": true,
         "inputs": [],
         "name": "greet",
         "outputs": [
           {
             "name": "",
```

```
            "type": "string"
          }
        ],
        "type": "function"
      },
      {
        "inputs: [
          {
            "name": "_greeting",
            "type": "string"
          }
        ],
        "type": "builder"
      }
    ],
    "params": [
      "Hello Coinsolidation Test"
    ]
 },
 {
    "name": "mortal",
    "solidity": "contract mortal {\n /* Define variable owner of the
type address*/\n     address owner;\n     /* this function is executed at
initialization and sets the owner of the contract */\n     function
mortal() { owner = msg.sender; }\n     /* Function to recover the funds
on the contract */\n     function kill() { if (msg.sender == owner)
suicide(owner); }\n}\n\ncontract greeter is mortal {\n     /* define
variable greeting of the type string */\n     string greeting;\n     /*
this runs when the contract is executed */\n     function greeter(string
_greeting) public {\n         greeting = _greeting;\n     }\n     /* main
function */\n     function greet() constant returns (string) {\n
return greeting;\n     }\n}",
    "bin":
"606060405260008054600160a060020a0319163179055605c8060226000396000f36060
60405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffffffff
ffffffffffffffffffffffffffffffff9081169116141560 5a5760005473ffffffffffffff
ffffffffffffffffffffffff16ff5b56",
    "abi": [
      {
        "constant": false,
        "inputs": [],
        "name": "kill",
        "outputs": [],
        "type": "function"
      },
      {
        "inputs": [],
        "type": "builder"
      }
    ],
    "params": [
      "Hello Coinsolidation Test"
]
 }
]
```

Block to compile, create and publish ERC20 Token - (SmartContractCreateTokenERC20v1)

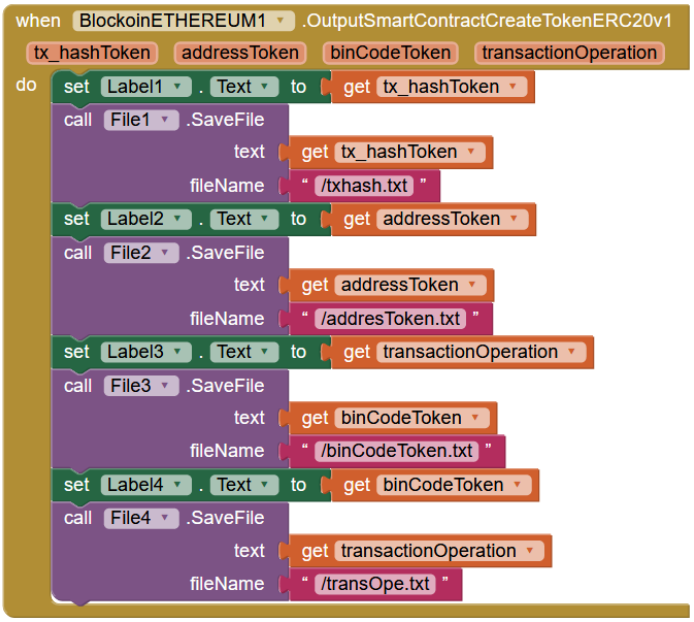


Mandatory dependency(s): Block (CreateTestingFie). IMPORTANT: First you must use this block to ensure that the path is correct, this is because if no valid path is given in the block (SmartContractCreateTokenERC20v1) will not run the creation of token because the input parameter "nameFileSolidityJSON" is used to create a temporary file.

Input parameters: tokenName <String>, tokenSymbol <String>, totalAmount <String>, numberDecimal <String>, privateKeyHex <Integer>, nameFileSolidityJSON <String> This file is the valid path to create a temporary file, you must ensure that the path is valid to test that the file is created you can use the Block (CreateTestingFile) after using it check that it has been created successfully.

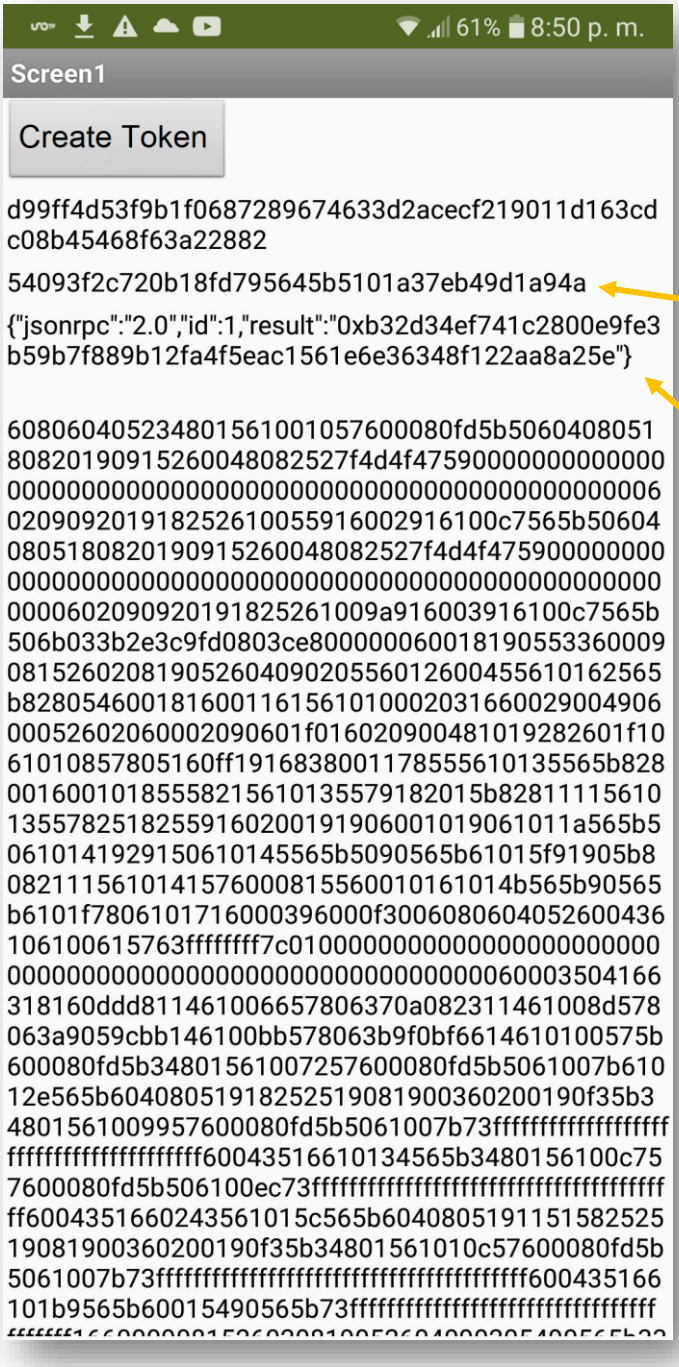
Output parameters: Event (OutputSmartContractTokenERC20v1).

Outputs: tx_hashToken<String> , addressToken<String> , binCodeToken<String> , trasactionOperation<String>.



Description: Smart contract "Token ERC20" - active published on the Ethereum network. Version 1 (v1) already has the Limit gas parameter set to 500,000 WEI.

Example of output of the previous function SmartContractCreateTokenERC20v1.



Transaction (Tx_hash) of the Ethereum network creation. This can be consulted at: www.etherscan.io

Address of the new contract that refers to the new ERC20 token created.

Transaction (Tx_hash) of the validated operation in the CoinSolidation.org system

This can be found at: www.etherscan.io

Binary code (BIN) of the new token contract that was processed in the EVM (Ethereum Virtual Machine).

9. Steps to create a CryptoToken or Cryptomoney Token.

Step 1.

Verify that the temporary path on the mobile device where the Smart contract was created is valid and a file can be successfully created. This is done using the Block (**CreateTestingFile**). Verify that the test file given in the input "pathTestFile" has been created, generically the path is given by: /mnt/sdcard/name_file.txt

Step 2 (optional).

In this step we will verify if the account (address) from which the operation will be charged has enough balance to perform the transaction of creating and publishing a Smart contract. This can be verified using the Block (**eth_VerifiBalanceForTransaccionSmartContract**). This use of this block is optional because the blocks that generate the **SmartContractCreateTokenERC20v1** or **SmartContractCreateTokenERC20v2** already contain this verification internally.

Step 3.

Select which Block to create ERC20 token will be used, we have two options:

a.- Block **SmartContractCreateTokenERC20v1** already has the implicit value of Gas Limit assigned with a value of 500,000 WEi.

b.- Block **SmartContractCreateTokenERC20v2** has the option of being able to configure the Gas Limit to the need of the end user or developer. It should be noted that if a very low Gas Limit of less than 350,000 Wei occurs, it is very possible that the Smart contr.

Step 4.

Use any of the **SmartContractCreateTokenERC20v1** or **SmartContractCreateTokenERC20v2** blocks making sure that when using the input variable "nameFileSolidityJSON" it is equal to the variable in the block's "pathTestFile" input (**CreateTestingFile**) already checked in step 1.

Step 5.

Before executing the creation of an ERC20 token with any of the **SmartContractCreateTokenERC20v1** or **SmartContractCreateTokenERC20v2** blocks, it is recommended to save the Event values (results) as appropriate to save the results (tx_hashToken, addressToken, binCodeToken, transactionOperation). See example of output of the previous function **OutPutSmartContractCreateTokenERC20v1.**

Step 6.

Execute the creation of the ERC20 token and later publish it for sale. See section 10.

10. How to put a new asset or your cryptoToken on sale (Token ERC20).

Since we have created an ERC20 - Cryptomoney Token (See SmartContractCreateTokenERC20v1 Block or with the SmartContractCreateTokenERC20v2 Block) we have to upload it to some Exchange so that anyone in the world can buy it. An Exchange is a site on the Internet where new tokens are published.

Exchanges are classified into two types, Centralized and Decentralized. The main difference is that one is governed and audited by some kind of international body (Centralized) and the decentralized ones have no one with the auditors. Although this could give more confidence, the reality is that lately the decentralized ones have taken more strength and are used mostly without major problems.

One of the best practices when handling assets of any kind is not to have all of them in one account but to distribute them in several accounts for the security of both private and public keys.

In our case we will use a Decentralized Exchange but already with a not bad history, we will use www.forkdelta.app

At this time we must already have installed the application for the browser (Mozilla or Chorme) METAMASK www.metamask.io this is because the Exchange to visit your page www.forkdelta.app need to connect to the account we have Ethereum.

An important point is that our Ethereum account that we already have in METAMASK must have a balance of more or less 10 USD this is because when we publish our new ERC20 token that we created with the SmartContractCreateTokenERC20v1 block or with the SmartContractCreateTokenERC20v2 block we will need to pay the transaction to publish it in the Exchange.

In order to use the Exchange www.forkdelta.app we must have the following token data that we want to publish to sell in the Exchange.

Address of the new ERC20 token we created earlier.

0x54093F2C720b18Fd795645b5101A37EB49d1A94a

Number of decimals used by our touch.

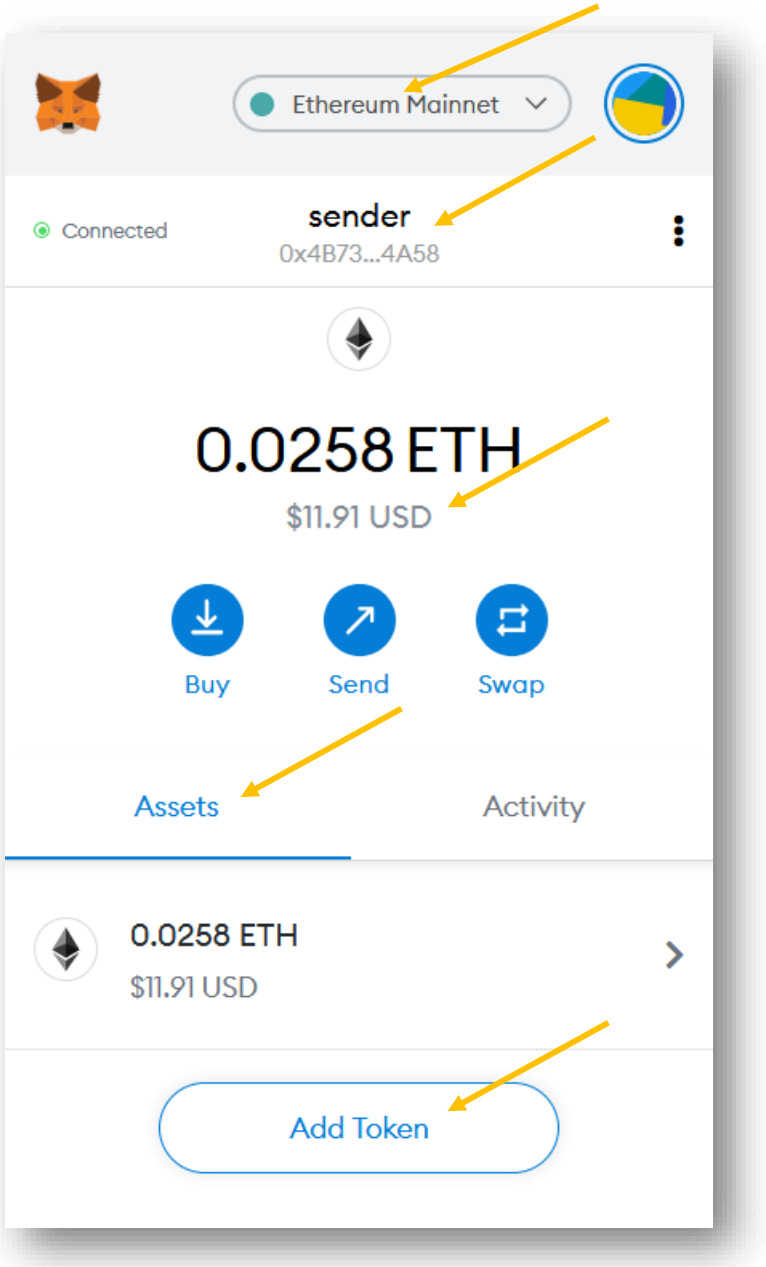
18

The name that identifies the token.

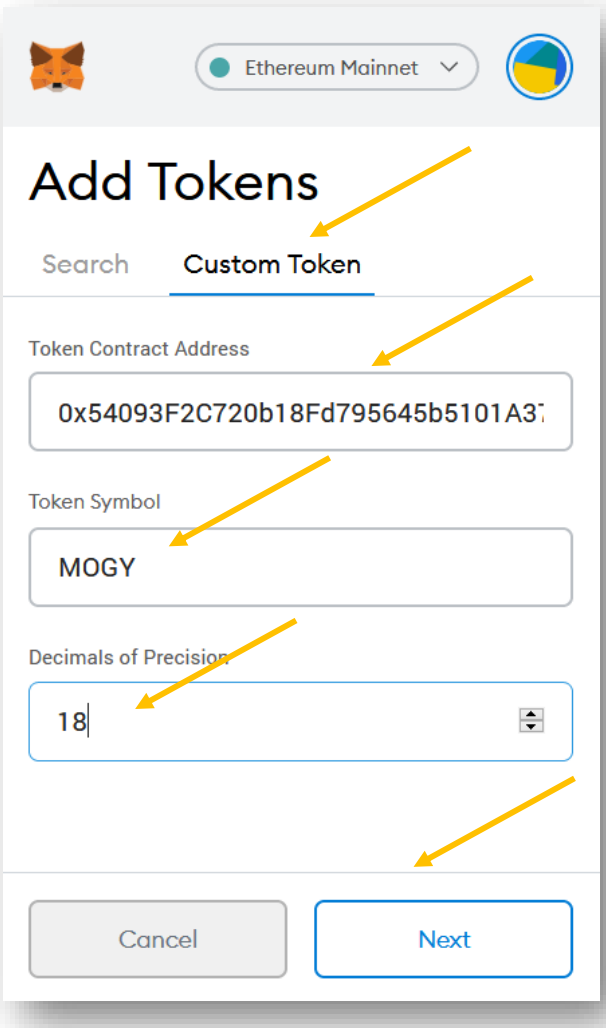
MOGY

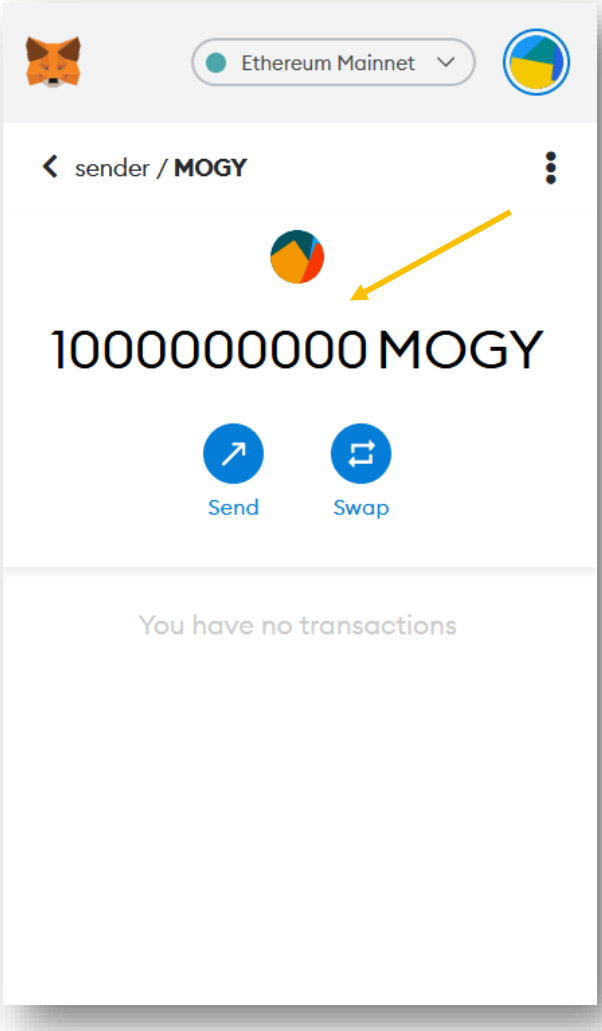
Let's start by checking that the token we created has the parameters mentioned above and that they were the ones it was initially created with.

We go to METAMASK and first we make sure we are in the account with which we created the token. Then we go to the bottom and click on the "Add Token" button.



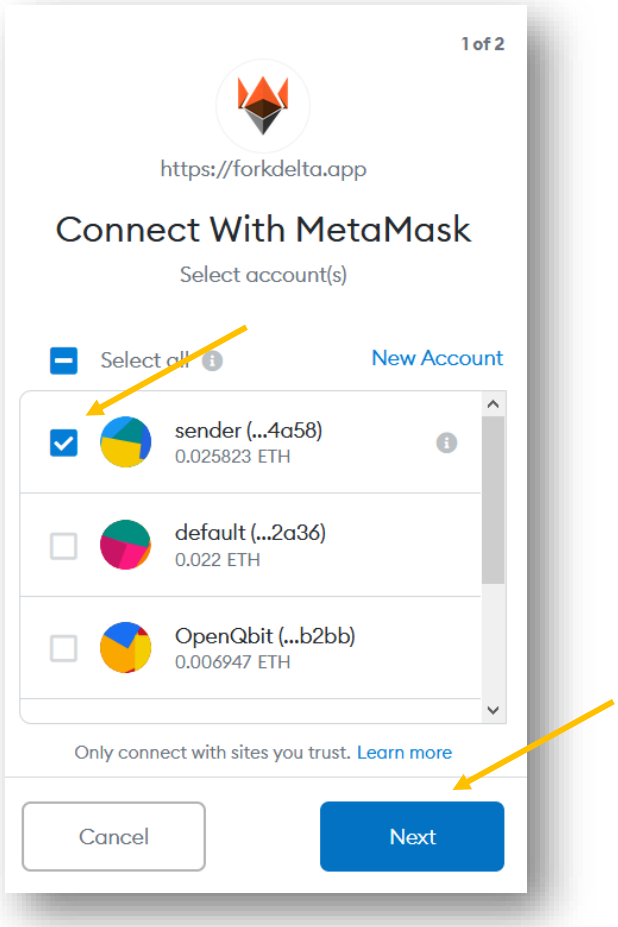
We will now add our new token to view your current balance. After clicking on the upper part of the token, choose the "Custom token" section and in the following fields write the information requested, then click on the "Next" button. After that, our new token appears with the balance it has. If you do not have a balance, check if you are in the account that you created the token with. If you are not in an account that you did not create, you will have a zero balance because you have not yet bought any token of this type.

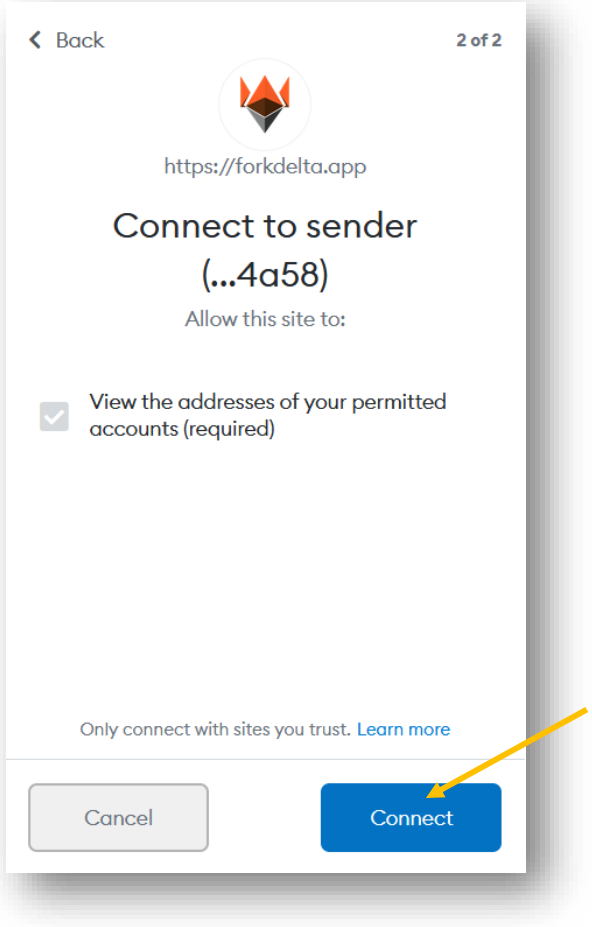


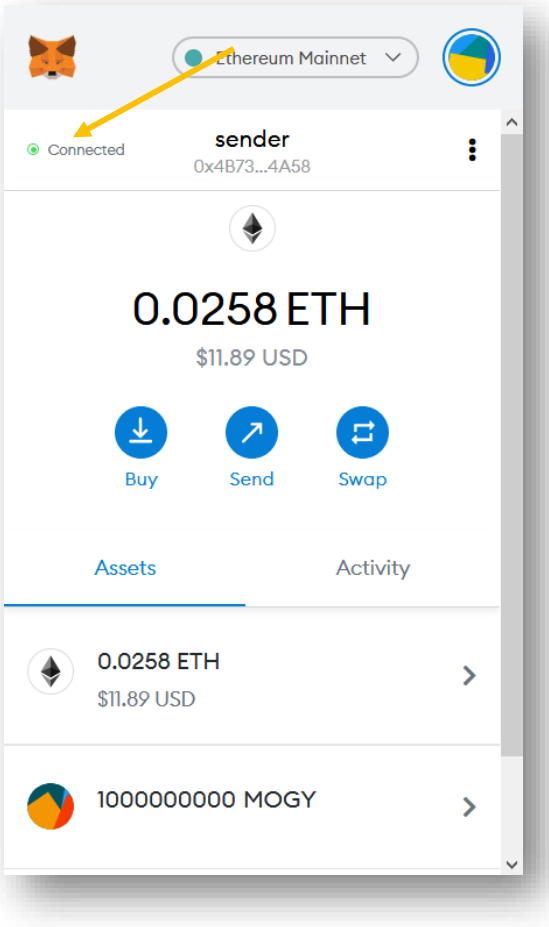


As soon as we upload our new token to the Exchange www.forkdelta.app

When you are in the Exchange site, you will be asked to connect to the site https://forkdelta.app. Click on the "Next" button below, then "Connect" and finally you will be able to check that you are connected to the site of fordelta.app

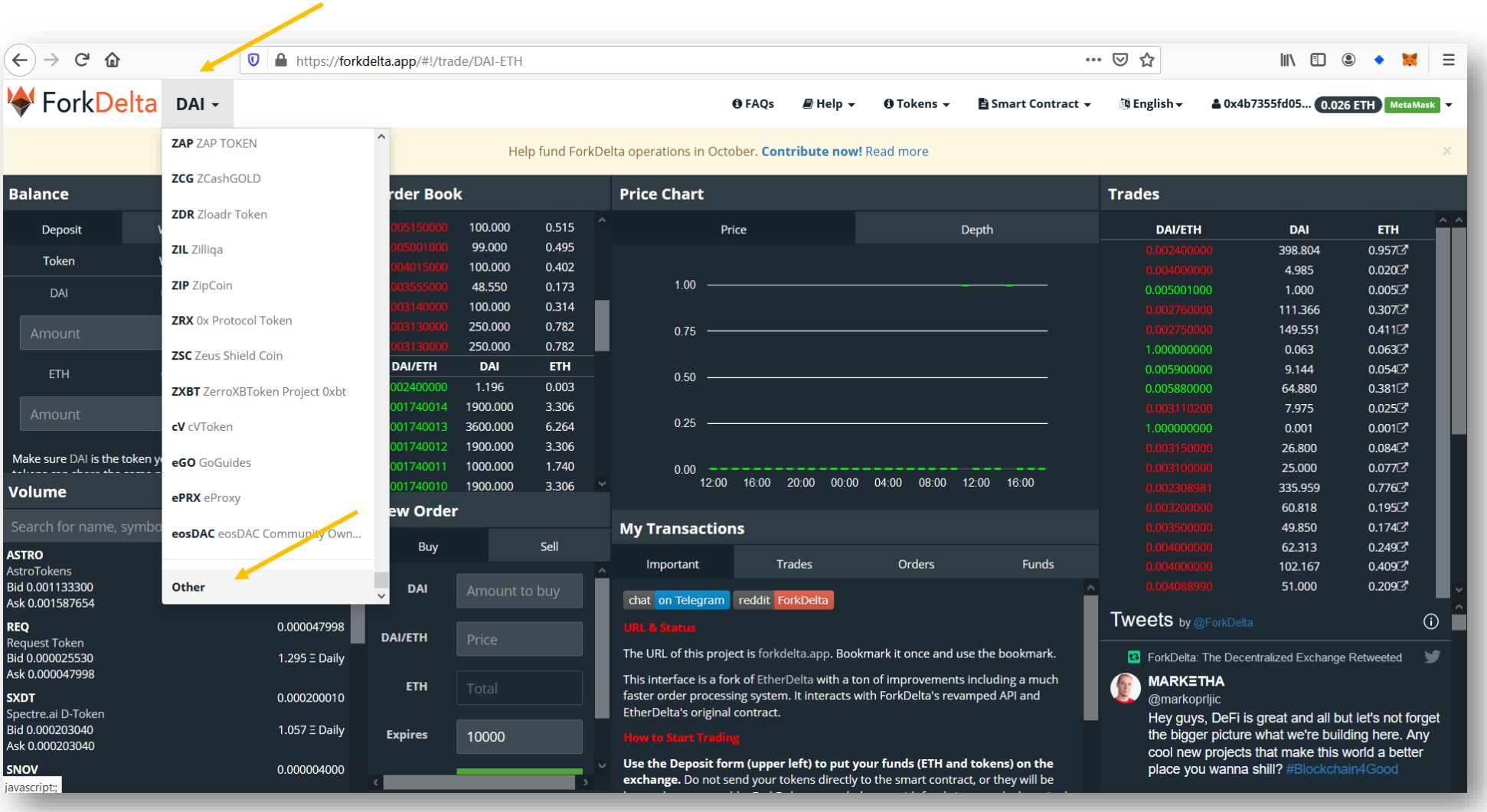




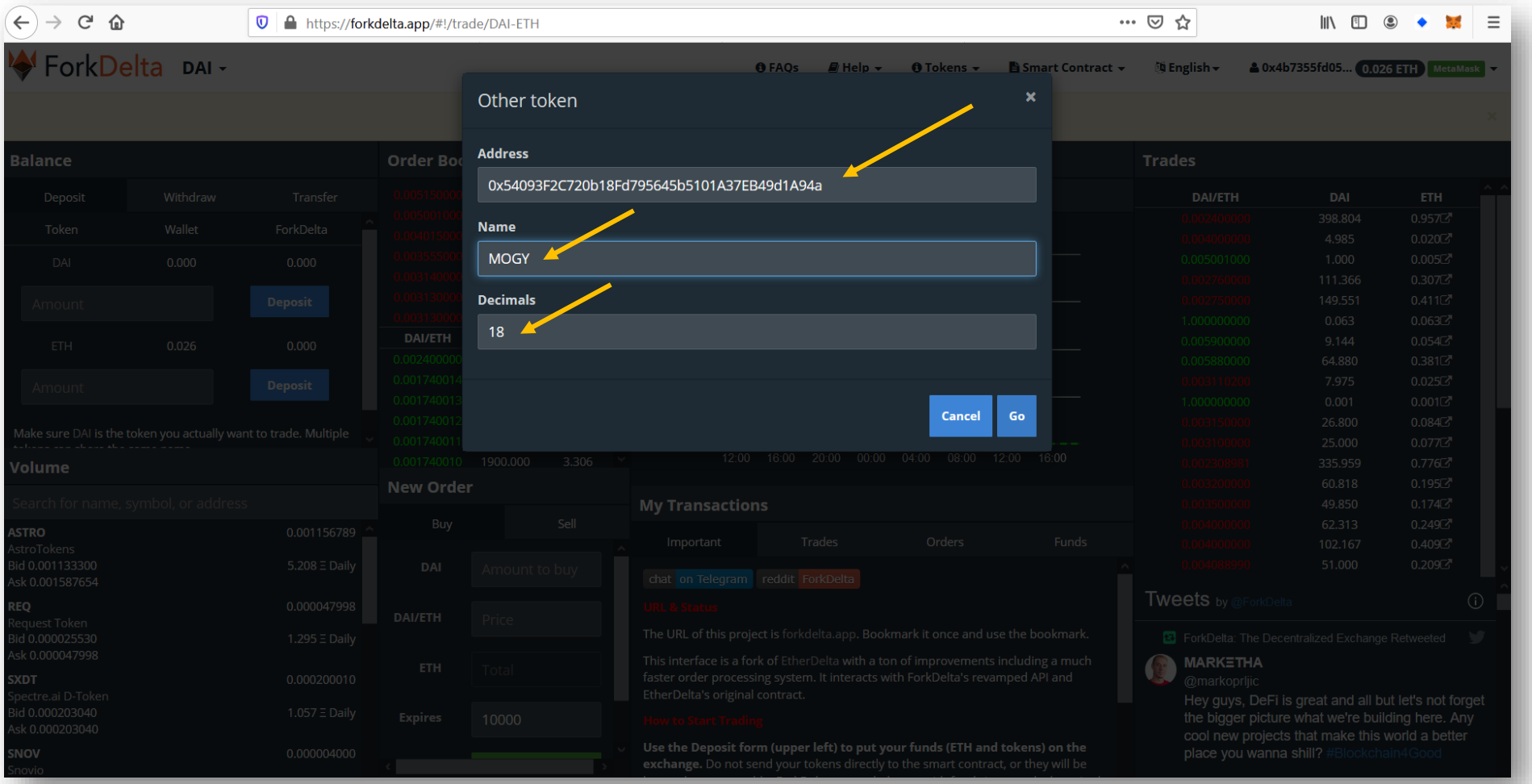


It's time to release the new token on the Exchange https://forkdelta.app

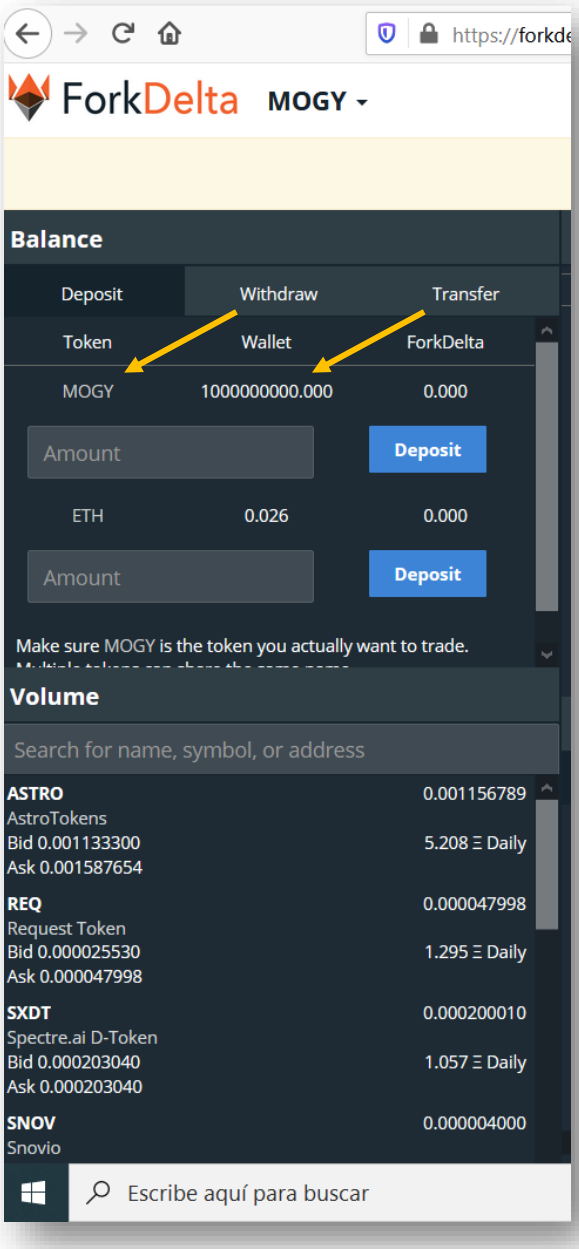
Click on the upper menu "DAI" and go to the end of the scroll and choose the option "Other

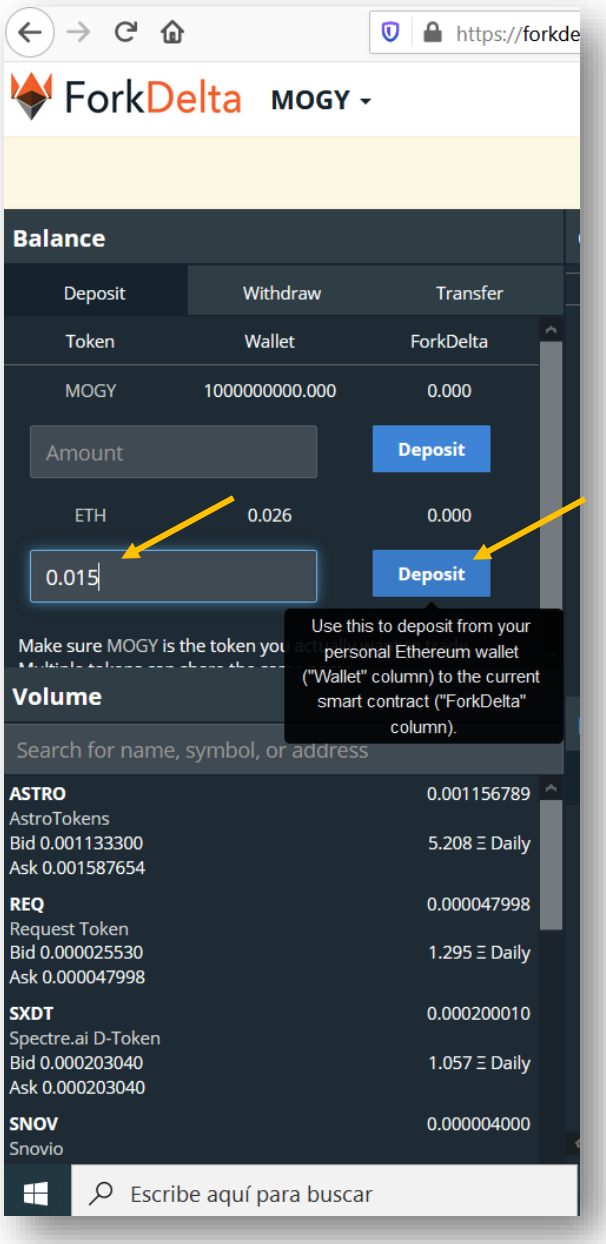


Then we register the new token with the data we already know.



At this moment the new token will appear in the upper left part of the Exchange, we have only uploaded it to the Exchange, we need to publish it so that everyone can buy it and see it. Now we need to deposit some Ether (0.015) is enough from our account to the Exchange.



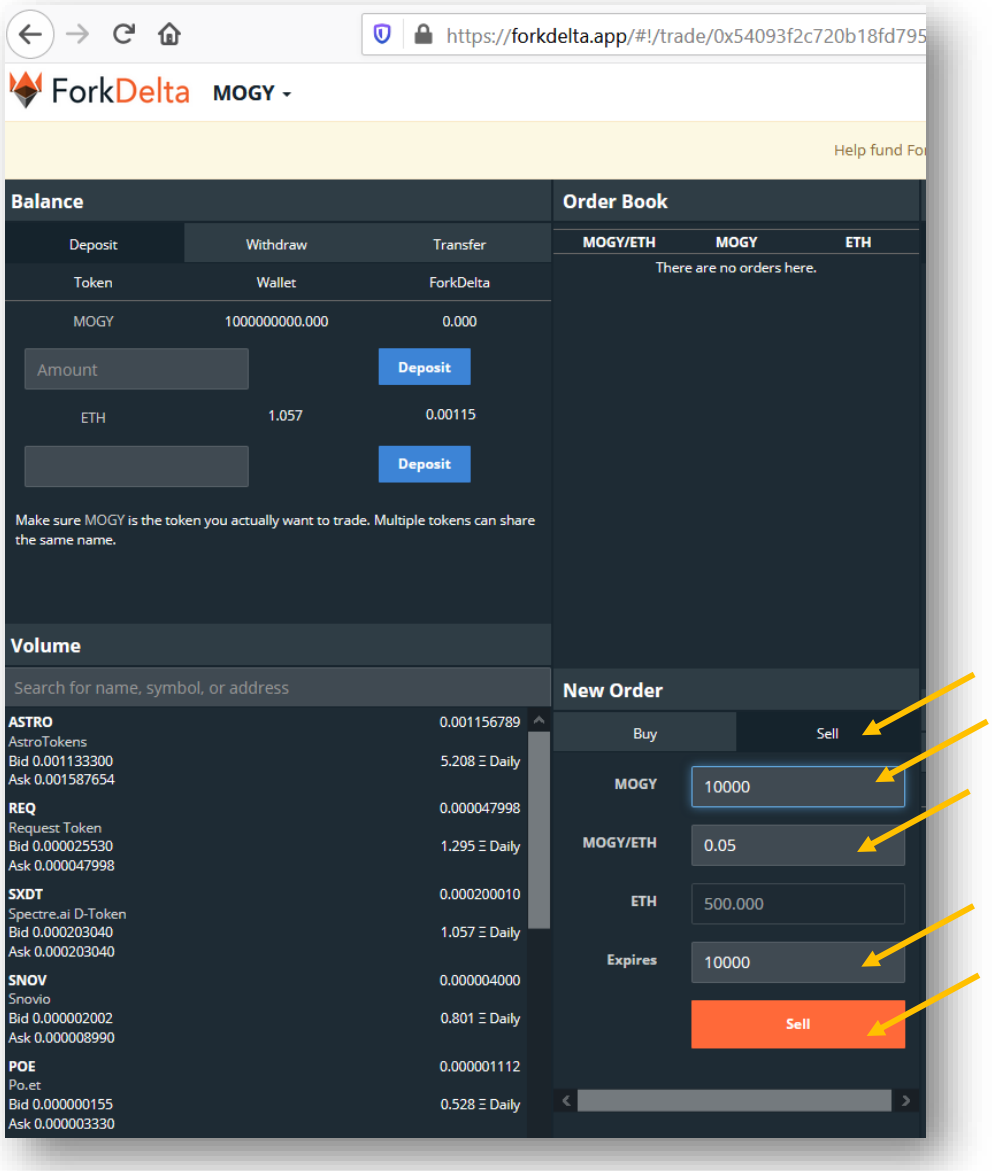


Since we have made the deposit we can deposit the amount of token we want in the Exchange www.forkdelta.app

To deposit a defined amount of tokens we need to create a purchase order, we do this at the bottom where it says "New Order" and we click on the option "Sell". Then we enter the amount of tokens that we want to put at the sight of any buyer in the world, the price in Ether that we want to sell each one (unit price) and the parameter "Expires" is the amount of time we want to have these tokens for sale, this is calculated with the following formula:

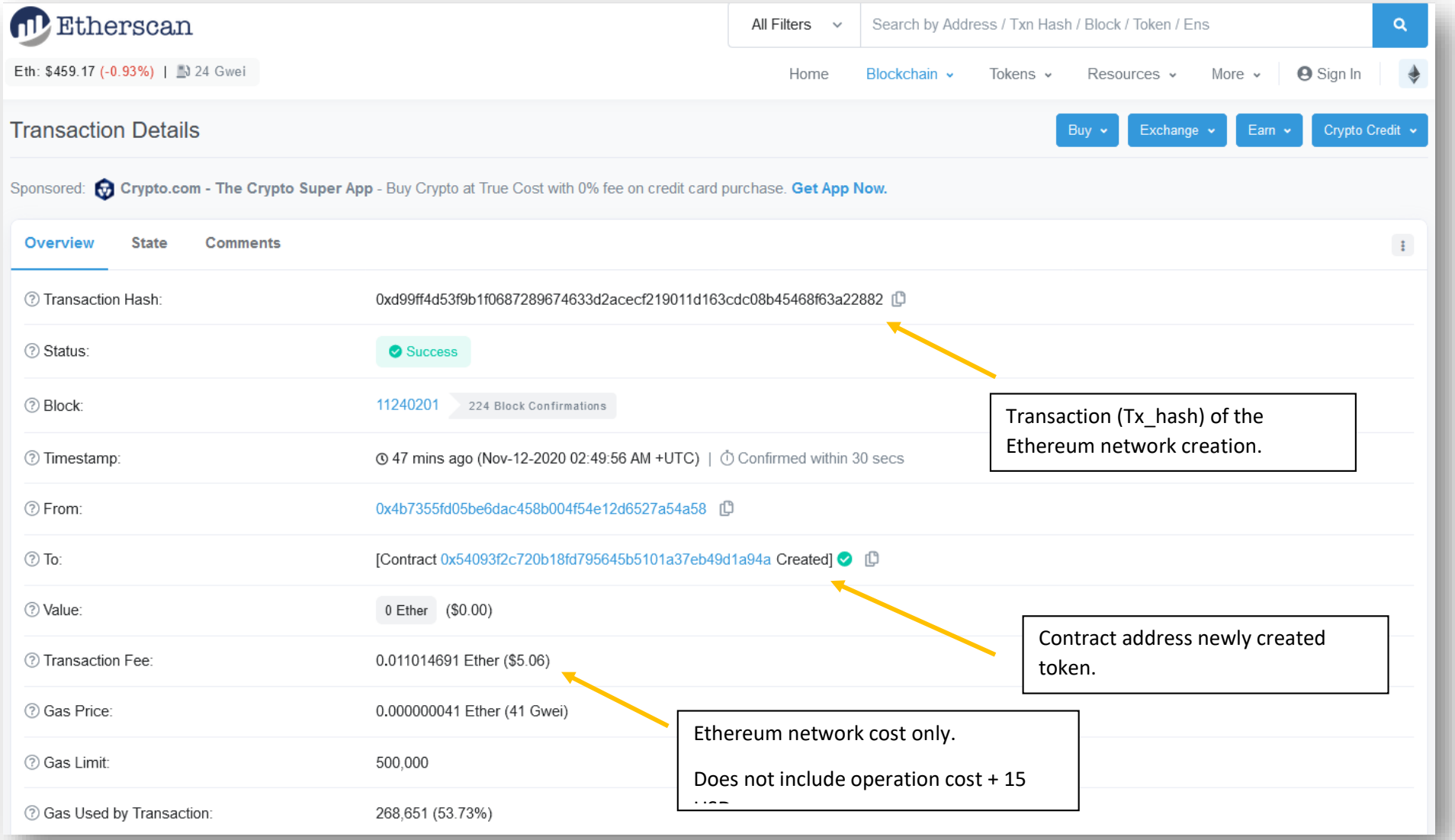
Expires Time = 14 seconds X amount entered.

Example: 14 seconds X 10000 = 140,000 seconds = 1.62 days

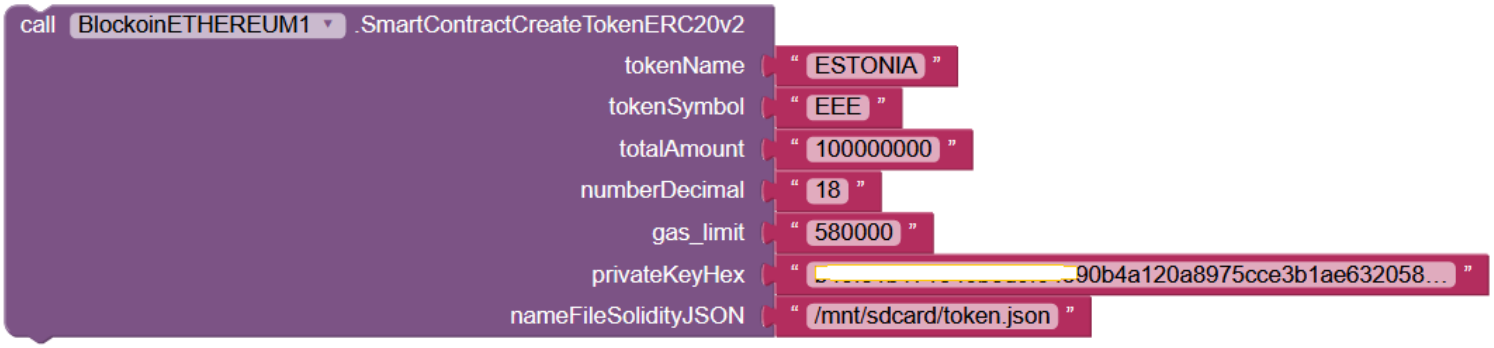


Ready, your new tokens are on sale, anyone can enter and buy them. Sometimes it does not recognize the new token so they will have to be sold from the Metamask application.

Example of consultation on the site [www.etherscan.io of](http://www.etherscan.io) the new token created.



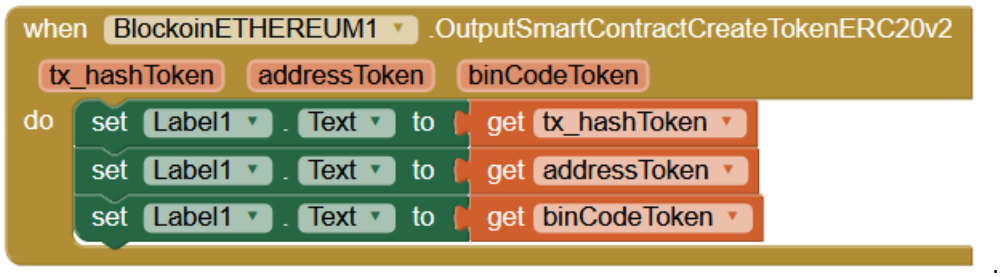
Block to compile, create and publish ERC20 Token - (**SmartContractCreateTokenERC20v2**)



Parámetros de entrada: **tokenName** <String>, **tokenSymbol** <String>, totalAmount **<String>,** numberDecimal **<String>,** gas_limit <Integer> , privateKeyHex <Integer>, nameFileSolidityJSON <String>.

Output parameters: Event (**OutputSmartContractTokenERC20v2**).

Outputs: **tx_hashToken<String>** , **addressToken<String>** , binCodeToken<String**>.**

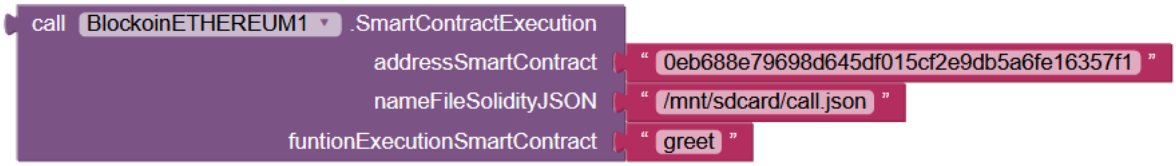


.

Description: Smart contract "Token ERC20" - active published on the Ethereum network. The version 2 (v2) can be optimized the value of the Gas Limit because depending on the smart contract how fast you want to publish on the ethereum network. It is recommended that minimum should be a value of 350,000 WEI to make the publication without failure or delay.

The difference between the functions of creation of TokenERC20v1 and creation of TokenERC20v2 is that in the case of version 1 the parameter of GasLimit is already pre-configured and in version 2 can be configured to the needs of the user or developer.

Block to call or execute ERC20 token - (**SmartContractExecution**)



Input parameters: addressSmartContract<String>, nameFileSolidityJSON<String>, funtionExecutionSmartContract<String**>.**

Output parameters: Execution of function specified in the referenced Smart contract.

Outputs: **Execution of smart contract.**

NOTE: To be executed you must create a file with JSON format which contains the parameters of the primary key of the address you want to run the Smart contract, you need to enter the gas limit (WEI).

Example JSON file to execute the functions of the Smart contract compiled in example of the compiler function mentioned above. The file name can be arbitrary.

File: call.json

```
{
 "private": "3ca40...",
 "gas_limit": 20000
}
```

Example of execution output of the "greet" function of the Smart contract:

```
{
 "gas_limit": 20000,
 "address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",
 "results": [
 "Hello Coinsolidation Test"
 ]
}
```

Block to show properties of Token ERC20 - (SmartContractGetCreationTx)

Input parameters: txSmartContract<String>

Output parameters: Shows properties of Smart contract referenced with (Tx_hash).

Description: Shows the main features and ABI code of the referenced Smart contract.

Example properties of ERC20 token, sample tokenName "MOGY" previously created using the function (martContractCreateTokenERC20v1)


```
{
 "block_hash":
"cadb8914c43ed28fb370c9e1b6f5d8818ba31a1e944d1f7049441b982902dea8",
 "block_height": 11240201,
 "block_index": 170,
 "hash":
"d99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882",
 "addresses: [
     "4b7355fd05be6dac458b004f54e12d6527a54a58"
 ],
 "total": 0,
 "fees": 110146910000000,
 "Size: 958,
 "gas_limit": 500000,
 "gas_used": 268651,
 "gas_price": 41000000000,
 "contract_creation": true,
 "relayed_by": "200.77.24.87",
 "confirmed": "2020-11-12T02:49:56Z",
 "received": "2020-11-12T02:50:13.185Z",
 "see": 0,
 "double_spend": false,
 "vin_sz": 1,
 "vout_sz": 1,
```

```
"confirmations": 333,
"confidence": 1,
"inputs: [
    {
      "sequence: 37,
      "addresses: [
        "4b7355fd05be6dac458b004f54e12d6527a54a58"
      ]
    }
],
"outputs": [
    {
      "value: 0,
      "script":
"6080604052348015610010576000 80fd5b506040805180820190915260048082527f4d4f
475900000000000000000000000000000000000000000000000000000000000060209092019 18
252610055916002916100c7565b506040805180820190915260048082527f4d4f47590000 0
000000000000000000000000000000000000000000000000000000602090920191825261009
a916003916100c7565b506b033b2e3c9fd0803ce8000000600181905533600090815260020
819052604090205560126004556101625 65b828054600181600116156101000203166002 9
004906000526020600020906 01f016020900481019282601f10610108578051 60ff191683
800117855561013556 5b828001600101855582156101355791820 15b828111156101355 78
251825591602001919060010190 61011a565b506101419291506101455 65b5090565b6101
5f91905b8082111561014157600081556001016101 4b565b90565b6101f78061017160003
96000f300608060405260043610610061576 3ffffffff7c0100000000000000000000000
0000000000000000000000000000000000600035041663181 60ddd81146100665780637 06370a08
2311461008d578063a9059cbb146100bb578063b9f0bf6614610100575b600080fd5b3480
156100725760008 0fd5b5061007b61012e565b604080519182525190819003602001 90f35
b3
      "addresses: [
"54093f2c720b18fd795645b5101a37eb49d1a94a"
      ]
    }
]
```


Block to display ERC20 token compilation code - (SmartContractGetcodeABI)

Input parameters: **addressSmartContract<String>**

Output parameters: Show referenced Smart contract code.

Description: Shows ABI code of the referenced Smart contract.

Example of ABI code of a generic Smart contract:

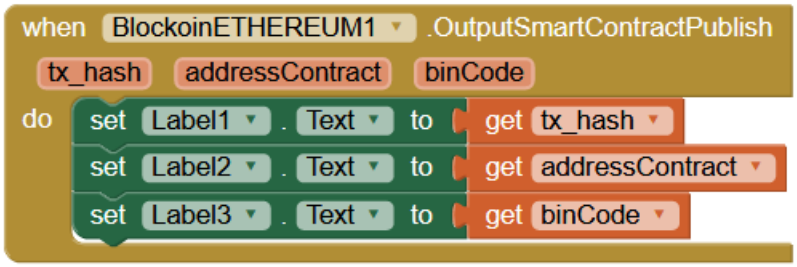

```
{
 "solidity": "contract mortal {\n /* Define variable owner of the type
address*/\n     address owner;\n     /* this function is executed at
initialization and sets the owner of the contract */\n     function
mortal() { owner = msg.sender; }\n     /* Function to recover the funds
on the contract */\n     function kill() { if (msg.sender == owner)
suicide(owner); }\n}\n\ncontract greeter is mortal {\n     /* define
variable greeting of the type string */\n     string greeting;\n     /*
this runs when the contract is executed */\n     function greeter(string
_greeting) public {\n         greeting = _greeting;\n     }\n     /* main
function */\n     function greet() constant returns (string) {\n
return greeting;\n     }\n}",
 "bin":
"606060405260405161023e38038061023e8339810160405280510160008054600160a060
020a0319163317905580600160005090805190602001908280546001816001161561010000
203166002900490600052602060002090601f016020900481019282601f10609f57805160
ff19168380011785555b50608e9291505b8082111560cc5760008155830160 7d565b50505
061016e806100d06000396000f35b828001600101855582156076579182015b8281111560
76578251826000505591602001919060010190 60b0565b5090566060606040526 0e060020a6
00035046341c0e1b5811461002657806 3cfae321714610068575b005b6100246000543373
ffffffffffffffffffffffffffffffffffffffff908116911614156101375760005473fff
ffffffffffffffffffffffffffffffffffffff16ff5b6100c960006060908152600180546 0
a06020601f60026000196101008688161502019094169390930492830181900402810160 4
052608082815292919082828015610164578060 1f10610139576101008083540402835291
6020019161016456 5b604051808060200182810382528381815181526020019150805190 6
020019080838382906000600460208460 1f0104600f02600301f150905090810190601f16
801561012957808203805160018360200361010 00a031916815260200191505b509250505
06
 "abi":
"[{\"constant\":false,\"inputs\":[],\"name\":\"kill\",\"outputs\":[],\"ty
pe\":\"function\"},{\"constant\":true,\"inputs\":[],\"name\":\"greet\",\"
outputs\":[{\"name\":\"\",\"type\":\"string\"}],\"type\":\"function\"},{\
"inputs\":[{\"name\":\"_greeting\",\"type\":\"string\"}],\"type\":\"const
ructor\"}]",
 "creation_tx_hash":
"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
 "created": "2016-07-20T01:54:50Z",
 "address": "0eb688e79698d645df015cf2e9db5a6fe16357f1"}
```


*Before using the following Block **(SmartContractPublish) you** must use the Block (**SmartContractCompilerSolidity)** to check if the Smart contract is well written.

Block to publish in network Ethereum the ERC20 token - (**SmartContractPublish**).

Input parameters: nameFileSolidityJSON<String>

Output parameters: Shows properties of referenced Smart contract. In the event (OutputSmartContractPublish).



Description: Publish in ethereum network the Smart contract referenced in the JSON file.
Example file: PubishSmartContract.json

```
{
 "solidity": "contract mortal {\n /* Define variable owner of the type
address*/\n     address owner;\n     /* this function is executed at
initialization and sets the owner of the contract */\n     function
mortal() { owner = msg.sender; }\n     /* Function to recover the funds
on the contract */\n     function kill() { if (msg.sender == owner)
suicide(owner); }\n}\n\ncontract greeter is mortal {\n     /* define
variable greeting of the type string */\n     string greeting;\n     /*
this runs when the contract is executed */\n     function greeter(string
_greeting) public {\n         greeting = _greeting;\n     }\n     /* main
function */\n     function greet() constant returns (string) {\n
return greeting;\n     }\n}",
"params": ["Hello Test"],
"publish": ["greeter"],
"private": "3ca40...",
"gas_limit": 500000
}
```

As shown in the code above is the same JSON code used in the example of the compilation function to the same code have been added parameters at the end of the JSON file:

Params: Implicit parameters of the Smart contr.
Publish: Name of how the Smart contract will be published.
Private: Primary key of the account that will execute the Smart contract must have a balance.
Gas_limit: Is the balance in WEI that you want to spend for the publication of the Smart contract.

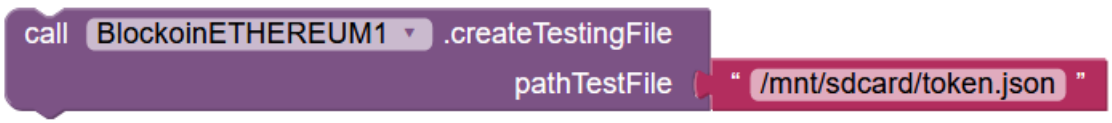
Example of output when running (publishing Smart contract) the Smart contract is introduced into the ethereum network. It shows the transaction made "creation_tx_hash" and the assigned address of the created Smart contract "address".

```
[
```

```
 {
     "name": "greeter",
     "solidity": "contract mortal {\n /* Define variable owner of the
type address*/\n     address owner;\n     /* this function is executed at
initialization and sets the owner of the contract */\n     function
mortal() { owner = msg.sender; }\n     /* Function to recover the funds
on the contract */\n     function kill() { if (msg.sender == owner)
suicide(owner); }\n}\n\ncontract greeter is mortal {\n     /* define
variable greeting of the type string */\n     string greeting;\n     /*
this runs when the contract is executed */\n     function greeter(string
_greeting) public {\n         greeting = _greeting;\n     }\n     /* main
function */\n     function greet() constant returns (string) {\n
return greeting;\n     }\n}",
     "bin":
"606060405260405161023e38038061023e833981016040528051016000805460016060a060
020a031916331790558060016000509080519060200190828054600181600116156101000
203166002900490600052602060002090601f016020900481019282601f10609f57805160
ff19168380011785555b50608e9291505b8082111560cc5760008155830160 7d565b50505
061016e806100d06000396000f35b828001600101855582156076579182015b8281111560
765782518260005055916020019190600101906 0b0565b5090566060604052 60e060020a6
00035046341c0e1b5811461002657806378063cfae321714610068575b005b6100246000543373
ffffffffffffffffffffffffffffffffffffffff90811691161415610137576000 5473fff
ffffffffffffffffffffffffffffffffffffff16ff5b6100c960006060908152600180546 0
a06020601f60026000196101008688161502019094169390930492830181900402810160 4
052608082815292919082828015610164578060 1f106101395761010080835404028352 91
602001916101645 65b604051808060200182810382528381815181526020019150805190 6
020019080838382906000600460208460 1f0104600f02600301f150905090810190601f16
801561012957808203805160018360200361010 00a031916815260200191505b509250505
06
     "abi": [
       {
         "constant": false,
         "inputs: [],
         "name": "kill",
         "outputs: [],
         "type": "function"
       },
       {
         "constant": true,
         "inputs: [],
         "name": "greet",
         "outputs": [
           {
             "name": "",
             "type": "string"
           }
         ],
         "type": "function"
       },
       {
         "inputs: [
           {
             "name": "_greeting",
             "type": "string"
           }
         ],
         "type": "builder"
```

```
        }
      ],
      "gas_limit": 500000,
      "creation_tx_hash":
"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
      "address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",
      "params": [
        "Hello Test"
      ]
  },
  {
      "name": "mortal",
      "solidity": "contract mortal {\n /* Define variable owner of the
type address*/\n     address owner;\n     /* this function is executed at
initialization and sets the owner of the contract */\n     function
mortal() { owner = msg.sender; }\n     /* Function to recover the funds
on the contract */\n     function kill() { if (msg.sender == owner)
suicide(owner); }\n}\n\ncontract greeter is mortal {\n     /* define
variable greeting of the type string */\n     string greeting;\n     /*
this runs when the contract is executed */\n     function greeter(string
_greeting) public {\n         greeting = _greeting;\n     }\n     /* main
function */\n     function greet() constant returns (string) {\n
return greeting;\n     }\n}",
      "bin":
"606060405260008054600160a060020a0319163179055605c80602260003960000f36060
60405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffffffff
ffffffffffffffffffffffffffffffff90811691161415605a5760005473ffffffffffffff
ffffffffffffffffffffffff16ff5b56",
      "abi": [
        {
          "constant": false,
          "inputs: [],
          "name": "kill",
          "outputs: [],
          "type": "function"
        },
        {
          "inputs: [],
          "type": "builder"
        }
      ],
      "gas_limit": 500000,
  "params: [ "Hello Test"]
  }
]
```


Test block to **create** file - (**createTestingFile**)



Input parameters: **pathTestFile<String>**

Output parameters: A test file is created in the referenced path.

Description: This serves to check the valid path of temporary file creation to ensure that it is correct when we use the Block (**SmartContractCreateTokenERC20v1)** or the Block **(SmartContractCreateTokenERC20v2).**

Block to get the rates of Gas Price - (**eth_RatesGasStationInfo**).

Input parameters: **nameFileSolidityJSON<String>**

Output parameters: Shows properties of referenced Smart contract. In the event **(OutputEth_GasStationInfo)** the delivered values are given in GWEI.

The Gas Price is the value that will be paid to the systems that execute the transactions in the Ethereum's network. These systems are commonly known as "miners" and the values of the Gas Price is a function of how fast (time and priority) the transaction is performed in the Ethereum's network.

The values delivered are a function of the following execution times. These times are approximate and may vary depending on how you are in the demands (transactions) in the Ethereum network.

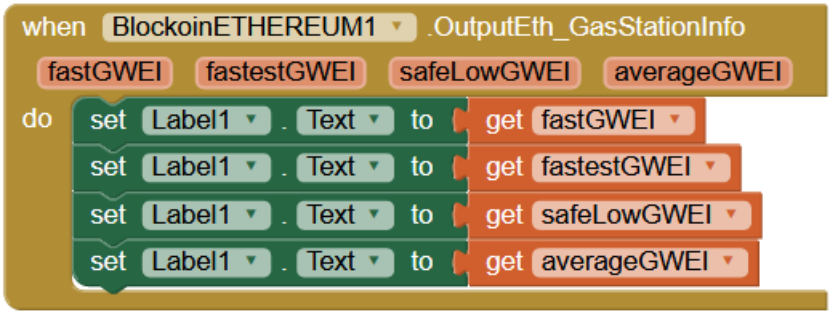
Fast < 2 minutes.

Fastest < 30 seconds.

SafeLow < 30 minutes.

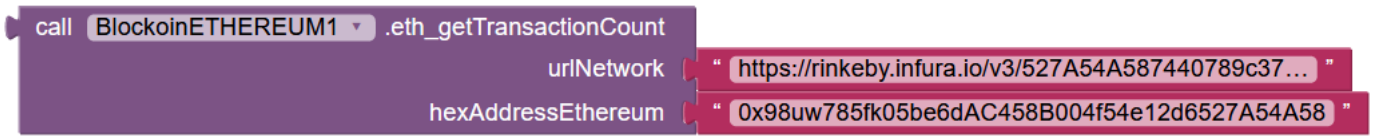
Average < 15 minutes.

Transactions made with the Exchange Ethereum Extension (EEE) always use the Gas Price = Average.



Description: Gets the updated Gas Price at the time of the query to create a new transaction.

Block to obtain the number "nonce" - (**eth_getTransactionCount**).



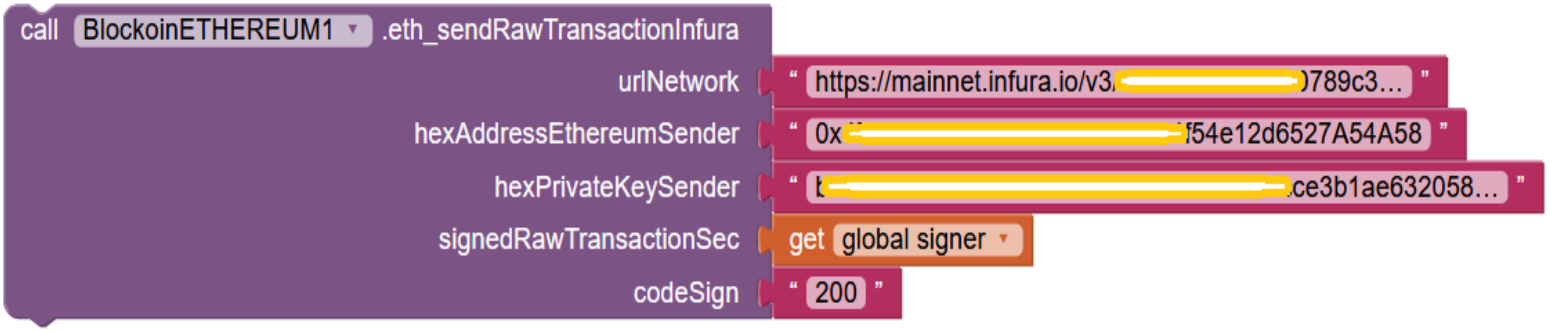
Input parameters: **urlNetwork<String>**, hexAddressEthereum<String>.

Output parameters: Shows in hexadecimal format the consecutive number "nonce" of the referenced address.

The "nonce" number is an incremental number that keeps track of the number of transactions that have been made from a specific address.

Description: Obtains the "nonce" number of the referenced address.

Block to send a signed transaction - (**eth_SendRawTransactionInfura**).

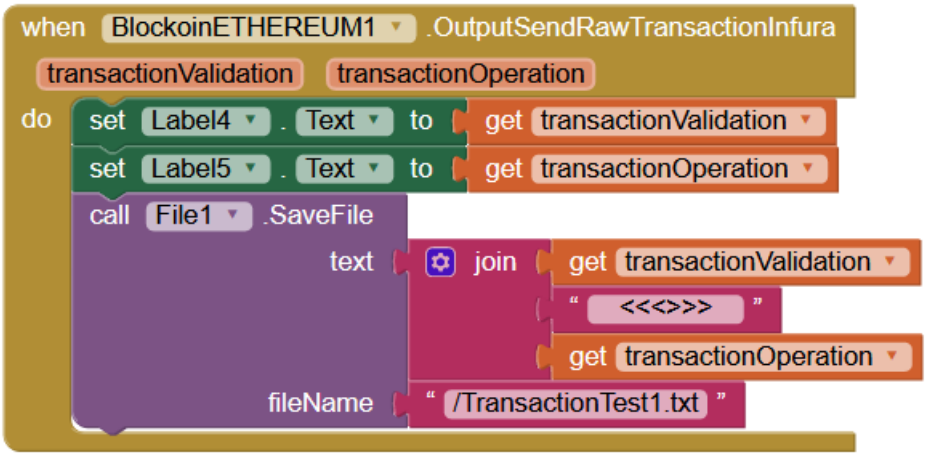


Required dependency(s): Block (**eth_getTransactionCount)**, Block (**SignerGenericPushRawTransactionOffline**). Check example of the block (**SignerGenericPushRawTransactionOffline**).

Input parameters: **urlNetwork <String>**, hexAddressEthereumSender **<String>,** hexPrivateKeySender <String>, **SignedRawTrasactionSec <String>, codeSign** <String>.

Output parameters: Event (**OutputSendRawTransactionInfura**).

Outputs: transactionValidation<String> , transactionOperation<String>.



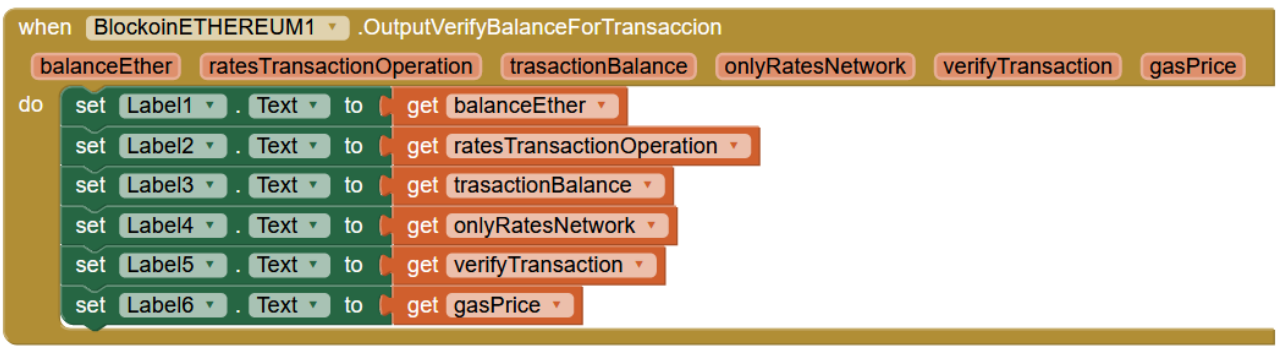
Description: It delivers two hexadecimal values as a result of the transactions. The transactionValidation value is the transaction made on the Ethereum network that contains the implicit cost of the Ethereum. The value of transactionOperation is the transaction made in the Coinsolidation network with the cost of $0.5 cents USD for each transaction based on the value of the ether at the time of the transaction. This cost is for the payment of services of the Coinsolidation.org network and is invested in the maintenance, support and creation of extensions for the crypto-currency sector and assets worldwide.

Block to calculate cost of a standard transaction - (**eth_VerifiBalanceForTransaction**)

Input parameters: **addressEthereumSender<String>**, valueEthertoSend<String>.

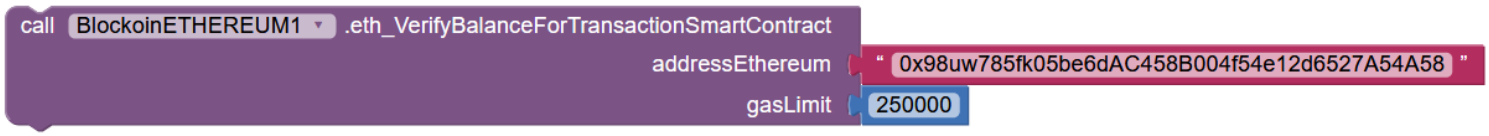
Output parameters: Event (**OutputVerifyBalanceForTransaction**).

Outputs: balanceEther<String> , ratesTransactionOperation<String> , **trasactionBalance<String>** , **OnlyRatesNetwork<String>** , **verifyTransaction<String>** , gasPrice<String>.



Description: Provides details of what the cost of a standard transaction will be with reference to the address of entry. The output parameter "verifyTransaction" tells us if the transaction can be made "True" or if the address referenced does not have enough balance will give us a "False".

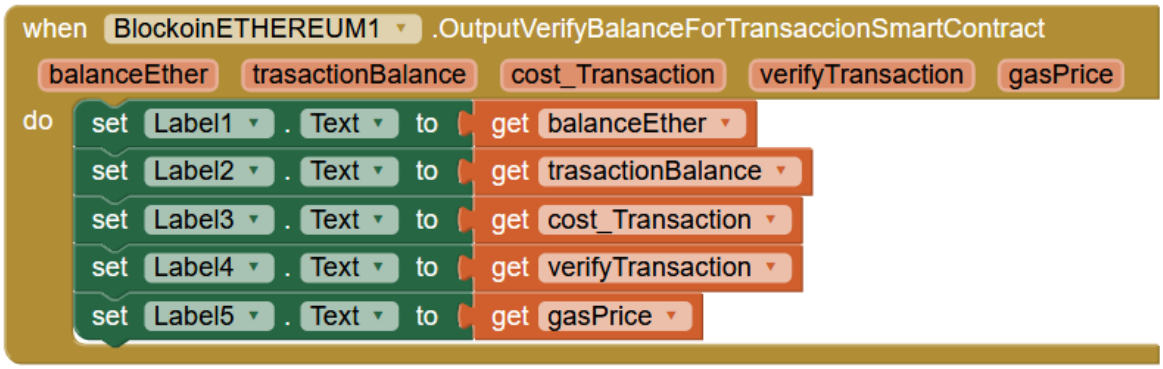
Block to calculate cost of a standard transaction - (**eth_VerifiBalanceForTransaccionSmartContract**)



Input parameters: **addressEthereum<String>**, gasLimit<String>.

Output parameters: Event (**OutputVerifyBalanceForTransaccionSmartContract**).

Outputs: balanceEther<String> , trasactionBalance<String> , **cost_Transaction<String> , verifyTransaction<String> ,** gasPrice<String>.



Description: Provides details of what the approximate cost of a standard transaction would be to publish a **Smart contract**, reference to the address of entry. The output parameter "verifyTransaction" tells us if the transaction can be made "True" or if the address referenced does not have enough balance will give us a "False".

balanceEther: Balance of the address referenced is delivered in ethers.

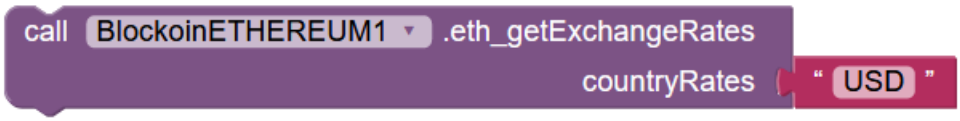
trasactionBalance: Balance after the transaction is made.

cost_Transaction: Is the cost of the transaction to publish the smart contrat.

verifyTransaction: (balanceEther minus cost_Transaction).

gasPrice: Current value of the GasPrice used by the "Miners", it can vary every minute.

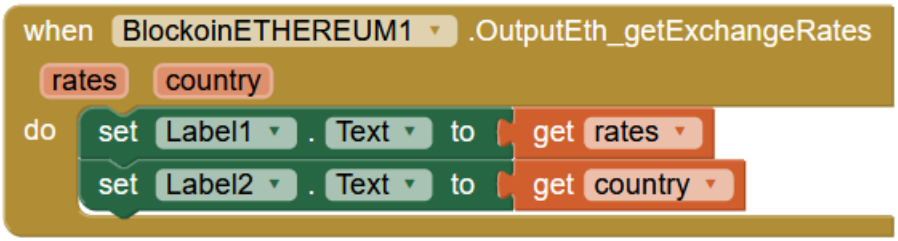
Block to get Ether price in currency of a specified country - (**eth_getExchangeRates**).



Input parameters: **countryRates<String>**. Review output parameter "country" where it contains all the country currency types to choose the desired one.

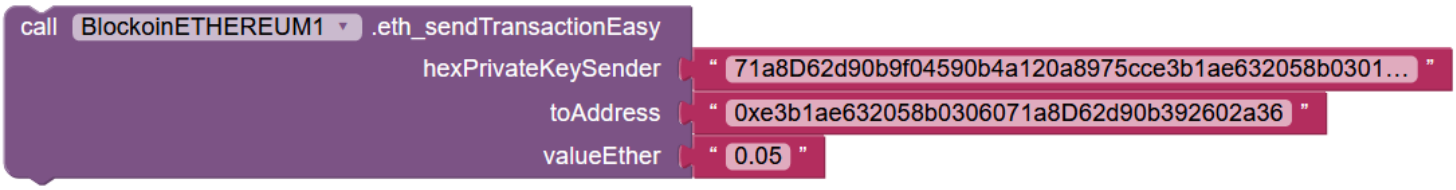
Output parameters: Event (**OutputEth_getExchangeRates**).

Outputs: **rates<String>**, **countries<String>** output in JSON format all the rates of the countries of the world



Description: It delivers the current price of an Ether at the exchange rate of the currency of the referenced country.

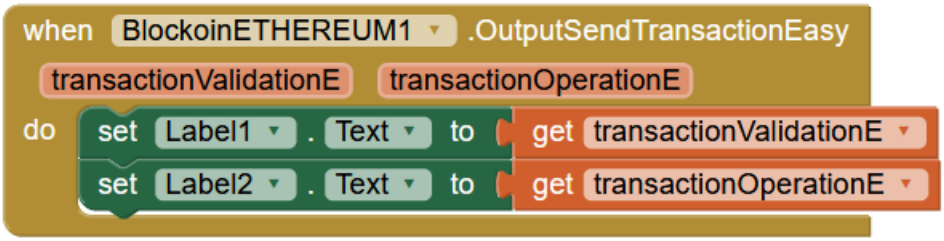
Block to perform a standard transaction with the minimum input parameters - (**eth_sendTransactionEasy**).



Input parameters: **hexPrivateKeySender<String>**, toAddress<String>, valueEther<String**>.**

Output parameters: Event (**OutputSendTransactionEasy**).

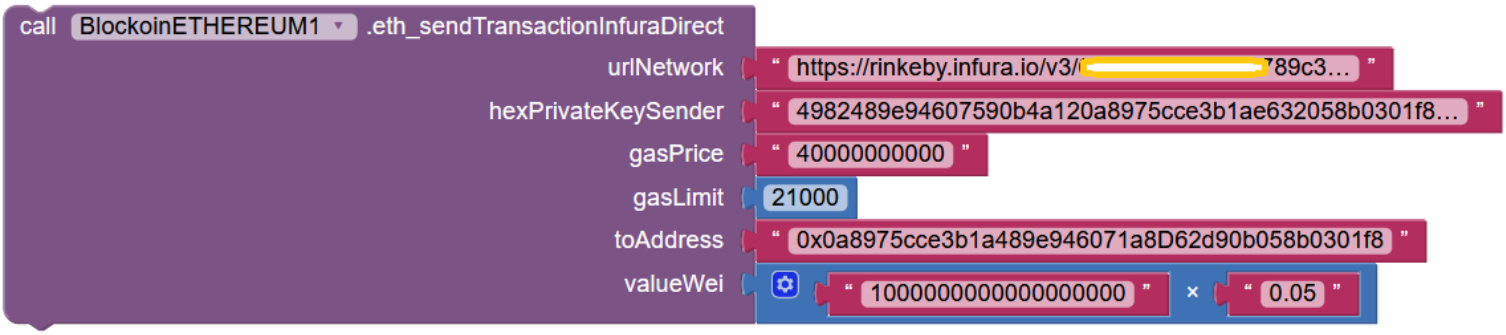
Outputs: transactionValidation<String> , transactionOperation<String>.



Description: Function to make a standard transaction in the Ethereum network, this function is of immediate use you don't need to have an account in INFURA and you only need 3 input parameters, you only need to have enough balance to make the desired transaction.

Transactions are placed on the Ethereum network directly using the official Ethereum Web3j library and our Coinsolidation.org network

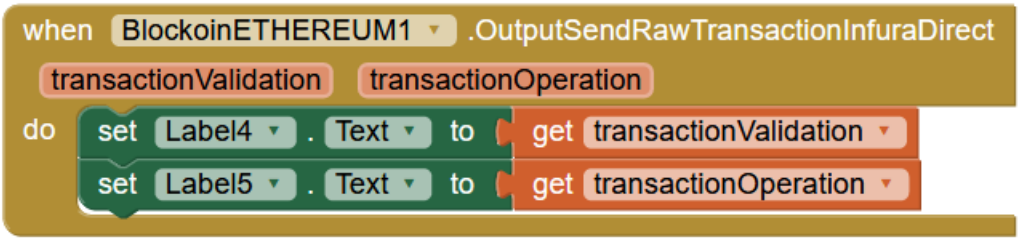
Block to perform a standard transaction with the minimum input parameters - (**eth_sendTransactionInfuraDirect**).



Input parameters: **urlNetwork<String>**, hexPrivateKeySender<String>, gasPrice<String> **,** gasLimit<String>, **toAddress<String>,** valueWEI<String>.

Output parameters: Event (**OutputSendTransactionInfuraDirect**).

Outputs: transactionValidation<String> , transactionOperation<String>.



Description: Function to send a standard transaction that already contains the implicit digital signature, this is useful for people who already have previous knowledge of the components of a transaction and want to optimize these parameters according to their needs.

11.Standard and Smart contract transaction cost calculation.

For the calculation of a standard transaction, 3 parameters are needed in the Ethereum network.

1.- **Gas Price.**
2.- **Gas Limit.**
3 .- **Current value of an Ether** (Ethereum digital currency).

GasPrice: This is normally given in GWEI units (GigaWEI). If 1 ether has 1,000,000,000,000,000,000 wei then 1 GWE is equal to 1,000,000,000. This unit serves as the payment for the systems that execute all the transactions of the Ethereum network and are called "miners", it is distributed all over the world. The GasPrice is not a fixed value and is variable and can change from minute to minute or second. Those who define the value of GasPrice are the "miners" and it depends on how saturated the Ethereum network is.

GasLimit: This value is normally given in units of WEI and in a standard transaction an average default value is 21,000 WEI although it can be higher or lower depending on what type of transaction you want to make, in a standard transaction we use the value for 40,000 WEI to ensure that we do not have rejection for having under Gas Limit.

Ether's Value: This value is also variable and is due to different parameters of a global financial market, this value can be obtained from entities that have always updated the value of Ether worldwide called centralized and decentralized exchanges.

IMPORTANT NOTE: In the Exchange Ethereum Extension (EEE) we use a higher Gas Limit (40,000) this is because in case of not reaching the minimum quota established by the "miners" THE TRANSACTION IS NOT EFFECTED, however the Ethereum network, if it will charge the expense that has made in the calculation of the transaction without making it, For this reason some users do not explain why their transaction is not performed, however an amount or all of the GasLimit that was offered when the transaction request was launched will be charged, for this reason we must always be clear what the values of GasLimit and Gas Price will be.

The GasPrice can be consulted with the function **eth_GetRatesGasStation** and the GasLimit for standard transactions must be higher than 21,000 WEI and transactions to publish and/or execute Smart contract must be at least 500,000 WEI.

Standard transaction cost.

It is defined by the following formula:

Cost = (GasLimit x (GasPrice / (1,000,000,000,000,000)) x Ether Value.

Example:

Let's assume the following values:

GasLimit = 40,000, GasPrice = 45 GWEI, Ether Value = 406 USD.

Cost = (40,000 x (45,000,000,000 / (1,000,000,000,000)) x 406 USD

Standard transaction cost = 0.0018 ether x 406 USD = $0.73 USD

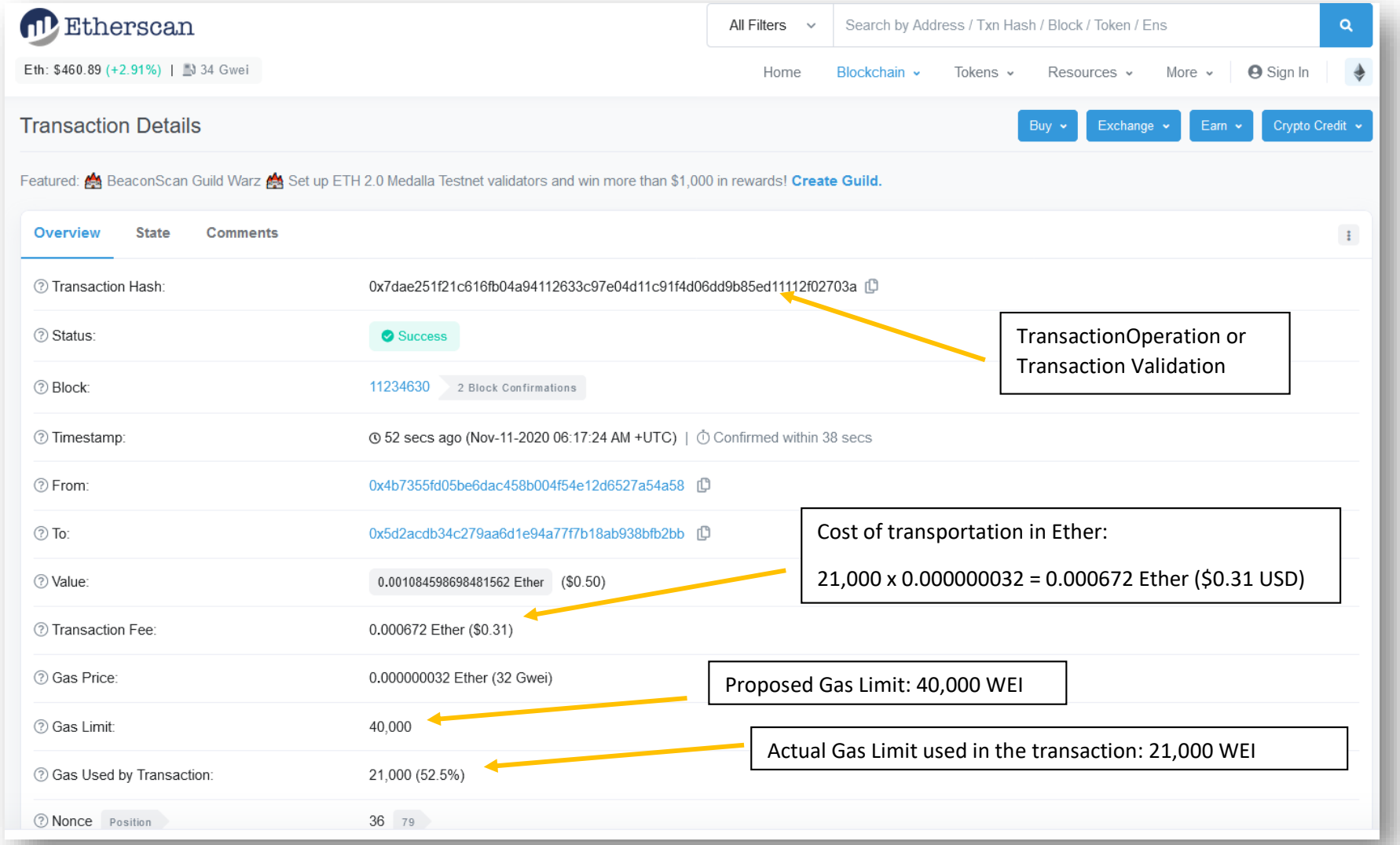
In the case of a Smart contract transaction, it is necessary to know what type of Smart contract will be published since the cost is directly proportional to the workload that the "miners" will have to perform in order to process the Smart contract, in other words, the amount of processing in the computer systems handled by the "miners" is simpler.

In the case of Smart contracts a default value would be to start the GasLimit with a value of 500,000 WEI.

An important point to take into account is that when one proposes a GasLimit, the "miners" will not necessarily take the entire amount proposed, that is, when one sends a transaction, the "miners" calculate the computation effort and take out what will be taken out of the GasLimit, which may be less than or equal to the default GasLimit of 21,000 WEI offered in some cases.

All the transactions will be able to be consulted in the site www.etherscan.io where we can consult the detail of each transaction.

Example of a standard transaction, where the transaction was sent with a Gas Limit of 40,000 WEIs, however the "miners" when calculating how much the processing cost would be took only 52.5% i.e. the default value which is 21,000 WEIs.



By returning to the Smart contract transaction and taking the 500,000 WEI GasLimit we get the following cost if we use it all.

Smart contract transaction cost.

It is defined by the following formula:

Cost = (GasLimit x (GasPrice / (1,000,000,000,000,000)) x Ether Value.

Example:

Let's assume the following values:

GasLimit = 500,000, **GasPrice =** 45 GWEI, **Ether Value** = 406 USD.

Cost = (500,000 x (45,000,000,000 / (1,000,000,000,000)) x 406 USD

Transaction cost publish Smart contract = 0.0225 ether x 406 USD = $9,135 USD

All transactions can be consulted on the site www.etherscan.io

NOTE: To get the total cost of the transaction must be added:

Total Transaction Cost = *Ethereum network cost + Coinsolidation.org fee*

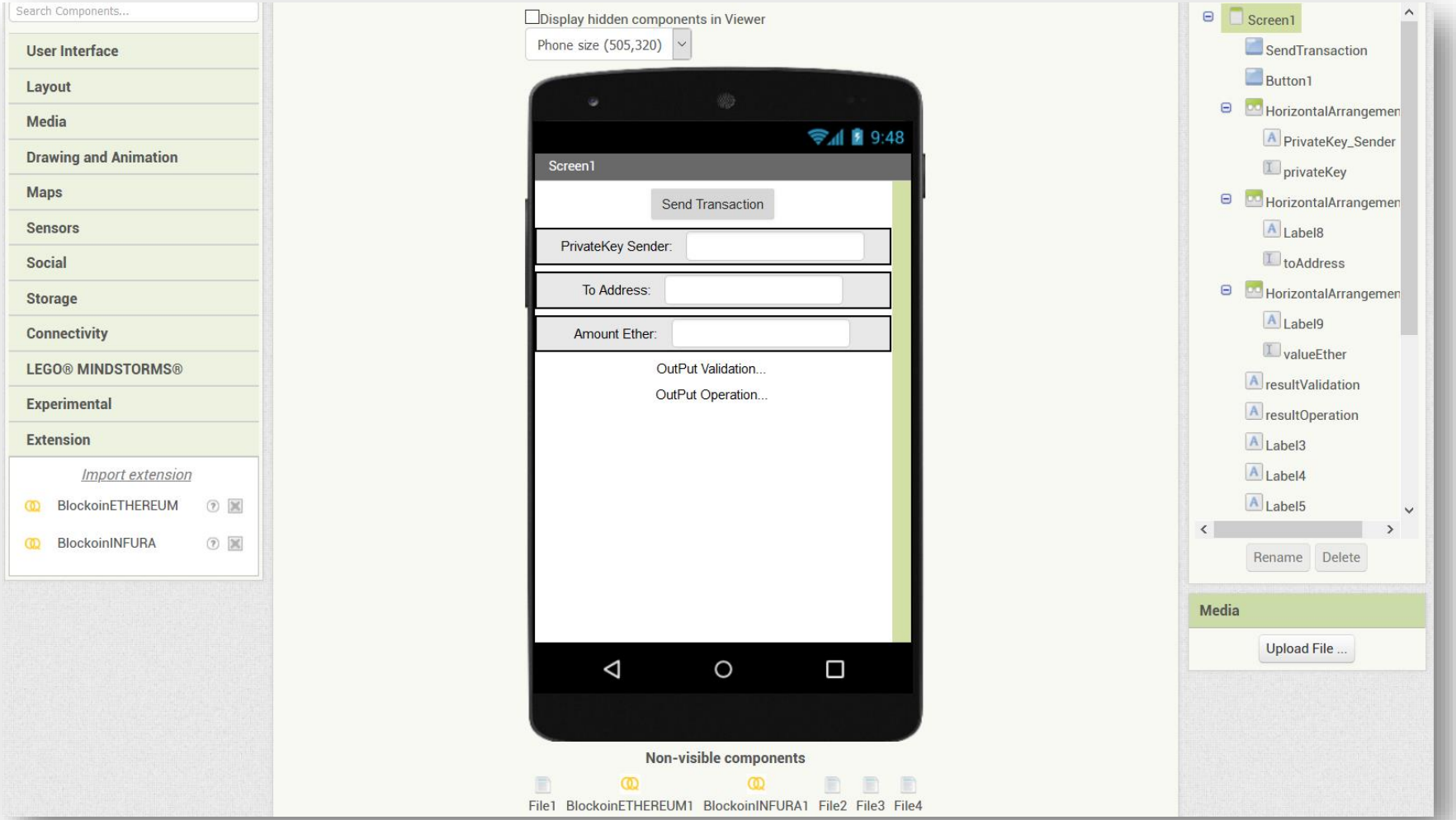
12.Coinsolidation.org Fees

Standard transaction: $ 0.5 cents USD + cost of Ethereum network.

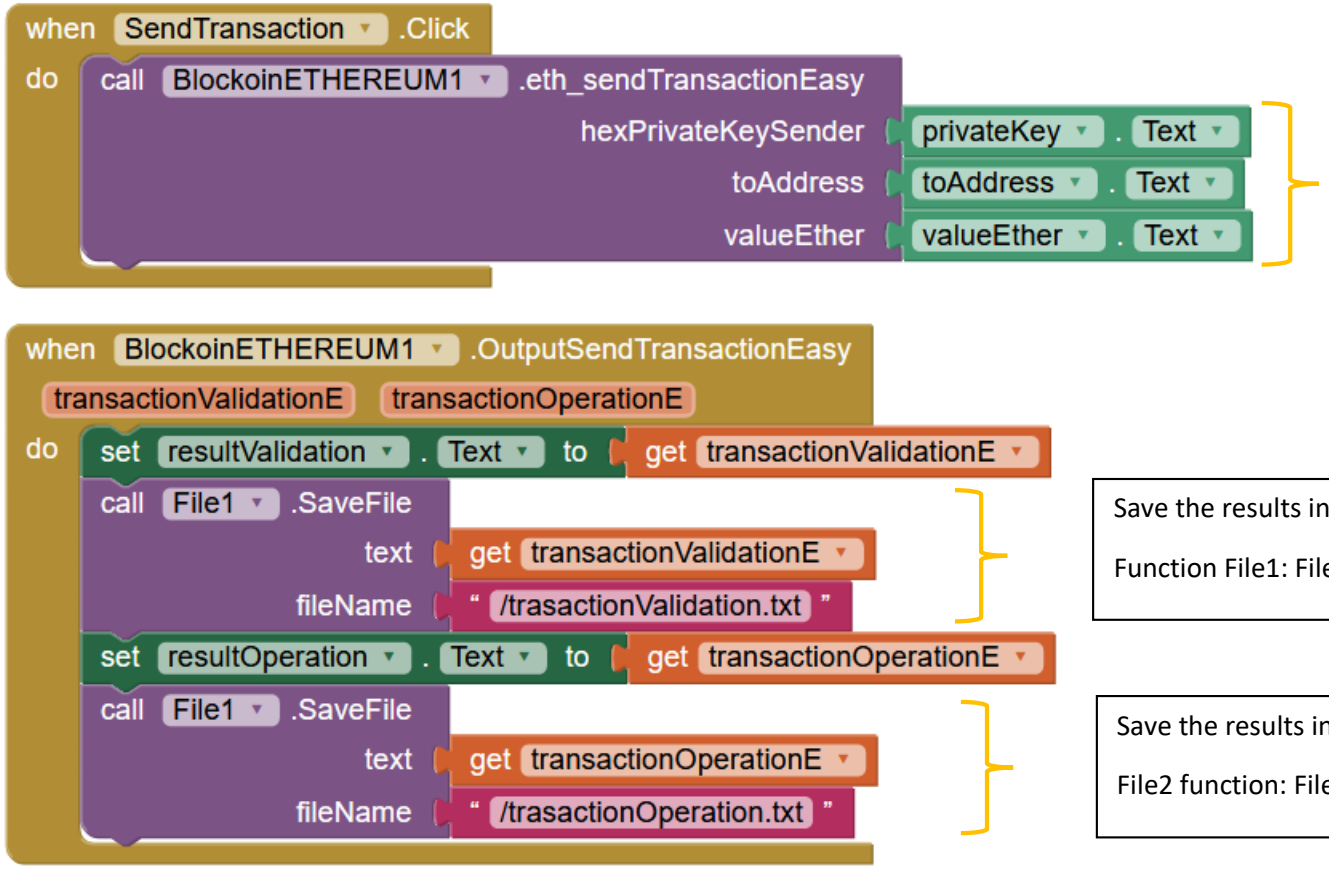
Transaction to publish and/or execute a Smart contract: $15 USD + cost of the Ethereum network.

13. Create your App (Exchange) for Android in 15 minutes.

Design in App Inventor (Screen). - **5 minutes.**



Function (**eth_SendTransactionEasy)** and event (**OutPutSendTransactionEasy**) blocks. - **5 minutes**



Input data:

PrivateKey: Primary key to the sender's address.

toAddress: Hexadecimal address of the receiver.

valueEther: Give the amount of Ether that will be sent.

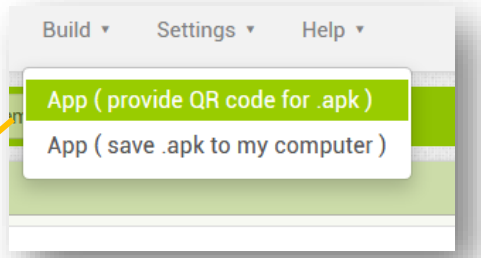
Save the results in text files:

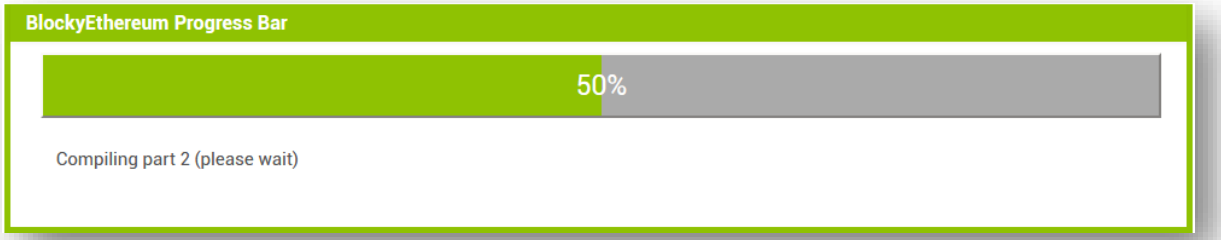
Function File1: File **trasactionValidation.txt**

Save the results in text files:

File2 function: File **trasactionValidation.txt**

We compile, generate APK file to install it on the Android device. - 5 minutes





NOTE: When the transaction is executed it will take approximately 6 to 8 seconds to release the "Send Transaction" button. Due to the connection time with the Ethereum network.

Token COINsolidation.

The Token COINsolidation is a project with three main guidelines

Imagine having your own crypto asset (cryptocurrency) with COINsolidation you can it. The first is to create the first network of extensions based on the visual programming methodology Blockly, in which by its easy and intuitive use can be used by anyone without previous knowledge of programming, the extensions that can be consulted in our Roadmap (White Paper) are directed to two fundamental sectors in the world economy, sector of crypto-currencies and / or tokens and the sector of currencies (fiat) or common use worldwide as it can be the dollar USD, Euros EU, Pounds or any other currency of current use.

The second guideline in the CoinSolidation project is to create a new algorithm to consolidate addresses of current and future cryptomontages. We are developing a new model algorithm to create an address that consolidates different addresses from different blockchains into a universal address see our (White Paper) at www.Coinsolidation.org or https://github.com/coinsolidation/whitepaper

The third guideline is to apply Quantum Computing technology in the security of the CoinSolidation environment, this will be applied with the already developed extensions of QRNG (Quantum Random Number Generator) and PQC (Post-Quantum Computing) security algorithms. These can be found in the official extensions repository on the Github site, https://github.com/coinsolidation.

General features of the CryptoToken CoinSolidation

Name: COINsolidation.
Symbol: CUAG
Launching Country: Estonia
Official website: www.Coinsolidation.org
Company: Coinsolidation International.
Launch date: April 31, 2021
Created by: Guillermo Vidal.
Consensus Algorithm: Proof of Quantum
Address algorithm: Consolidated Universal Address.

14. Licensing and use of software.

Licensing, terms and conditions of use see www.coinsolidation.org or write to info@coinsolidation.org