



配置与管理。

Ethereum Exchange Extension (EEE)。

用户指南

1.0.0测试版

2020年11月。

Blockoin.org是Bankoin Inc的注册商标，在自由使用和商业许可下。使用条款和条件见：www.CoinSolidation.org

內容

1. 介绍 :	3
2. 什么是Blockly编程 ?	5
3. 什么是延伸 ?	5
4. 什么是BlockoinEthereum和BlockoinINFURA ?	5
5. 在Ethereum平台中应用的基本概念。	6
6. 安装和配置扩展和Ethereum测试环境。	13
7. 块的定义和使用(通用功能).....	21
8. 交流Ethereum扩展 (EEE) 的功能和事件。	22
9. 创建CryptoToken或Cryptomoney代币的步骤。	34
10. 如何把一个新的资产或你的加密代币出售 (代币ERC20) 。	36
11. 标准交易成本和智能合约交易的计算方法.....	61
12. Coinsolidation.org费用	65
13. 在15分钟内创建你的Android应用程序 (Exchange) 。	66
14. Token CoinSolidation。	69
15. 软件的许可和使用 ;	70

1. 介绍：

Ethereum是一个开源平台，去中心化，不像其他区块链，Ethereum可以做得更多。它是可编程的，这意味着开发人员可以使用它来创建新类型的应用程序。其数字货币被称为"以太坊"。

这些去中心化的应用（或"dapp"）得到了加密技术和区块链技术的好处。它们是可靠和可预测的，这意味着一旦它们被"加载"到Ethereum中，它们将始终按计划运行。他们可以控制数字资产，创造新型的金融应用。它们可以是分散的，这意味着没有一个实体或个人控制它们。

眼下，全世界成千上万的开发者正在Ethereum上创建应用，并发明了新的应用类型，其中许多应用你今天就可以使用。

- 允许你用ETH或其他资产进行廉价、即时支付的加密货币组合
- 允许您借贷或投资您的数字资产的金融应用。
- 去中心化的市场，允许你交换数字资产，甚至交换对现实世界事件的"预测"。
- 游戏中，你在游戏中拥有资产，甚至可以赢得真金白银。

乙醚是如何工作的？

以太坊和其他加密货币一样，使用共享的数字账本，所有交易都会被记录下来。它是公开的，完全透明的，事后也很难修改。

这就是所谓的**区块链**，它是通过**挖矿**过程建立起来的。

矿工负责验证以太坊交易组形成"区块"，并通过解决复杂的算法进行编码。这些算法或多或少都会有一定的难度，以此来保持区块处理时间的一致性（大约每14秒一个）。

然后，新的区块会与之前的区块链相连，相关矿工会获得**奖励**，即固定数量的乙醚代币。正常情况下，这是5个以太单位，不过如果加密货币继续上涨，可以看出这个数字是可变的。

Ethereum是如何工作的？

Ethereum区块链与比特币区块链非常相似，但它的编程语言允许开发者创建软件，通过这些软件来管理交易并自动实现某些结果。这个软件被称为**智能合约**。

如果说传统的合同描述了关系的条款，那么智能合同则通过在代码中编写这些条款来确保这些条款得到满足。这些程序一旦满足预定义的条件，就会自动执行合同，消除了手动执行协议时的延迟和成本。

举个简单的例子，一个Ethereum用户可以创建一个智能合约，在某个日期向朋友发送一定数量的乙醚。他们会在区块字符串中写下这段代码，当合同完成后（即达到约定日期时），乙醚会自动发送。

这一基本思路可以应用于更复杂的配置，其潜力可能是无限的，在保险、房地产、金融服务、法律服务和小额贷款等行业，已经有一些项目取得了显著进展。

智能合约还有一些额外的好处。

1. 它们消除了中间商的身影，为用户提供了全面的控制，并将额外的成本降到最低。
2. 它们在公共区块链中被注册、加密和复制，所有用户都可以看到市场活动情况
3. 消除人工流程所需的时间和精力。

当然，智能合约还是一个很新的系统，还有很多细节需要打磨。代码是按字面意思翻译的，所以在创建合同过程中的任何错误都可能导致无法改变的不必要的结果。

DApps与智能合约

智能合约与DApps(去中心化应用)有相似之处，但也有一些重要的区别。

和智能合约一样，DApp也是一个接口，它通过去中心化的对等网络将用户与提供商的服务连接起来。但是，智能合约的创建需要固定的参与者数量，而DApps则没有用户数量的限制。此外，它们不仅仅局限于金融应用，如智能合约：一个DApp可以服务于任何你能想到的目的。

2. 什么是Blockly编程？

Blockly是一种基于JavaScript编程语言的**可视化编程方法论**，它由一组简单的命令组成，我们可以将它们组合起来，就像拼图的碎片一样。对于那些想以直观、简单的方式**学习编程**的人来说，或者对于那些已经知道如何编程并想看到这种类型的编程潜力的人来说，这是一个非常有用的工具。

Blockly是一种不需要任何计算机语言背景的编程形式，这是因为它只是将图形块连接起来，就像我们玩乐高或拼图一样，你只需要具备一些逻辑性，就可以了！

任何人都可以为手机（智能手机）创建程序，不用再去搞那些难懂的编程语言，只需用图形化的方式把积木拼接起来，以简单、方便、快捷的方式进行创建。

3. 什么是延伸？

扩展名是指用Java编程语言制作的一个模块，其文件的扩展名为.AIX。

在Blockoin.org项目中，我们基于为金融领域创建用户友好的扩展，他们可以在几分钟内制作移动应用程序，而无需大量的编程知识。

4. 什么是BlockoinEthereum和BlockoinINFURA？

BlockoinEthereum和BlockoinINFURA是免费使用的软件(扩展)，包括以下技术方案(算法)，能够创建在Ethereum的网络中使用。

- 在以太坊平台上建立新的账户。
- 资产负债表咨询、资产转移(CRYPTOMONEDA-ETHER和TOKENS)

- 创造新的ERC20标记和CRYPTOMONEDA-TOKEN。
- 智能合同的汇编、创建、咨询和出版。
- 建立线下和线上交易。
- 主键和公共键的加载。
- 汇率和加油站咨询。
- 发展、测试和EREUM核心网络环境(网络)
- 包括39个INFURA功能。

5. 在Ethereum平台中应用的基本概念。

什么是区块链？

区块链通常与比特币和其他加密货币相关联，但这些只是冰山一角，因为它不仅用于数字货币，而且可以用于任何可能对用户和/或公司有价值的信息。这项技术起源于1991年，当时Stuart Haber和W.Scott Stornetta描述了第一个关于密码学安全区块链的工作，直到2008年，随着比特币的到来，这项技术才被人们所关注。但目前其在其他商业应用中的使用需求，预计在中期的未来，在一些市场，如金融机构或物联网物联网等行业中，其使用量会有所增长。

区块链这个词比较好理解，它是指分布在在网络中多个节点（PC、智能手机、平板电脑等电子设备）上的单一的、约定俗成的记录。就加密货币而言，我们可以把它看作是记录每一笔交易的会计账本。

如果我们去了解它的内部实现细节，它的操作可能会很复杂，但基本思路很简单。

它被存储在每个块中。

1.- 有效记录或交易的数量；

2. 有关该区块的资料；

因此，每个区块在链中都有特定的、不可移动的位置，因为每个区块都包含前一个区块的哈希信息。整条链存储在组成区块链的每个网络节点上，所以所有网络参与者身上都存储着链的精确副本。

区块链Ethereum平台内的地址或账户是什么？

它是Ethereum平台中42个字符的字符串，代表一个以十六进制为基础的数字，Ethereum中定义的资产将被存入或发送。在其他区块链平台中，账户或地址的字符数可以不同，例如。

0x5d2Acdb34c279Aa6d1e94a77F7b18aB938BFb2bB

氪金是什么？

它是一种数字或虚拟货币，旨在作为一种交易媒介。它使用密码学（数字安全）来保护和验证交易，以及控制特定密码货币的新单位的创建。

什么是以太？

以太坊是Ethereum区块链平台的原生数字货币，是2013年开发的去中心化平台。以太是一个单位，可以分成更小的单位，称为WEI，具有以下等价关系。

1以太=1,000,000,000,000,000,000魏。(在数学表达上是 10^{18})。

使用以太坊时处理哪些单位？

1	ether
10^3	finney
10^6	szabo
10^9	Shannon or GWEI this is use for the GasPrice.

10^{12}	babbage
10^{15}	lovelace
10^{18}	wei

什么是令牌？

代币是可以在特定项目生态系统中使用的数字资产。

代币和加密货币的主要区别在于，前者需要另一个区块链平台（不是自己的）来操作。Ethereum是最常见的代币创建平台，主要是因为其智能合约功能。在Ethereum区块链上创建的代币一般被称为ERC-20代币，尽管还有其他更专业的代币类型，如ERC-721代币，主要用于收藏资产（卡片，在视频游戏中使用，艺术品等）。

什么是天然气？

气是指在Ethereum网络上执行一个操作或一组操作的成本。这些操作可以有好几种：从进行交易到执行智能合约或创建一个去中心化的应用程序。

换句话说，更简单的说，Gas是衡量Ethereum中工作的单位。

和物理世界一样，在Ethereum中，也有一些工作的成本比其他工作高：如果我们想要执行的操作需要组成平台的节点更多的使用资源，这将使Gas也增加，反之亦然。

气的作用主要是在Ethereum平台上实现三个功能。

1.-为任务的执行分配成本；在Ethereum中，执行任务也有成本。**根据任务的难度或我们希望该任务的处理速度，该操作的计算成本会相应提高或降低**，Gas的数量也会按比例增加或减少。

2、-保障系统安全；Ethereum系统是一个安全的系统，这主要得益于Gas。

通过要求对每笔交易的执行支付佣金，该平台确保网络上不会处理任何无法使用的交易。这有助于使区块链变得更轻，因为它不会向区块链添加大量无用的兆字节信息。

此外，通过Gas，系统还可以防止"垃圾邮件"和循环的无限使用：通过代码执行重复任务的指令。

例如，如果"气"不存在，就没有什么能阻止任务无限重复，使系统崩溃，无法使用。

3、奖励矿工；矿工是分布在世界各地的独立外部系统，负责执行Ethereum平台内的交易。当我们进行交易或执行智能合约时，我们会"支付"一定的Gas。

这个Gas的作用是"支付"矿工们所使用的资源（硬件、电力和时间），同时也增加了对他们工作的**奖励**。因此，我们可以说，"气"也有助于维持平台的平衡。

我们谈论"支付"天然气--用引号表示--因为那是运营成本。换句话说，你为Ethereum处理交易所花费的费用"买单"。

什么是汽油价格？

气单元对应于一个指令的执行，如计算机步骤。

那么，矿工们是如何将获得的天然气作为奖励"变现"的呢？

气体本身不值钱，因此不能充电。要想对这些用过的天然气"收费"，就必须给这些消耗的资源赋予价值，也就是给矿工的工作赋予货币价值。在交易或智能合约中使用的Gas数量有一个等价的以太币价格。这个价格被称为"天然气价格"，它通常由矿工分配，并根据Ethereum区块链的繁忙程度而变化。通常在GWEI单位处理。

什么是气体限制？

此数据表示一个交易可以消耗的最大气体值才有效。

通常情况下，用于在Ethereum网络上进行交易的软件会自动计算出进行交易所需的Gas量的估计值，并立即显示给我们。

什么是加油站？

它是矿工处理的值被参考的地方，以便以协商一致的方式决定哪一个是在Ethereum区块链中进行不同类型交易所需的GasPrice。

根据选择GasPrice的多少，会对交易执行的优先时间进行加权，通常处理三种GasPrice：TRADER：ASAP，FAST<2分钟，STANDARD<5分钟。这些都是平均值，可能会根据Ethereum主网络的工作负荷（tractions）而改变时间。

<https://ethgasstation.info/>

什么是交易所？

氪金交易所是指氪金者交换货币或其他氪金者的聚集点。在这些在线交易所中，市场价格的产生标志着基于供求关系的加密货币的价值。

什么是汇率？

这些是以太币在各国货币中的价值率。例如，在本手册编写之日，以太币的美元价值为430.94美元。

什么是智能合约？

在Ethereum中，智能合约是一个程序，在编程语言中称为Solidity，是在区块链Ethereum中，它有指令，根据预先建立的规则自动执行，这个属性使得所有现有的区块链的进化，因为你可以创建许多智能合约，适应不同的私人和公共部门，使公司更有效的操作，或在适当的地方适应各种系统或程序。

什么是"nonce"？

它是一个增量的"十六进制数"计数器，可以跟踪Ethereum中每个方向的交易或创建的账户。

什么是交易？

它是指在Ethereum系统内执行或转让某种类型的非有形资产，可以在Ethereum系统内被赋予一个预先设定的价值，并且可以在以后改变为公司或个人的有形价值。

什么是txHash？

它是一个十六进制数字，有助于跟踪每笔交易的详细结果。

有哪些交易类型？

你有两种类型，一个是交易"离线"，这创建不需要有连接到Ethereum的主网络可以存储，直到你选择连接到Ethereum的网络，并释放交易，有安全的优势，因为整个交易是离线处理，防止任何异常，可能是在网络连接。另一种交易是"在线"交易，它总是需要与互联网连接，连接带来的安全优势和劣势。

什么是INFURA.io？

Infura.io是一个平台，它提供了一套工具和基础设施，允许开发人员轻松地将他们的应用区块链从测试，到扩展，简单可靠地访问Ethereum和IPFS。

Ethereum网络上的地址是什么？

一个地址或账户由三部分组成，地址、公钥和私钥，这两个密钥是一串十六进制格式的数字和字符，用于发送和接收（主动）或乙醚（数字货币）。

主钥匙绝对不能与任何人共享，因为它是授权释放账户中的余额（签署交易）的钥匙。

公钥是整个公众都知道的，而且是与任何人共享的，因为它是确认交易价值和发送对象正确的参考。

例如：

```
{  
  "private": "429a043ea6393b358d3542ff2aab9338b9c0ed928e35ec0aed630b93adb14a1c",  
  "public":  
    "049b4b7e72701a09d3ee09165bba460f2549494a9d9fd7a95aaac57c2827eac162fd9e105b  
    2461cd6594ca8ca6a8daf10fe982f918be1b0060c87db9cfbcd289a8",  
  "address": "88ab6dcecc3603c7042f4334fc06db8e8d7062d5"  
}
```

什么是Coinolidation.org ?

它是一个整合加密货币的平台，我们在中心化和去中心化的金融世界中创建了第一个混合地址，使用Blockly技术，这是一种可视化的编程形式，不需要基于功能扩展的高级编程知识。参见我们的白皮书：www.coinsolidation.org

区块链Ethereum中的测试网络和主网有哪些类型？

<https://mainnet.infura.io>

主网、生产网，所有交易都是实时进行的。

<https://ropsten.infura.io>

Ropsten测试网络允许区块链开发在真实环境中测试他们的工作，但不需要真正的ETH代币，其算法称为（工作证明）。

<https://kovan.infura.io>

科凡测试网，与前者不同的是ropsten使用的是一种名为权威证明的算法。

<https://rinkeby.infura.io>

测试网络，使用了一种叫做权威证明的算法。检验最常用的一种。

<https://goerli.infura.io>

Goerli是Ethereum的一个公共测试网络，POA（权威证明）共识引擎支持与各种客户端。防垃圾邮件。

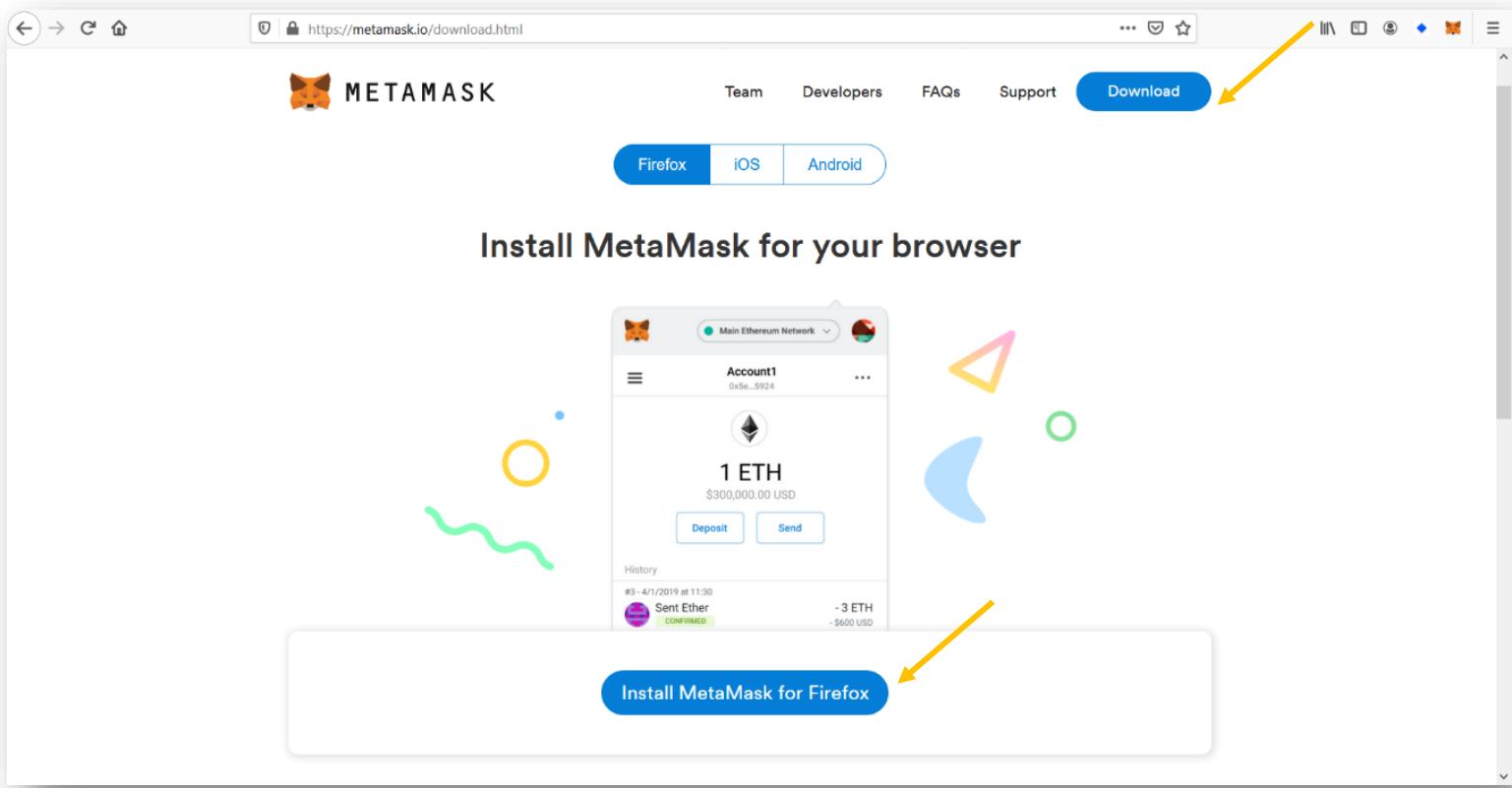
基于区块链Ethereum的网络总共有5个，一个用于生产或主网，四个用于测试，那么我们将使用Rinkeby的网络来安装我们的测试环境。

6. 安装和配置扩展和Ethereum测试环境。

我们首先需要一个测试环境。要学习如何使用这两个扩展的不同功能，这两个扩展将用于创建、发送、发布、审查和获取我们在Ethereum平台上进行的所有交易的数据。

首先我们要做的是设置我们的测试环境。我们将不得不安装**METAMASK**应用程序。这是一个钱包，以创建，导入Ethereum帐户和管理交易从我们的互联网浏览器，我们将使用的是Mozilla，也可以在Chrome中使用。

进入官网[www.metamask.io](https://metamask.io)，点击"下载"按钮。



然后点击"安装MetaMask for Firefox"按钮，接受默认选项。安装完成后，我们会在右上方看到一个图标，在这里我们会看到已经安装好的Metamask软件。

点击Metamask图标，会出现导入现有账户或创建新账户的选项。在我们的例子中，我们将使用"通过种子恢复"的方法导入一个已经运行的账户。如果您没有，那么只需创建一个新的帐户。在这种情况下，我们会被要求在两种情况下分配一个密码。

后面我们用"密码"输入，就可以查看我们的余额是多少，从逻辑上讲如果是新的余额就会是空的。

The screenshot shows a web browser window with the URL <https://www.google.com> in the address bar. The main content is the Google search page. Overlaid on the right side is a wallet extension interface for Ethereum. The top part of the extension shows a profile picture of a fox, a dropdown menu set to "Ethereum Mainnet", and a balance of "0.0719 ETH" worth "\$31.29 USD". Below this are three buttons: "Buy", "Send", and "Swap". There are two tabs at the bottom: "Assets" (selected) and "Activity". A blue button labeled "Add Token" is visible. A yellow arrow points from the text above to the "Ethereum Mainnet" dropdown in the extension.

我们现在继续回顾一下我们如何将一些以太币（数字货币）存入我们的Rinkeby测试账户。

在顶部，我们可以选择我们将使用的网络类型，默认情况下，当您输入时，它将您放置在"Ethereum Mainnet"中，然而，当您点击时，我们可以审查所有我们可以选择的可选网络，在我们的情况下，我们选择了Rinkeby网络。

The screenshot shows a web browser window with the URL <https://www.google.com> in the address bar. The main content is the Google search page. Overlaid on the right side is a wallet extension interface for Ethereum. The top part of the extension shows a profile picture of a fox, a dropdown menu set to "Rinkeby Test Network", and a balance of "1063.5384 ETH" (Ropsten Test Network). Below this is a list of networks: Ethereum Mainnet (selected), Ropsten Test Network, Kovan Test Network, Rinkeby Test Network (checked), and Goerli Test Network. A yellow arrow points from the text above to the "Rinkeby Test Network" option in the dropdown menu.

通过应用程序"Termux"搜索，选择它并开始安装过程。 

下一步是将乙醚存入我们的Rinkery网络参考测试账户。

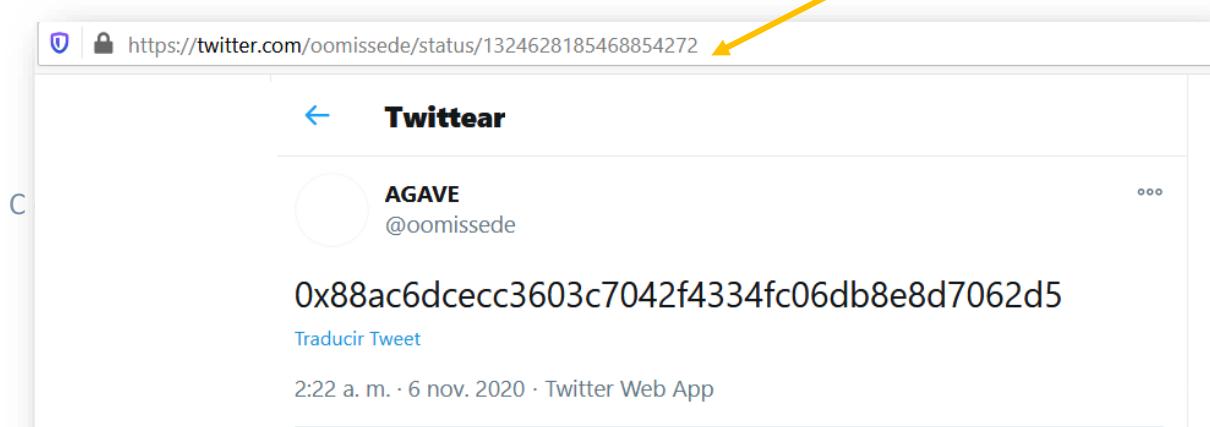
重要提示：我们存入账户的以太币（数字货币）参考Rinkery测试网络，在加密市场上没有任何价值，它们只会被我们用于软件测试。一定要检查我们在哪个网络上工作，避免交易出现错误。

为了得到测试用的乙醚，我们需要执行以下程序。

在你的任何twitter帐户，我们将不得不进入twitter和创建一个评论，只包括帐户和/或地址，我们在Metamask中，然后我们将不得不复制评论的链接，因为我们有这个我们将去下一个链接锚定我们的帐户。

<https://faucet.rinkeby.io/>

我们创建了一个微博评论，并复制了评论中的链接。 



The screenshot shows a Twitter post from the user AGAVE (@oomissede). The post contains a single line of text: "0x88ac6dcecc3603c7042f4334fc06db8e8d7062d5". Below the tweet, there are options to "Traducir Tweet" and "2:22 a. m. · 6 nov. 2020 · Twitter Web App". Above the tweet, the browser's address bar displays the URL "https://twitter.com/oomissede/status/1324628185468854272". A yellow arrow points from the text above to the URL in the address bar.

在网站<https://faucet.rinkeby.io/>, 我们复制twitter的链接，并在“给我以太”按钮上选择所需金额。

The screenshot shows two consecutive screenshots of the Rinkeby Authenticated Faucet website.

Top Screenshot: The page title is "Rinkeby Authenticated Faucet". A URL input field contains "https://twitter.com/oomissede/status/1324626802388750337". To the right is a button labeled "Give me Ether ▾". A dropdown menu shows three options: "3 Ethers / 8 hours", "7.5 Ethers / 1 day", and "18.75 Ethers / 3 days". Three yellow arrows point from the text "How does this work?" and the explanatory text below it to the dropdown menu. Another yellow arrow points from the "Give me Ether" button to the "18.75 Ethers / 3 days" option.

Bottom Screenshot: The page title is "Rinkeby Authenticated Faucet". The URL input field still shows "https://twitter.com/oomissede/status/1324626802388750337". The "Give me Ether" button is now highlighted. A green box at the top displays the message "Funding request accepted for oomissede@twitter into 0x9ec478ee4982489e946071a8d62d90b39260". A yellow arrow points from this message to the "Give me Ether" button. Below this, a message box shows "0x9ec478ee4982489e946071a8d62d90b39260a36 a few seconds ago" and "5 peers 7499496 blocks 9.046256971665328e+56 Ethers 383581 funded".

现在我们的账户上有了Ethers，我们可以开始在Ethereum平台上进行测试了。

我们有两个扩展来与Ethereum区块链进行交互。

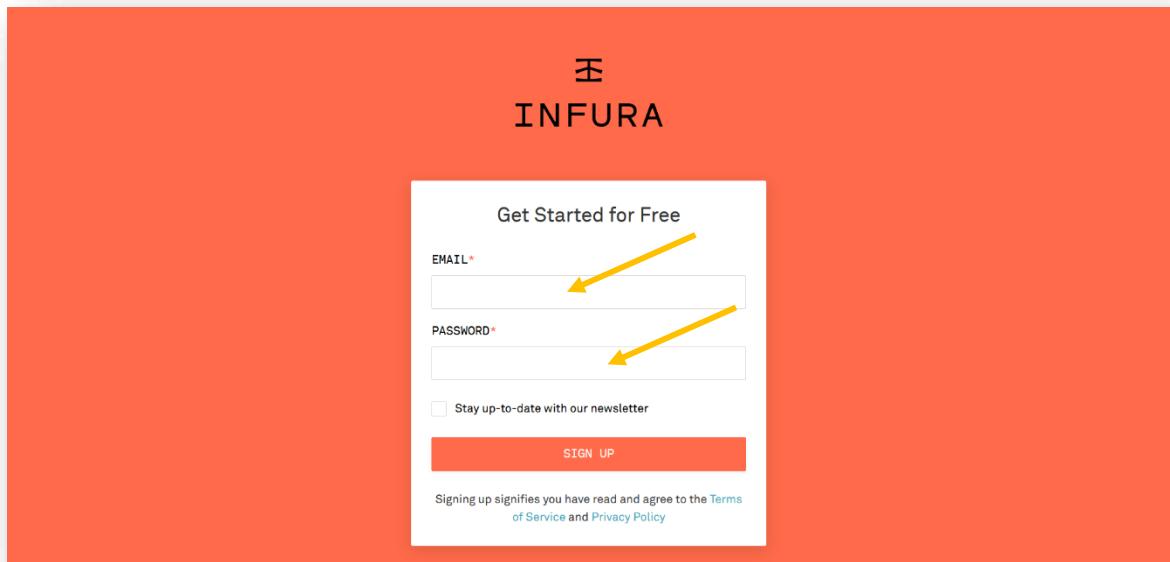
共固的延伸。 BlockoinETHEREUM.aix包含执行以下功能的函数。

- 在区块链Ethereum中创建新的账户（地址）（privateKey, publicKey, 地址）。
- 在二进制文件中存储账户数据（地址）。
- 导入二进制文件账户(地址)
- 通过privateKey获取账户（地址）。
- 在账户（地址）之间"在线"建立和发送交易；
- 创建、签署和发送离线PushRaw交易。
- 咨询交易细节Tx。
- 对地址和智能合约进行余额查询。
- 智能合约的编译器。
- 智能合约的创建、发布和执行。
- 创建、发布和执行代币ERC20（加密货币代币）。
- 从智能合约中获取ABI代码。
- 验证网络连接。
- 查询乙醚在世界任何国家的加密市场上的价值（该国的本币） - 汇率。
- GasPrice咨询。
- 协商具体账户的"nonce"。

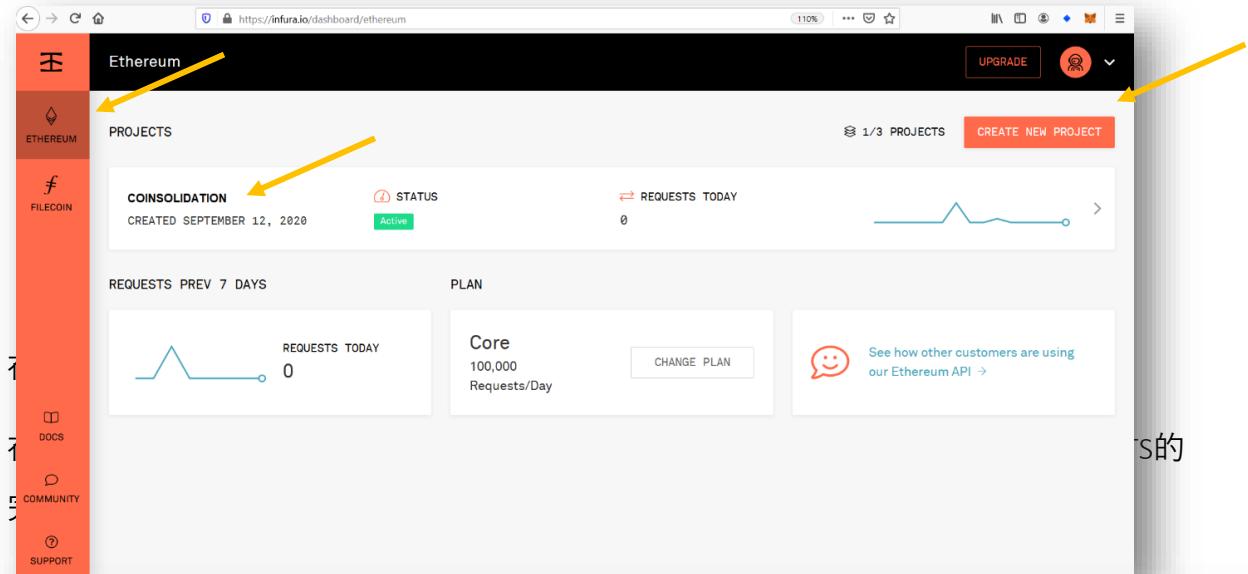
币化的延伸。 BlockoinINFURA.aix为我们提供了Infura.io平台的40个功能，详情请直接查阅js-son-rpc文档，<https://infura.io/docs>。

为了使用INFURA扩展，你需要在infura.io网站上创建一个账户，因为我们需要一个KEY API来向Ethereum网络发送查询，以及能够使用Ethereum测试网络。

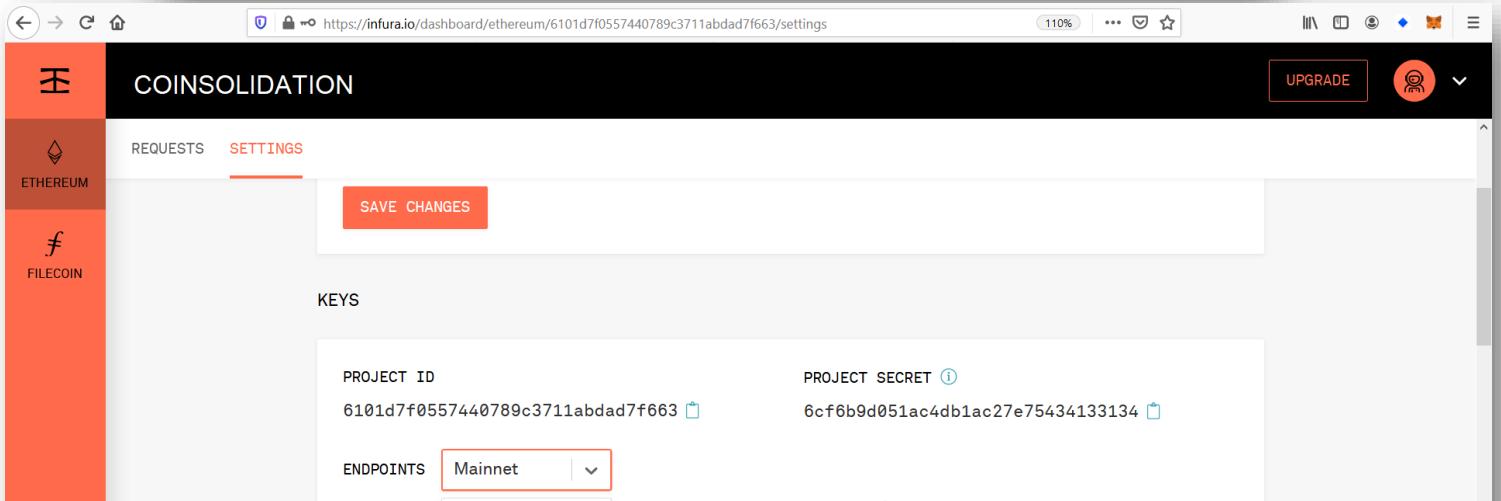
我们如何开户很简单，如下图所示。



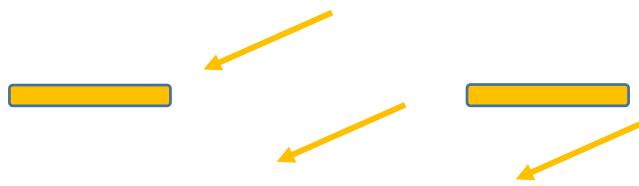
账号创建后，我们进入，就可以拥有不同网络的KEY API。我们到左上方，我们点击 Ethereum，然后我们会看到项目，我们创建一个新的项目，然后我们介绍自己的项目。



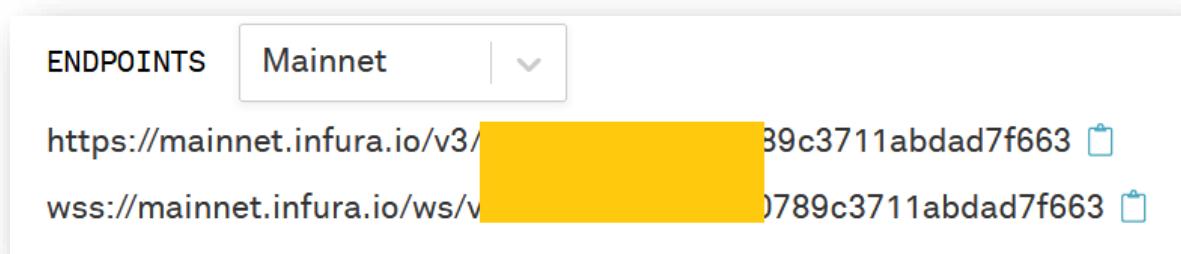
A screenshot of the Infura Ethereum dashboard. On the left, there's a sidebar with icons for Ethereum (selected), Filecoin, Docs, Community, and Support. The main area shows a project named 'COINSOLIDATION' created on September 12, 2020. It has an 'Active' status and 0 requests today. There are graphs for requests over the previous 7 days and today. A 'PLAN' section shows 'Core 100,000 Requests/Day'. A 'CREATE NEW PROJECT' button is in the top right. Yellow arrows point from the sidebar to the Ethereum icon and from the sidebar to the 'CREATE NEW PROJECT' button.



A screenshot of the Infura project settings for 'COINSOLIDATION'. The sidebar on the left shows 'REQUESTS' (selected) and 'SETTINGS'. The main area has tabs for 'REQUESTS' and 'SETTINGS', with 'SETTINGS' selected. A 'SAVE CHANGES' button is visible. In the 'KEYS' section, it shows 'PROJECT ID' (6101d7f0557440789c3711abdad7f663) and 'PROJECT SECRET' (6cf6b9d051ac4db1ac27e75434133134). An 'ENDPOINTS' dropdown is set to 'Mainnet'. Yellow arrows point from the sidebar to the REQUESTS tab and from the sidebar to the 'SETTINGS' tab.



例如，要使用Ethereum的主网络，我们将采取以下联盟。

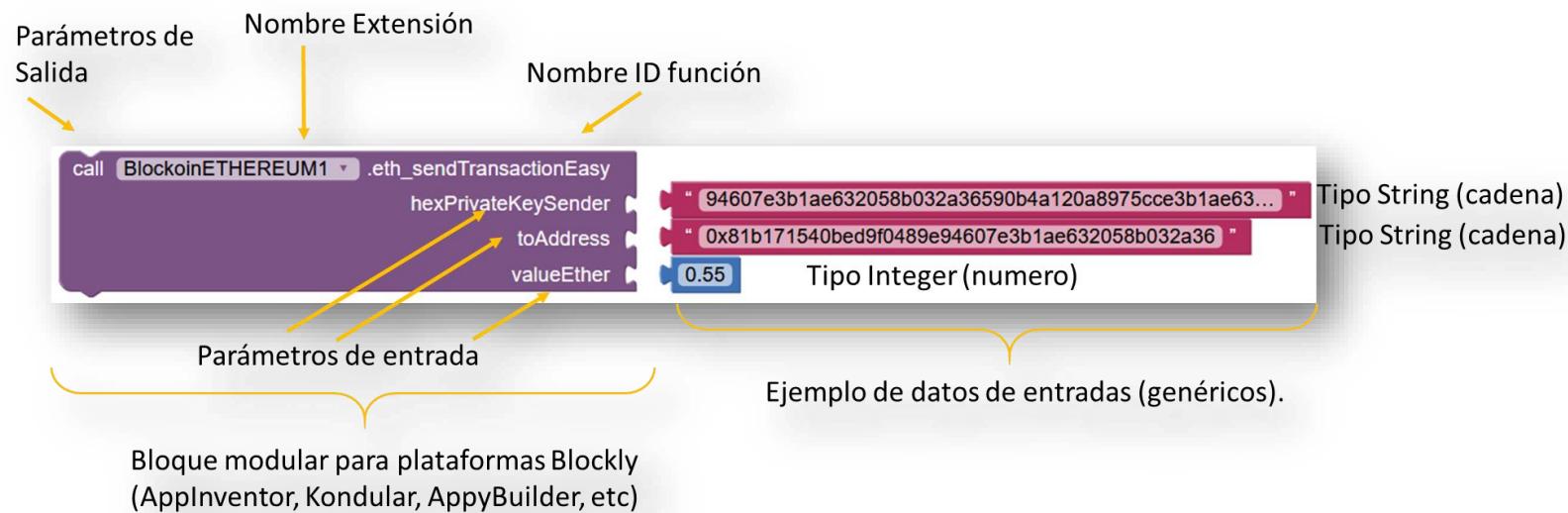


注意：该扩展已经在AppInventor、Kondular、Thunkable和AppyBuilder系统上进行了测试。

7. 块的定义和使用(通用功能)

我们先来解释一下所有区块会有的数据分布，它们的使用语法和配置。

在下面的例子中，我们可以看到一个模块块及其输入和输出参数，以及输入数据的类型，这些数据的类型可以是String（字符串）或Integer（整数或十进制）。我们将展示如何使用它，并配置它以使其正常运行。



每一个模块块都会有它的描述，并会被命名，如果它有任何强制性或可选性的依赖其他模块作为输入参数，集成过程将被公布

8. 交流Ethereum扩展 (EEE) 的功能和事件。

***测试网络，我们将使用**Rinkeby**，作为urlNetwork，当你想进行真正的交易时，你只需要改变urlNetwork网络为**mainnet**。

阻止检查互联网连接-- (CheckInternetConnection) 。

call BlockoinETHEREUM1 .CheckInternetConnection

输入参数：不适用。

输出参数：如果有连接则返回"True"，如果没有连接则返回"False"。

说明：阻止检查互联网连接和发送数据（交易）。

生成一个新的"离线"地址的区块--(GenerateNewAddressEthereum)

call BlockoinETHEREUM1 .GenerateNewAddressEthereum
phraseHex " exchange ethereum extension for systems blockly "

输入参数：phrasaHex <字符串>。

输出参数：事件 (OutputGenerateNewAddressEthereum) 。

输出。PrivateKey<String>, PublicKey<String>, addressEtehreum<String>.

when BlockoinETHEREUM1 .OutputGenerateNewAddressEthereum
privKeyEther pubKeyEther addressEthereum
do

描述：根据短语或数字序列创建一个新的以太坊地址（账户）。新地址可以在没有网络或互联网连接的情况下创建--"离线"。

块生成一个新的"在线"地址--（GenerateNewAddressEthereum）。

call BlockoinETHEREUM1 .GenerateNewAddressEthereumAPI

输入参数：不适用。

输出参数：以JSON数据格式返回；privateKey、publicKey、地址。

产出示例：

```
{  
    "private": "9ab4b2fba728a55643414f26adc04eea080740860cbe39b13fe4acb43dbb9f83",  
    "public":  
        "0421d559aa98d6fc77688ae9f19b3dc502c05f0b7f70bbe924cb712d6243a489695b1c1ee03993b3b6d97277  
        97e780677a5469800b4d98374bdb910ed99fa2b5c8",  
    "address": "92a2f157d5aec3fa79f92995fea148616d82c5ef"  
}
```

描述：创建一个新的以太币地址（账户）。由于生成是通过REST API服务--"在线"进行的，所以必须要有互联网的访问或连接。

块生成一个新的"离线"地址，并将公钥和私钥保存在二进制文件中（GenerateNewAddressEthereumStoreKeys）

call BlockoinETHEREUM1 .GenerateNewAddressEthereumStoreKeys

pathFilePrivateKey

“ /mnt/sdcard/privateKey.bin ”

pathFilePublicKey

“ /mnt/sdcard/publicKey.bin ”

输入参数：pathFilePrivateKey<String>, pathFilePublicKey<String>。

输出参数：事件（OutputGenerateNewAddressEthereumStoreKeys）。

输出：addressEthereum<String> , privateKeyECC<String> , publicKeyECC<String> ,
privateKeyHex<String> , publicKeyHex<String>。

```
when BlockoinETHEREUM1 .OutputGenerateNewAddressEthereumStoreKeys  
    addressEthereum privateKeyECC publicKeyECC privateKeyHex publicKeyHex  
do
```

描述：创建一个新的随机以太坊地址（账户），并将公钥和私钥存储在二进制文件中，用于导入和导出账户数据。新地址可以在没有网络或互联网连接的情况下创建--"离线"。

块生成公钥--（GeneratePublicKeyHexFromPrivateKey）。

```
call BlockoinETHEREUM1 .GeneratePublicKeyHexFromPrivateKey  
    hexPrivateKey "429a043ea6333b358d3542ff2aab9338b9c0ed928e35ec0a..."
```

输入参数：hexPrivateKey <字符串>。

输出参数：事件(OutputGeneratePublicKeyHexFromPrivateKey)

输出：地址<字符串> , publicKeyHex<字符串>。

```
when BlockoinETHEREUM1 .OutputGetAddressEthereumFromPrivateKey  
    address publicKeyHex  
do
```

说明：根据输入的私钥创建公钥。

块获取一个地址的余额--（GetBalanceAddrEthereum）。

```
call BlockoinETHEREUM1 .GetBalanceAddrEthereum  
    addressEthereum "0x92a2f157d5aec3fa79f92995fea148616d82c5ef"
```

输入参数：hexPrivateKey <字符串>。

输出参数：事件(OutputGeneratePublicKeyHexFromPrivateKeyHex)

输出：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>。

```
when BlockchainETHEREUM1 .OutputDataBalanceAddrEthereumWEI
  balance total_received total_sent unconfirmed_balance final_balance n_tx unconfirmed_n_tx final_n_tx
do
```

说明：显示资产负债表和详细的账户（地址）数据。

阻止检查你的手机是否有网络接口被激活--（GetDataNetworkConnection）。

call BlockchainETHEREUM1 .GetDataNetworkConnection

输入参数：不适用。

输出参数：事件(OutputGeneratePublicKeyHexFromPrivateKeyHex)

输出：interfacename<String> isconnected<String>。

when BlockchainETHEREUM1 .OutputGetDataNetworkConnection
interfacename isconnected
do

说明：显示使用界面的名称，并显示该界面是否被激活。

阻止签署交易"离线"--(SignerGenericPushRawTransactionOffline)

```
call BlockoinETHEREUM1 .SignerGenericPushRawTransactionOffline
    urlNetwork "https://rinkeby.infura.io/v3/440789c3... "
    hexPrivateKeySender "9eC478ee49824890b4a120a8975cce3b1ae632058b0301f8..."
    nonceNumber get global nonce
    gasPrice "250000000000"
    gasLimit 21000
    toAddress "0x92a2f157d5aec3fa79f92995fea148616d82c5ef"
    valueWei "1000000000000000000" x "0.01"
```

所需的依赖关系 : Block (eth_getTransactionCount), Block (eth_SendRawTransactionInfura)

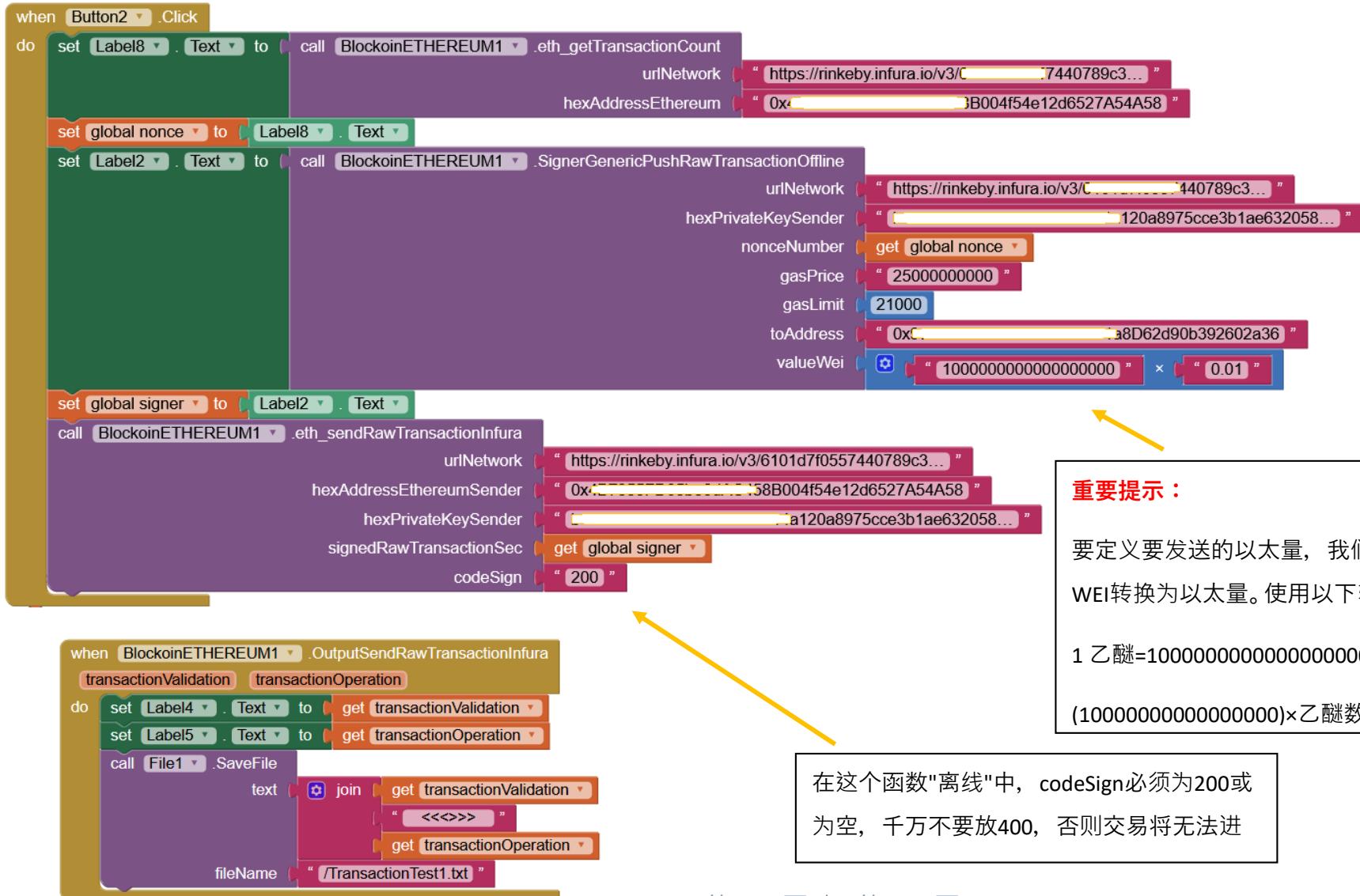
输入参数 : urlNetwork <String>, hexPrivateKeySender <String>, nonceNumber <String>, gasPrice <String>, gasLimit <Integer>, toAddress <String>, valueWEI <Integer>。

输出参数 :

输出 : 要发送的已签署的交易。<字符串>。

说明 : 准备发送一个新的交易 (加密和签名)。这可以在没有网络或互联网连接的情况下进行处理 - "离线"。

与区块依赖的完整使用示例（SignerGenericPushRawTransactionOffline）。



重要提示：

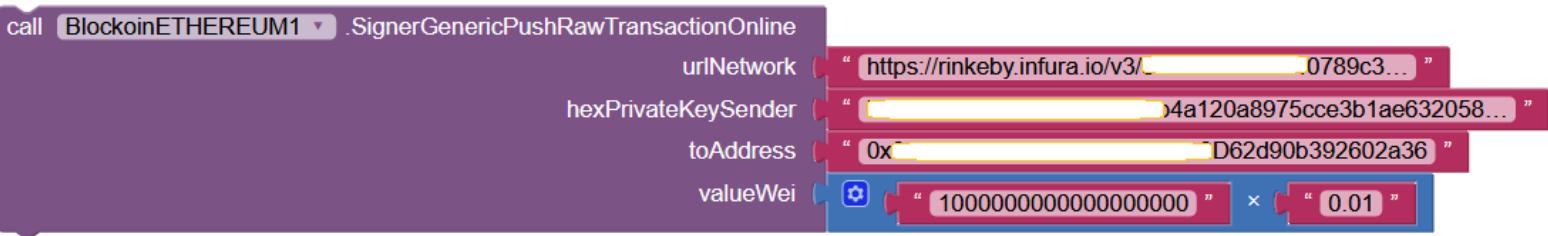
要定义要发送的以太量，我们必须首先将WEI转换为以太量。使用以下转换：

$$1 \text{ 乙醚} = 1000000000000000000 \text{ WEI} = 10^{18}$$

$(1000000000000000000) \times \text{乙醚数量}.$

在这个函数“离线”中，codeSign必须为200或为空，千万不要放400，否则交易将无法进

阻止签署"在线"交易-- (SignerGenericPushRawTransactionOnline)。

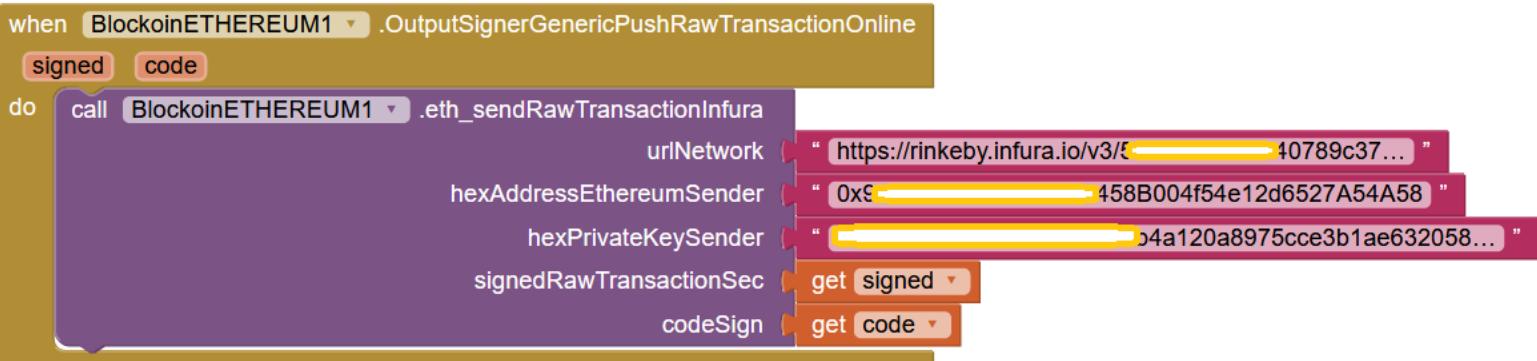


法定单位：块(eth_SendRawTransactionInfura)。

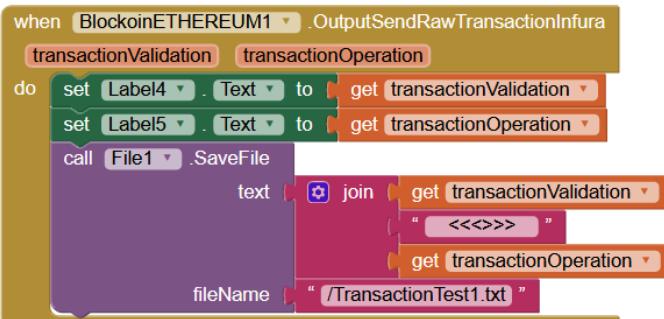
输入参数：`urlNetwork <String>`, `hexPrivateKeySender <String>`, `toAddress <String>`,
`valueWEI <Integer>`。

输出参数：按以下顺序使用的事件（OutputSignerGenericPushRawTransactionOnline）
和（OutputSendRawTransactionInfura）。

输出：signed<String>, code<String>。



输出块 eth_sendRawTransactionInfura: transactionValidation<String> ,
transactionOperation<String>。



说明：准备发送一个新的交易（加密和签名）。需要网络或互联网连接--"在线"。

用于编译智能合约"在线"的块--（SmartContractCompilerSolidity）。

```
call BlockoinETHEREUM1 .SmartContractCompilerSolidity
      nameFileSolidityJSON " /mnt/sdcard/smartcontract.json "
```

输入参数：nameFileSolidityJSON <String>。

输出参数：显示编译好的智能合约，这个功能可以帮助我们在Ethereum网络中发布智能合约之前，检查它是否写好。

输出：编译后的代码。

智能合约必须是JSON格式的文件。

用Solidity语言编写的基本智能合约示例。

```
pragma solidity ^0.5.0.

宿命约
地占有者
函数 mortal() { owner = msg.sender; }。
function kill() { if (msg.sender == owner) suicide(owner); } }。
签绿员敲的
弦词语。
function greeter(string _greeting) public { greeting = _greeting; }。
函数 greet() constant returns (string) {return greeting; }。
```

以前的智能合约的例子，是JSON格式的，有注释。

```
# 通过Solidity编译器编译
# 以"greeter"合约为例Ethereum的"hello world"。
```

文件：smartcontract.json

```
{
  "稳固性。"contract mortal {/n /*定义地址    类型的拥有者*/n address owner;/n /
  *该函数在初始时执行并设置合同的所有者    /n function mortal() { owner = msg.sender; }n /
  *函数kill同向的资金    /n function kill() { if (msg.sender == owner)
  suicide(owner); }n}n /*回始同向的资金*/n function kill() { if
  (msg.sender.sender == owner) suicide(owner); }n}/n}/n /
  *定义字符串类型的变量greeting */n      string greeting;\\"  \n/*这合同执行运行 */
  function greeter(string _greeting) public {/n      greeting =
  _greeting;\n}n /      * main function */n      function greet()
  constant returns (string) {           /n return greeting;\n}n      }/n",
  "params."["你好""测试"]
}
```

重要提示：JSON格式必须在每行末尾有一个换行符。

编译后的Smartcontract输出示例。

```
[
{
  "name": "u003cstdin003e:greeter",
  "稳固性。"contract mortal {/n /*定义地址    类型的拥有者*/n address owner;/n /
  *该函数在初始时执行并设置合同的所有者    /n function mortal() { owner = msg.sender; }n /
  *函数kill同向的资金    /n function kill() { if (msg.sender == owner)
  suicide(owner); }n}n /*回始同向的资金*/n function kill() { if (msg.sender ==
  owner) suicide(owner); }n}/n}/n      *定义字符串类型的变量greeting */n      string
  greeting;\\"  \n/*这合同执行运行 */n      function greeter(string _greeting)
  public {/n      greeting = _greeting;\n}n /      * main function
  *n      function greet() constant returns (string) {           /n return
  greeting;\n}n      }/n",
  "bin": "606060405260405161023e38038061023e83398101604052805101600080546001
60a060020a031916331790558060016000509080519060200190828054600181600116156
101000203166002900490600052602060002090601f016020900481019282601f10609f57
805160f19168380011785555b50608e9291505b8082111560cc57600081558301607d565
b50505061016e806100d06000396000f35b828001600101855582156076579182015b8281
1115607657825182600050559160200191906001019060b0565b509056606060405260e06
0020a600035046341c0e1b58114610026578063cf3ae321714610068575b005b6100246000
543373ffffffffffffffffffffffffffffff908116911614156101375760005
473fffffffffffff16ff5b6100c9600060609081526001
805460a06020601f600260001961010086881615020190941693909304928301819004028
1016040526080828152929190828280156101645780601f10610139576101008083540402
83529160200191610164565b6040518080602001828103825283818151815260200191508
0519060200190808383829060006004602084601f0104600f02600301f150905090810190
601f1680156101295780820380516001836020036101000a031916815260200191505b509
25050506
  "abi": [
}
```

```

"常数":假
"输入":[],
"名称":"杀"。
"产出": [],
"类型":"功能"
},
{
"常数":真
"输入": [],
"姓名":"问候"。
"产出": [
{
"姓名":"",
"类型":"字符串"
}
],
"类型":"功能"
},
{
"输入": [
{
"name": "greeting",
"类型": "字符串"
}
],
"类型": "建模"
}
],
"params": [
"测试"
]
},
{
"姓名":"凡人"。
"稳固生."contract mortal {/n /*定义地址    类型所有者/n address owner;/n /
*该函数在初始时执行，并置合同所有者    /n function mortal() { owner = msg.sender; }/n /
*函数返回同的资金    /n function kill() { if (msg.sender == owner)
suicide(owner); }/n /*函数返回同的资金/n function kill() { if (msg.sender == owner)
suicide(owner); }/n }/n /*定义字符串变量greeting */n     string
greeting;\n\n      /*这合同执行*/     /n function greeter(string _greeting)
public {/n         greeting = _greeting;\n\n     }/n /     * main function
/*     function greet() constant returns (string) {           /n return
greeting;\n     }/n }",
"bin":"606060405260008054600160a060020a03191633179055605c8060226000396000
f3606060405260e060020a600035046341c0e1b58114601a575b60186000543373fff
ffffffffffffffffffff90811691161415605a5760005473ffffffff
ffffffffffffffffffffffff16ff5b56",
"abi": [
{
"常数":假
"输入": [],
"名称":"杀"。
"产出": [],

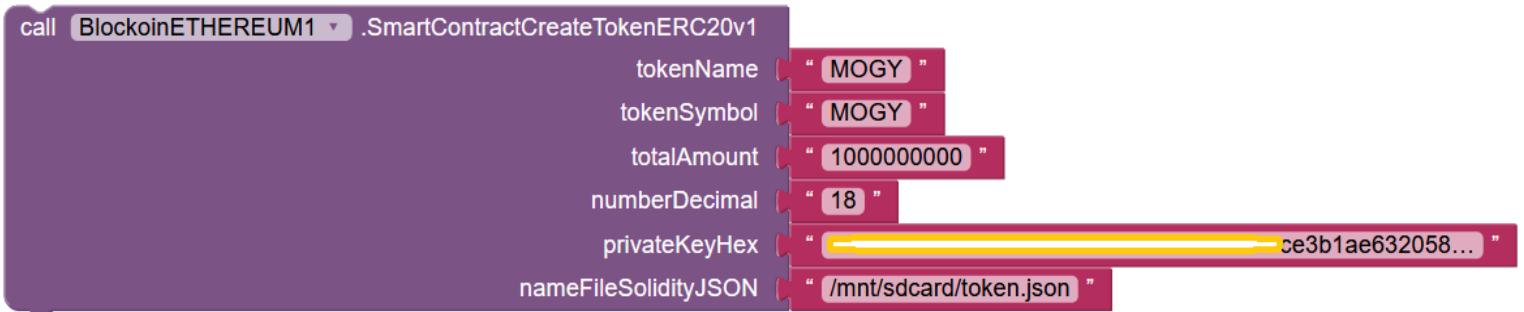
```

```

    "类型": "功能"
},
{
    "投入": [],
    "类型": "建商"
}
],
"params": [
    "你好", "测试"
]
}
]

```

编译、创建和发布ERC20令牌的区块-- (SmartContractCreateTokenERC20v1)。

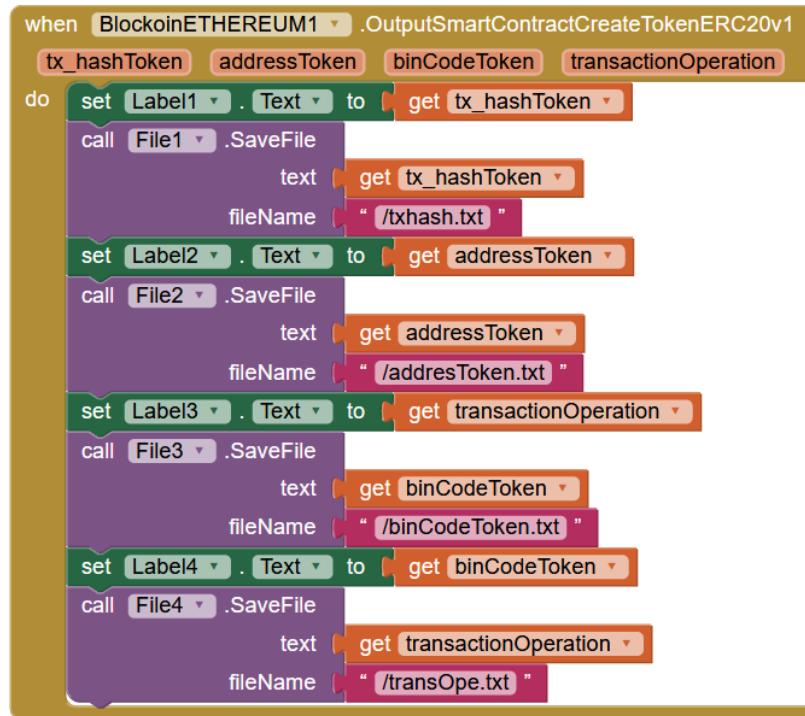


法定单位：块(CreateTestingFie)。**重要的是**：首先你必须使用这个区块来确保路径正确，这是因为如果你没有在区块中给出一个有效的路径（SmartContractCreateTokenERC20v1），令牌创建将不会被执行，因为输入参数 "nameFileSolidityJSON" 被用来创建一个临时文件。

输入参数：`tokenName <String>`, `tokenSymbol <String>`, `totalAmount <String>`, `numberDecimal <String>`, `privateKeyHex <Integer>`, `nameFileSolidityJSON <String>`**该文件是创建临时文件的有效路径**，你必须确保该路径是有效的，测试文件被创建后你可以使用Block(CreateTestingFile)使用后检查它是否被创建成功。

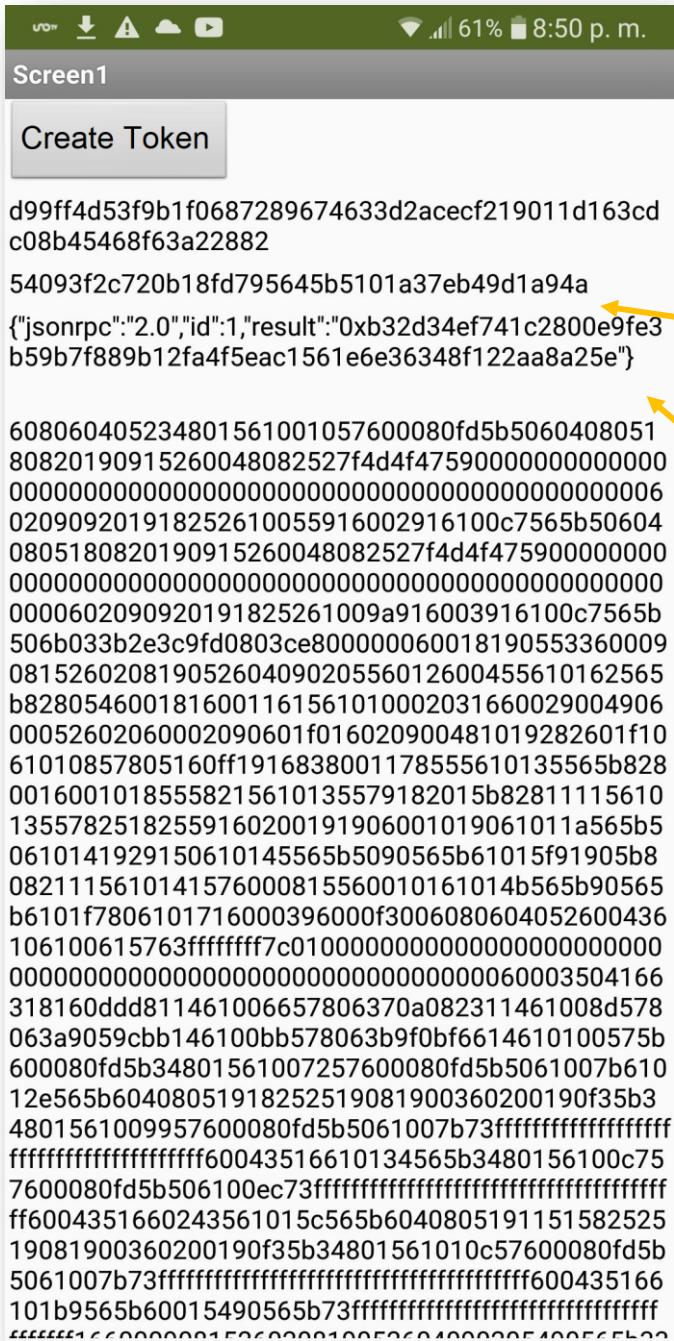
输出参数：事件（OutputSmartContractTokenERC20v1）。

输出：tx_hashToken<String>， addressToken<String>， binCodeToken<String>，
trasactionOperation<String>。



描述：智能合约"代币ERC20"--在Ethereum网络上发布的资产。第1版（v1）已经将气体极限参数设置为50万WEI。

前一个函数SmartContractCreateTokenERC20v1的输出示例。



交易 (Tx_hash) 的Ethereum网络
创建。可在以下网站查询：
www.etherscan.io

新合约的地址，指的是新创建的ERC20代币。

在CoinSolidation.org系统中的验证
操作的交易（Tx_hash）。

这可以在以下网站找到：
www.etherscan.io

EVM (Ethereum虚拟机) 中处理的新代币合约的二进制代码 (BIN))。

9. 创建CryptoToken或Cryptomoney代币的步骤。

第一步：

确认创建智能合约的移动设备上的临时路径是否有效，是否可以成功创建文件。这是用Block (**CreateTestingFile**) 完成的。验证输入"pathTestFile"中给出的测试文件是否已经创建，一般情况下给出的路径为：[/mnt/sdcard/name_file.txt](#)。

第二步（可选）。

在这一步骤中，我们将检查将从该操作中收费的账户（地址）是否有足够的余额来执行创建和发布智能合约的交易。这可以使用块（**eth_VerifiBalanceForTransaccionSmartContract**）来验证。这个块的使用是可选的，因为生成**SmartContractCreateTokenERC20v1**或**SmartContractCreateTokenERC20v2**的块已经在内部包含了这个验证。

第三步：

选择使用哪个区块来创建ERC20令牌，你有两个选择。

- a.-块**SmartContractCreateTokenERC20v1**已经有隐含的Gas Limit值，赋值为500,000 WEi。
- b.-块**SmartContractCreateTokenERC20v2**可以根据终端用户或开发者的需要配置气体限制。需要注意的是，如果给出一个很低的气限，小于35万威，很有可能智能对比。

第四步：

使用**SmartContractCreateTokenERC20v1**或**SmartContractCreateTokenERC20v2**区块，确保在使用输入变量"nameFileSolidityJSON"时，它等于区块的"pathTestFile"输入变量(**CreateTestingFile**)已经在步骤1中检查过。

第五步：

在使用任何一个**SmartContractCreateTokenERC20v1**或**SmartContractCreateTokenERC20v2**块执行ERC20令牌的创建之前，建议根据情况保存Event值（结果），以保存结果(tx_hashToken、addressToken、binCodeToken、transactionOperation）。参见前面函数**OutPutSmartContractCreateTokenERC20v1**的输出示例。

第六步：执行ERC20代币的创建，然后发布出售。见第10节。

10. 如何把一个新的资产或你的加密代币出售（代币ERC20）。

由于我们已经创建了一个ERC20 - Cryptomoney代币（参见 SmartContractCreateTokenERC20v1 Block或与SmartContractCreateTokenERC20v2 Block），我们必须将其上传到一些交易所，以便世界上任何人都可以购买它。交易所是互联网上发布新代币的网站。

交易所分为集中式和分散式两类。主要区别在于，一个是由某种国际机构（Centralized）进行管理和审计，Decentralized的没有人与审计师。虽然这可能会给人更多的信心，但实际情况是，最近分散式的已经占据了更多的力量，而且大部分都在使用，没有出现大的问题。

在处理任何种类的资产时，最好的做法之一是不要把所有的资产都放在一个账户中，而是把它们分布在几个账户中，以保证私钥和公钥的安全。

在我们的案例中，我们将使用一个去中心化的交易所，但已经有了一个不坏的历史，我们将使用www.forkdelta.app。

在这个时候，我们必须已经安装了应用程序的浏览器（Mozilla或Chorme）METAMASK www.metamask.io 这是因为交易所访问您的页面www.forkdelta.app 需要连接到我们有 Ethereum的帐户。

重要的一点是，我们在METAMASK中已经拥有的Ethereum账户应该有或多或少的10美元余额，这是因为当我们发布我们用SmartContractCreateTokenERC20v1区块或 SmartContractCreateTokenERC20v2区块创建的新的ERC20代币时，我们将需要支付交易来在交易所发布它。

为了使用交易所www.forkdelta.app，我们必须拥有以下我们想要在交易所发布销售的代币数据。

我们之前创建的新ERC20令牌的地址。

0x54093F2C720b18Fd795645b5101A37EB49d1A94a

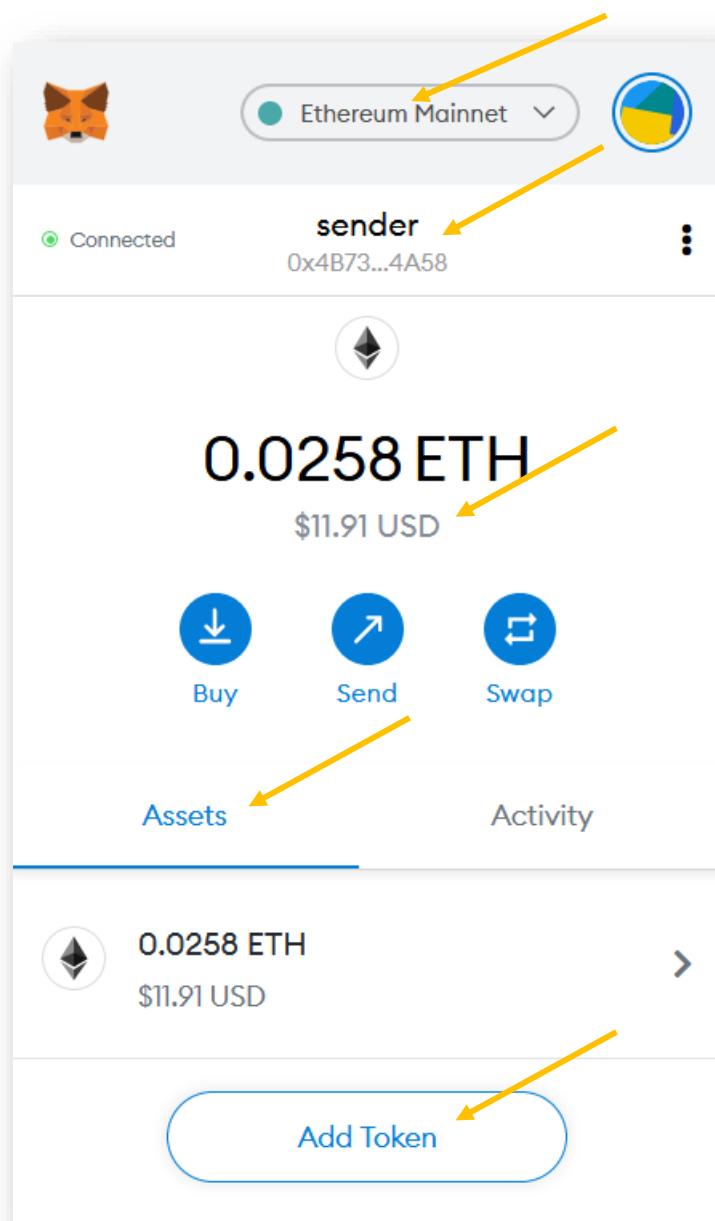
我们的触摸使用的小数点的数量。

识别令牌的名称。

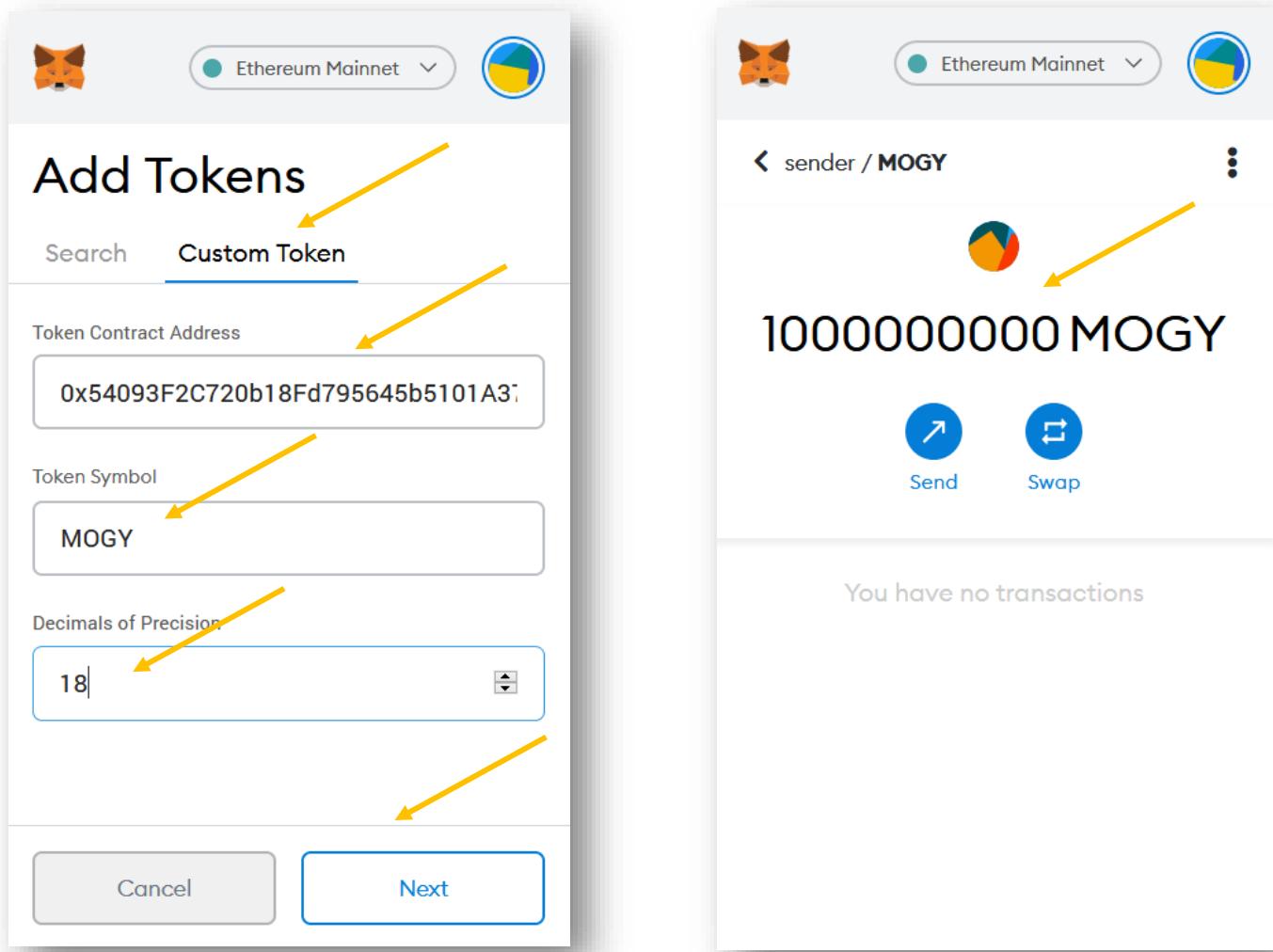
MOGY

让我们先检查一下我们创建的令牌是否有上面提到的参数，以及它们是否是最初创建时的参数。

让我们进入METAMASK，首先确保我们在创建令牌的账户上。然后到底部点击"添加代币"按钮。

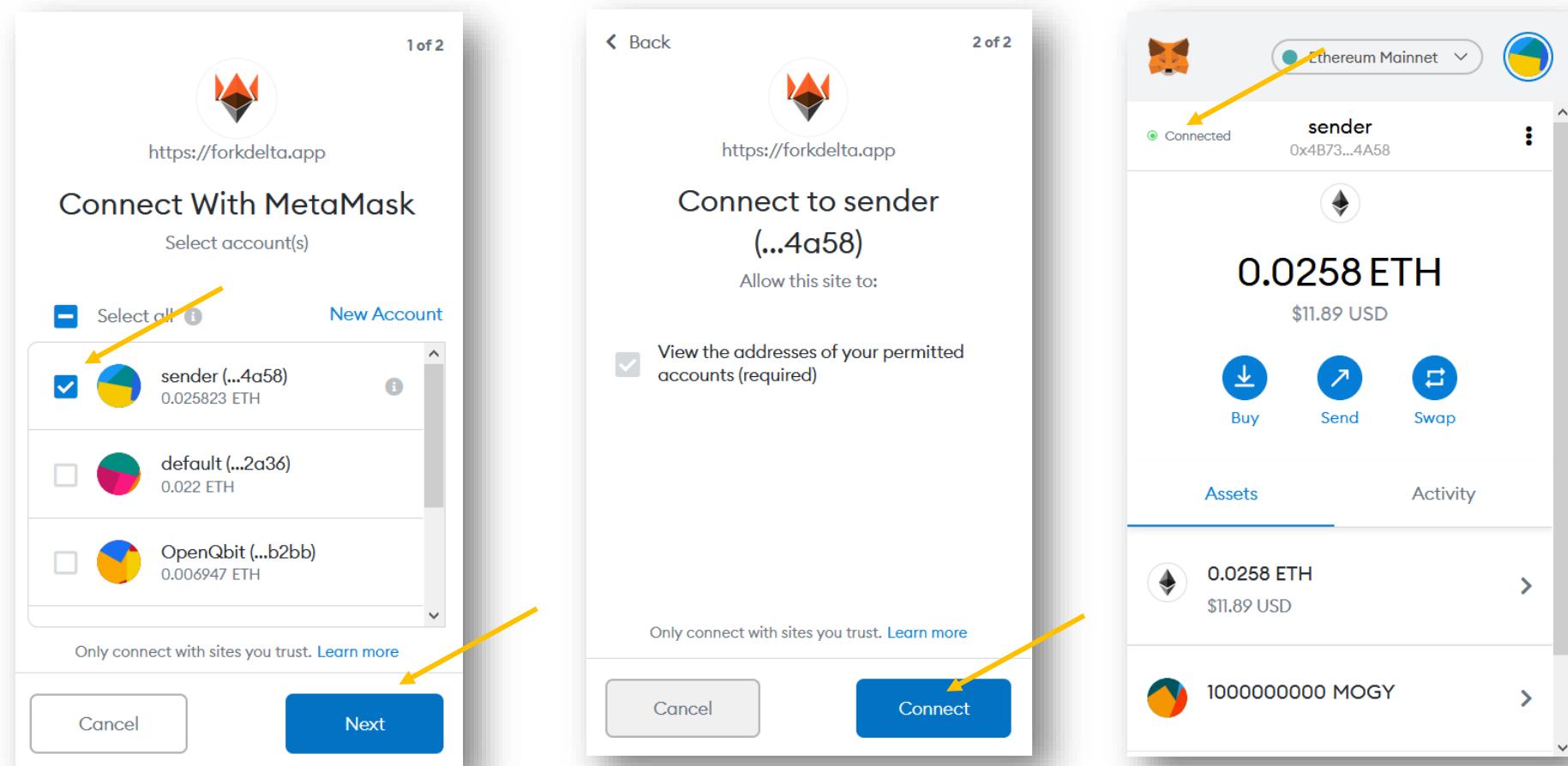


现在，我们将添加我们的新令牌来查看您的当前余额。点击页面顶部的“自定义令牌”按钮后，在以下字段中输入所需信息，然后点击“下一步”按钮。之后，我们的新令牌将与你的余额一起出现。如果您没有余额，请检查您是否在您创建代币的账户中，如果您不在您创建代币的账户中，您的余额将为零，因为您还没有购买任何代币。



一旦我们上传我们的新代币在交易所销售[www.forkdelta.app](https://forkdelta.app)。

进入交流网站后，会要求你连接到网站<https://forkdelta.app>，点击下面的“下一步”按钮，再点击“连接”，最后就可以检查是否连接到forkdelta.app的网站了。

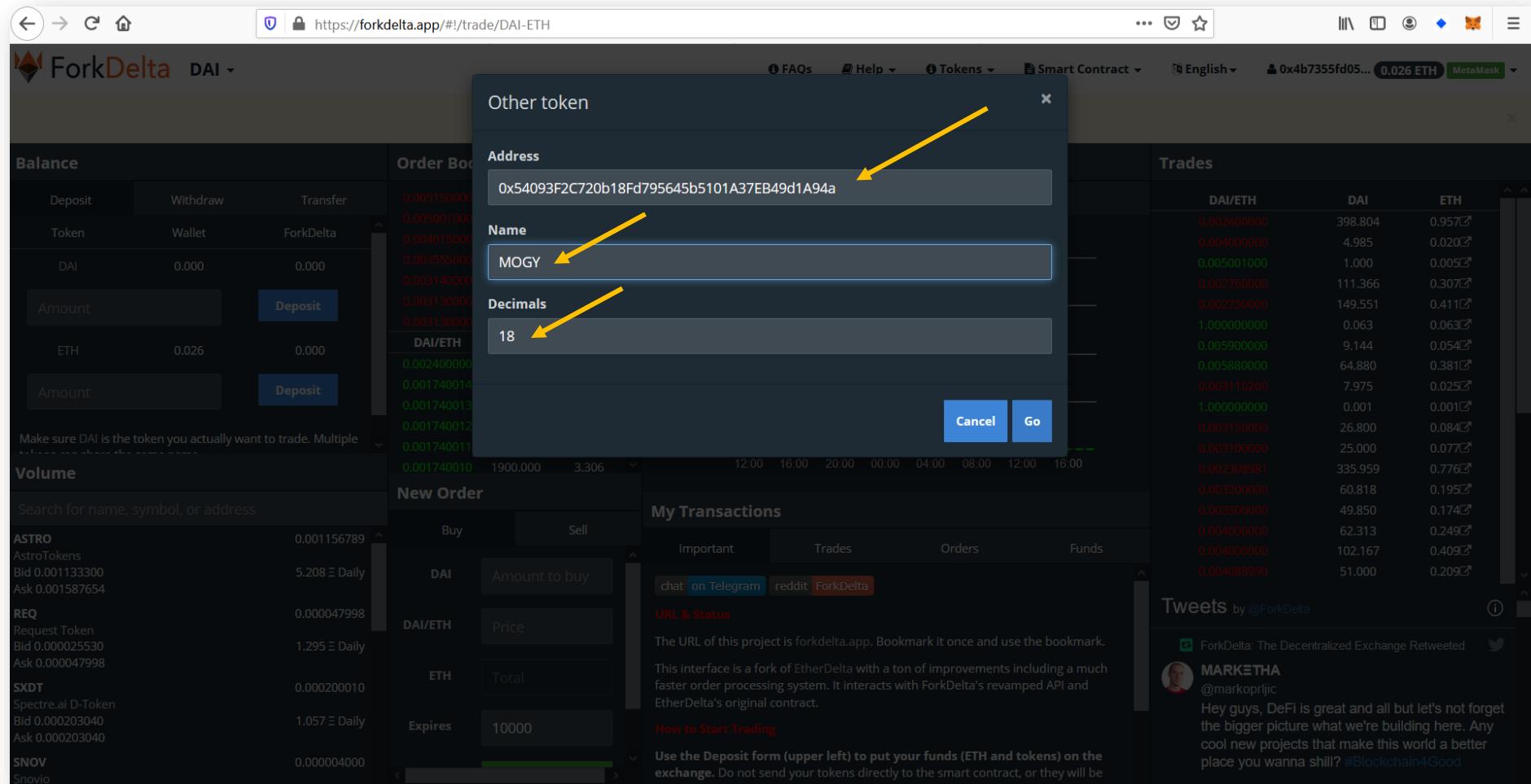


是时候在交易所发布新的代币了<https://forkdelta.app>。

点击上面的菜单"DAI", 走到滚动的最后, 选择"其他"选项。

The screenshot shows the ForkDelta app interface. At the top, there's a navigation bar with links for 'FAQs', 'Help', 'Tokens', 'Smart Contract', 'English', and a MetaMask integration. Below the navigation is a search bar with the URL 'https://forkdelta.app#!/trade/DAI-ETH'. On the left, there's a sidebar with sections for 'Balance' (Deposit, Token, DAI, ETH), 'Volume' (Amount), and a list of tokens like ZAP, ZCG, ZDR, ZIL, ZIP, ZRX, ZSC, ZXBT, cV, eGO, and ePRX. A yellow arrow points from the text above to the 'DAI' dropdown in the sidebar. Another yellow arrow points to the 'Other' option in the dropdown menu. The main area has tabs for 'Order Book', 'Price Chart', and 'Trades'. The 'Price Chart' tab is active, showing a candlestick chart for DAI/ETH with price levels at 0.00, 0.25, 0.50, 0.75, and 1.00. The 'Trades' tab shows a list of recent trades. Below the chart, there's a 'New Order' form and a 'My Transactions' section with tabs for 'Important', 'Trades', 'Orders', and 'Funds'. A 'Tweets' section on the right shows a tweet from @ForkDelta. The bottom of the page has a note about bookmarking the URL and a 'How to Start Trading' guide.

然后我们用我们已经知道的数据注册新的令牌。



此刻新的代币会出现在交易所的左上方，我们只是把它上传到了交易所，我们需要把它公布出来，让大家都能买到它，看到它。现在我们需要从我们的账户中存入一些以太币(0.015)到交易所。

The image consists of two side-by-side screenshots of the ForkDelta website. Both screenshots show the 'Balance' section of the interface.

Left Screenshot:

- The 'Token' dropdown is set to 'MOGY'.
- The 'Withdraw' tab is selected.
- The 'ForkDelta' column shows a balance of '0.000'.
- The 'Wallet' column shows a balance of '1000000000.000'.
- A yellow arrow points from the 'Amount' input field below 'MOGY' to the 'Wallet' balance.
- A second yellow arrow points from the 'Amount' input field below 'ETH' to the 'ForkDelta' balance.
- A message at the bottom left says: 'Make sure MOGY is the token you actually want to trade.'
- A message at the bottom right says: 'Use this to deposit from your personal Ethereum wallet ("Wallet" column) to the current smart contract ("ForkDelta" column).'

Right Screenshot:

- The 'Token' dropdown is set to 'MOGY'.
- The 'Deposit' tab is selected.
- The 'ForkDelta' column shows a balance of '0.000'.
- The 'Wallet' column shows a balance of '1000000000.000'.
- A yellow arrow points from the 'Amount' input field below 'ETH' to the 'ForkDelta' balance.
- A second yellow arrow points from the 'Amount' input field below '0.015' to the 'Deposit' button.
- A message at the bottom left says: 'Make sure MOGY is the token you actually want to trade.'
- A message at the bottom right says: 'Use this to deposit from your personal Ethereum wallet ("Wallet" column) to the current smart contract ("ForkDelta" column).'

既然我们已经交了押金， 我们就可以在交易所存入我们想要的代币，

[www.forkdelta.app.](https://forkdelta.app/)

为了存入规定数量的代币，我们需要创建一个购买订单，这是在底部的"新订单"处完成的，我们点击"卖出"选项。然后我们输入我们想要提供给世界上任何买家的代币数量，我们想要出售每一个代币的以太币价格（单价），而"到期"参数则是我们想要这些代币出售的时间。

过期时间=14秒X输入金额。

例如：14秒X10000=140000秒=1.62天。

The screenshot shows the ForkDelta app interface. On the left, there's a 'Balance' section with tabs for Deposit, Withdraw, Transfer, Token, Wallet, and ForkDelta. Under Token, MOGY is listed with 1000000000.000. Below it are input fields for Amount (with placeholder 'Deposit') and ETH (with value 1.057). On the right, there's an 'Order Book' section for MOGY/ETH, MOGY, and ETH, which currently shows 'There are no orders here.' In the center, a 'New Order' modal is open. It has tabs for Buy and Sell, with Sell selected. The modal contains four input fields: 'MOGY' with value '10000', 'MOGY/ETH' with value '0.05', 'ETH' with value '500.000', and 'Expires' with value '10000'. A note at the bottom of the modal says: 'Make sure MOGY is the token you actually want to trade. Multiple tokens can share the same name.' Arrows from the text above point to each of the four input fields in the 'New Order' modal.

在那里，你的新代币正在出售，任何人都可以进来购买。有时它不能识别新的令牌，所以它们必须从Metamask应用程序中出售。

在新创建的代币网站www.etherscan.io, 咨询的例子。

The screenshot shows the Etherscan Transaction Details page for a specific Ethereum transaction. The transaction hash is 0xd99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882. The status is marked as 'Success'. The transaction was included in block 11240201, which has 224 block confirmations. It occurred 47 minutes ago (Nov-12-2020 02:49:56 AM +UTC) and was confirmed within 30 seconds. The transaction originated from address 0x4b7355fd05be6dac458b004f54e12d6527a54a58 and was sent to a contract at address [Contract 0x54093f2c720b18fd795645b5101a37eb49d1a94a Created]. The value was 0 Ether (\$0.00). The transaction fee was 0.011014691 Ether (\$5.06), and the gas price was 0.000000041 Ether (41 Gwei). The gas limit was 500,000, and the gas used by the transaction was 268,651 (53.73%).

**交易 (Tx_hash) 的Ethereum网络
合约**

合同地址新创建的令牌。

**仅仅是Ethereum网络成本。
不包括操作费用+15美元..**

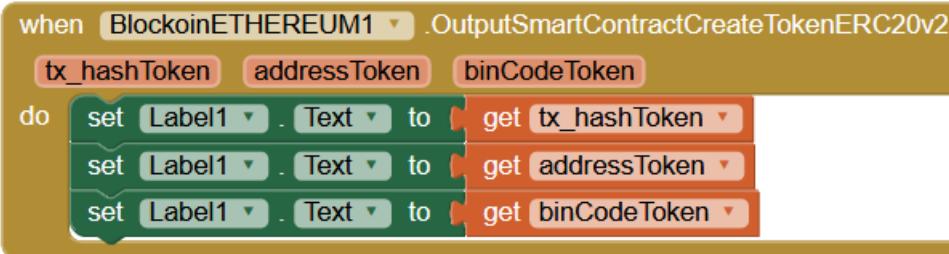
编译、创建和发布ERC20令牌的区块--(SmartContractCreateTokenERC20v2)



输入参数 : tokenName <String>, tokenSymbol <String>, totalAmount <String>, numberDecimal <String>, gas_limit <Integer>, privateKeyHex <Integer>, nameFileSolidityJSON <String>。

输出参数 : 事件 (OutputSmartContractTokenERC20v2) 。

输出 : tx_hashToken<String>, addressToken<String>, binCodeToken<String>。



描述 : 智能合约"代币ERC20"--活跃在Ethereum网络上发布。版本2 (v2) 可以优化气体限制值, 因为根据智能合约的速度, 你想在以太坊网络上进行发布。建议最小值应为35万WEI, 以便无故障或延迟发布。

创建TokenERC20v1和创建TokenERC20v2的功能区别在于, 在版本1中, GasLimit的参数已经预先配置好了, 而在版本2中, 它可以根据用户或开发者的需求进行配置。

阻止调用或执行ERC20令牌 - (SmartContractExecution)



输入参数：地址SmartContract<String>, nameFileSolidityJSON<String>, funtionExecutionSmartContract<String>。

输出参数：执行被引用的智能合约中指定的功能。

输出：**智能合约的执行。**

注意：要执行，必须创建一个JSON格式的文件，其中包含你要执行智能合约的地址主键的参数，你需要输入气限（WEI）。

以上述编译器功能为例，执行编译后的智能合约功能的JSON文件示例。文件名可以是任意的。

文件：call.json

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

智能合约的"问候"功能的输出示例。

```
{  
"gas_limit": 20,000,  
"address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",  
"结果": [  
"你好 BlockCypher 测试"  
]  
}
```

ERC20代币属性显示块 - (SmartContractGetCreationTx)

call BlockoinETHEREUM1 .SmartContractGetCreationTx

txSmartContract

" d99ff4d53f9b1f0687289674633d2acecf219011d163cdc0... "

输入参数 : txSmartContract<字符串>。

输出参数: 显示以 (Tx_hash) 引用的智能合约的属性。

描述 : 显示被引用的智能合约的主要特征和ABI代码。

ERC20令牌的属性示例, 之前使用函数(martContractCreateTokenERC20v1)创建的 tokenName"MOGY"示例。

```
{  
  "block_hash":  
    "cadb8914c43ed28fb370c9e1b6f5d8818ba31a1e944d1f7049441b982902dea8",  
  "block_height":11240201。  
  "block_index":170。  
  "hash":  
    "d99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882",  
  "地址": [  
    "4b7355fd05be6dac458b004f54e12d6527a54a58"  
  ],  
  "总数":0。  
  "费用":110146910000000。  
  "大小":958。  
  "gas_limit": 500,000,
```


显示ERC20令牌编译代码的块-- (SmartContractGetcodeABI)。

```
call BlockchainETHEREUM1 .SmartContractGetcodeABI  
addressSmartContract "0eb688e79698d645df015cf2e9db5a6fe16357f1"
```

输入参数：地址智能合约<字符串>。

输出参数：显示引用的智能合约代码。

描述：显示被引用的智能合约的ABI代码。

通用智能合约的ABI代码示例。

```
{  
"稳定性。"contract mortal {/n /*地址所有者*/n address owner;/n /  
*该函数在始时执行并设置合同所有者*/n function mortal() { owner = msg.sender; }/n /  
*函数返回同的签名*/n function kill() { if (msg.sender == owner)  
suicide(owner); }/n /*回退同的签名*/n function kill() { if  
(msg.sender.sender == owner) suicide(owner); }/n}/n / * 定义字符串的可读写语  
*/n     string greeting;\n/*这合同执行后 *//n function  
greeter(string _greeting) public {/n         greeting = _greeting;\n}/n / * main function */n     function greet() constant returns  
(string) {             /n return greeting;\n}/n }/n",  
"bin". "606060405260405161023e38038061023e83398101604052805101600080546001  
60a060020a031916331790558060016000509080519060200190828054600181600116156  
101000203166002900490600052602060002090601f016020900481019282601f10609f57  
805160ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565  
b50505061016e806100d06000396000f35b828001600101855582156076579182015b8281  
1115607657825182600050559160200191906001019060b0565b509056606060405260e06  
0020a600035046341c0e1b58114610026578063cfae321714610068575b005b6100246000  
543373ffffffffffffffffff908116911614156101375760005  
473ffffffffffffffffff908116911614156101375760005  
805460a06020601f600260001961010086881615020190941693909304928301819004028  
1016040526080828152929190828280156101645780601f106101395761010080835404028
```

```

83529160200191610164565b6040518080602001828103825283818151815260200191508
0519060200190808383829060006004602084601f0104600f02600301f150905090810190
601f1680156101295780820380516001836020036101000a031916815260200191505b509
25050506
"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\\":\\"constructor\\"]",
"creation_tx_hash":
"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
"created": "2016-07-20T01:54:50Z",
"address": "0eb688e79698d645df015cf2e9db5a6fe16357f1"

```

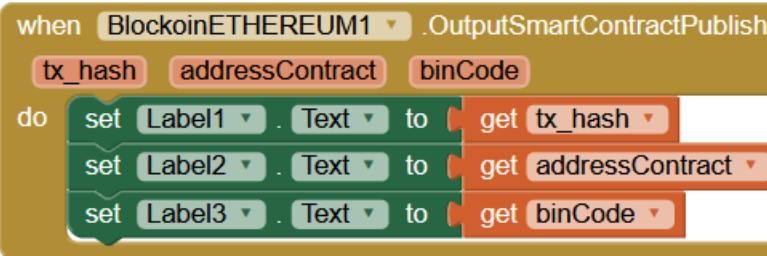
*在使用下面的块(SmartContractPublish)之前，你必须使用块(SmartContractCompilerSolidity)来检查智能合约是否写得很好。

块在Ethereum网络上发布ERC20代币-- (SmartContractPublish)。



输入参数：nameFileSolidityJSON<String>。

输出参数：显示引用的智能合约的属性。在事件（OutputSmartContractPublish）中。



描述：在ethereum网络中发布JSON文件中引用的智能合约。

示例文件：PublishSmartContract.json

```
{
" solidity": "0.4.11",
" contract": "mortal": {
    " inputs": [
        {
            " type": "address",
            " name": "owner"
        }
    ],
    " stateMutability": "nonpayable",
    " functions": [
        {
            " name": "mortal",
            " type": "function",
            " body": "owner = msg.sender"
        },
        {
            " name": "kill",
            " type": "function",
            " body": "if (msg.sender == owner) suicide(owner)"
        }
    ]
},
" contract": "greeter": {
    " inputs": [
        {
            " type": "string",
            " name": "greeting"
        }
    ],
    " stateMutability": "nonpayable",
    " functions": [
        {
            " name": "greet",
            " type": "function",
            " body": "greeting = _greeting"
        },
        {
            " name": "greeter",
            " type": "function",
            " body": "return greeting"
        }
    ]
}
}
```

```

"发布": ["greeter"]。
"私钥": "3ca40..."。
"gas_limit": 500000
}

```

如上面的代码所示，它是在编译函数的例子中使用的相同的JSON代码，在JSON文件的末尾添加了参数。

Params：智能对比的隐性参数。

发布：智能合约发布方式的名称。

私钥：执行智能合约的账户的主密钥必须有余额。

Gas_limit：是你想为智能合约的发布花费的WEI中的余额。

当执行（发布智能合约）智能合约进入以太坊网络时的输出示例。它显示了已执行的交易"**create_tx_hash**"和已创建的智能合约的分配地址"**地址**"。

```

[
{
  "姓名": "greeter",
  "稳固生": "contract mortal {/n /*定义地址    类型拥有者/n address owner;/n /
*该函数在初始时执行，并设置合同所有者    /n function mortal() { owner = msg.sender; }/n /
*函数回始同的资金    /n function kill() { if (msg.sender == owner)
suicide(owner); }/n /*回始同的资金/n function kill() { if
(msg.sender.sender == owner) suicide(owner); }/n}/n/n      *定义字符串类型的变量
/*/n      string greeting;\n/n      /*适合同执行 */      /n function
greeter(string _greeting) public {/n          greeting = _greeting;\n/n
}/n      * main function *//n      function greet() constant returns
(string) {          /n return greeting;\n/n      }/n}",
  "bin": "606060405260405161023e38038061023e83398101604052805101600080546001
60a060020a031916331790558060016000509080519060200190828054600181600116156
101000203166002900490600052602060002090601f016020900481019282601f10609f57
805160ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565
b50505061016e806100d06000396000f35b828001600101855582156076579182015b8281
1115607657825182600050559160200191906001019060b0565b509056606060405260e06
0020a600035046341c0e1b58114610026578063cf321714610068575b005b6100246000
543373fffffffffffff908116911614156101375760005
473fffffffffffff16ff5b6100c9600060609081526001
805460a06020601f600260001961010086881615020190941693909304928301819004028
1016040526080828152929190828280156101645780601f10610139576101008083540402
83529160200191610164565b6040518080602001828103825283818151815260200191508
0519060200190808383829060006004602084601f0104600f02600301f150905090810190
601f1680156101295780820380516001836020036101000a031916815260200191505b509
25050506
  "abi": [
}

```

```

"常数":假
"投入":[],
"名称":"杀"。
"产出":[],
"类型":"功能
},
{
"常数":真
"投入":[],
"姓名":"问候"。
"产出":[
{
"姓名":"",
"类型":"字符串"
}
],
"类型":"功能
},
{
"投入":[
{
"name":"_greeting"。
"类型":"字符串"
}
],
"类型":"建商"
},
],
"gas_limit": 500,000,
"creation_tx_hash":
"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
"address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",
"params": [
"你好 测试"
]
},
{
"姓名":"凡人"。
"稳固": "contract mortal {/n /*定义地址    类型为所有者*/n address owner;/n /
*该函数在初始时执行，并设置合同的所有者*/n function mortal() { owner = msg.sender; }n /
*函数kill同的资金*/n function kill() { if (msg.sender == owner)
suicide(owner); }n}n /*回给同的资金*/n function kill() { if
(msg.sender.sender == owner) suicide(owner); }n}/n}/n /      * 定字符串为所有者语
/*/n      string greeting;\n\n      /*这台同执行语*/      /n function
greeter(string _greeting) public {/n          greeting = _greeting;\n\n
}/n /      * main function */n      function greet() constant returns
(string) {          /n return greeting;\n\n        }/n}",
"bin": "606060405260008054600160a060020a03191633179055605c8060226000396000
f3606060405260e060020a600035046341c0e1b58114601a575b005b60186000543373fff
ffffffffffffffffffffffffffffffffffffffff90811691161415605a5760005473ffffffff
ffffffffffffffffffffffff16ff5b56",
"abi": [
{

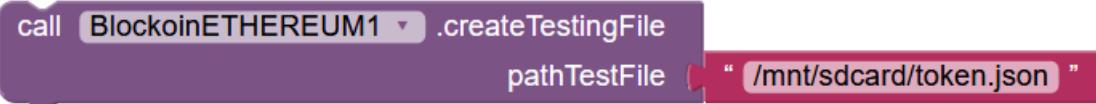
```

```

    "常数":假
    "输入":[],
    "名称":"杀",
    "产出":{},
    "类型":"功能"
},
{
    "输入":{},
    "类型":"建商"
}
],
"gas_limit":500,000,
"params":["你好 测试"]
}
]

```

创建文件的测试块 - (createTestingFile)



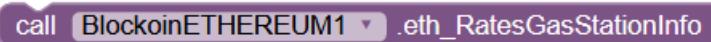
输入参数：pathTestFile<字符串>

输出参数：在引用的路径中创建一个测试文件。

说明：其作用是在使用块（`SmartContractCreateTokenERC20v1`）或块（`SmartContractCreateTokenERC20v2`）时，检查有效的临时文件创建路径，确保其正确。

-

块获取天然气价格的比率 - (eth_RatesGasStationInfo)。



输入参数：nameFileSolidityJSON<String>。

输出参数：显示引用的智能合约的属性。在事件(`OutputEth_GasStationInfo`)中，交付的值以GWEI形式给出。

Gas Price是将支付给在Ethereum的网络中执行交易的系统的价值。这些系统通常被称为"矿工"，Gas Price的价值是Ethereum的网络中执行交易的速度（时间和优先级）的函数。

交付的数值是下列运行时间的函数：这些时间是近似的，可能会根据你在Ethereum网络上的需求（交易）而有所不同。

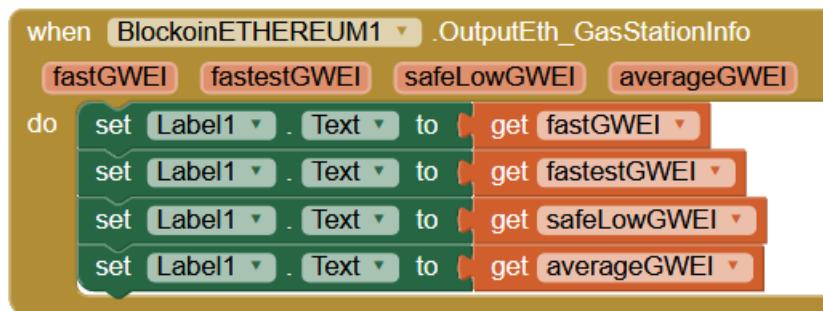
快速<2分钟。

最快<30秒。

SafeLow<30分钟。

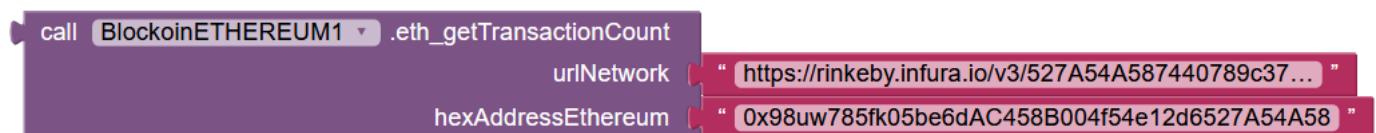
平均<15分钟。

与交易所Ethereum扩展（EEE）进行的交易总是使用天然气价格=平均。



描述：获取查询时更新的天然气价格，以创建新的交易。

块获取数字"nonce" - (eth_getTransactionCount)。



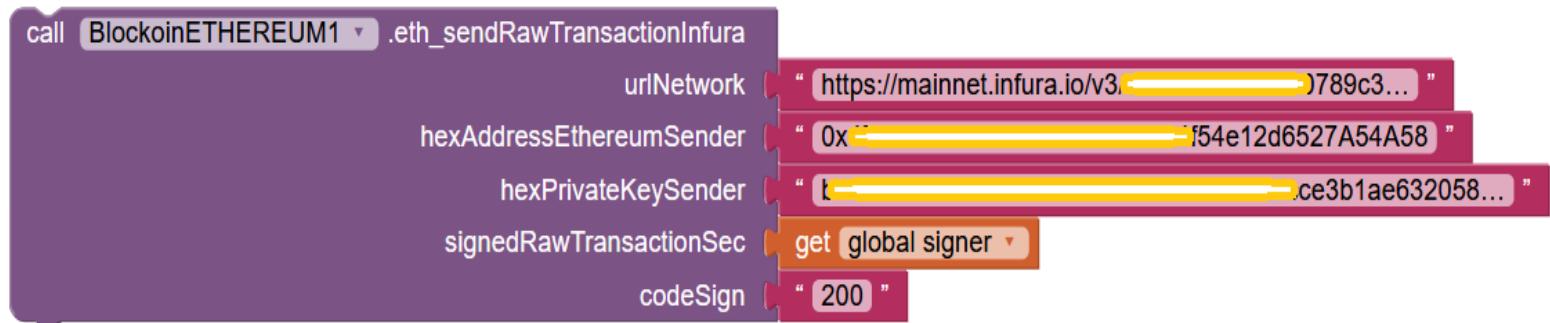
输入参数：urlNetwork<字符串>, hexAddressEthereum<字符串>。

输出参数：它以十六进制格式显示引用地址的连续数字"nonce"。

"nonce"数字是一个增量数字，用于记录从特定地址进行的交易次数。

说明：获取被引用地址的"nonce"号码。

块发送签名交易--(eth_SendRawTransactionInfura)。

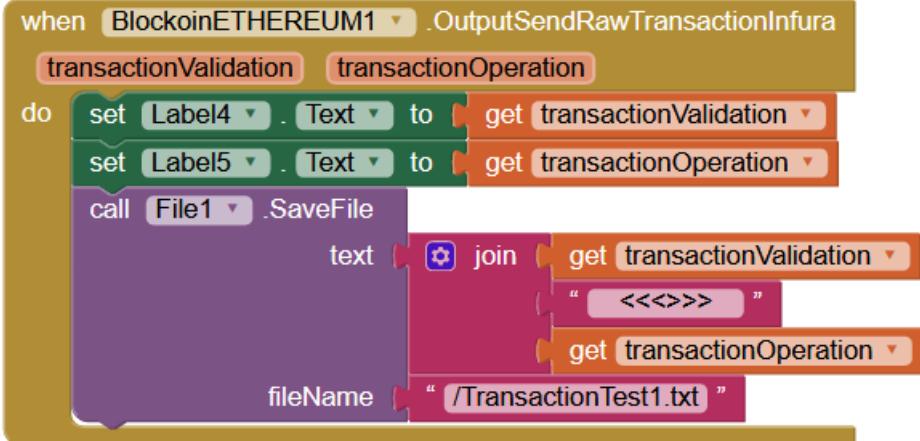


所需的依赖关系：Block (eth_getTransactionCount), Block
`(SignerGenericPushRawTransactionOffline)`.审查区块的例子（`SignerGenericPushRawTransactionOffline`）。

输入参数：`urlNetwork <String>`, `hexAddressEthereumSender <String>`,
`hexPrivateKeySender <String>`, `SignedRawTrasactionSec <String>`, `codeSign <String>`。

输出参数：事件 `(OutputSendRawTransactionInfura)`。

输出：`transactionValidation<String>`, `transactionOperation<String>`。



说明：它提供两个十六进制值作为交易的结果。transactionValidation值是指在 Ethereum网络上进行的交易，它包含了Ethereum的隐性成本。交易价值Operation是指在Coinolidation网络中进行的交易，根据交易时乙醚的价值，每笔交易的成本为0.5美元，该成本用于支付Coinolidation.org网络的服务费用，并投入到全球加密和资产领域的维护、支持和创建扩展。

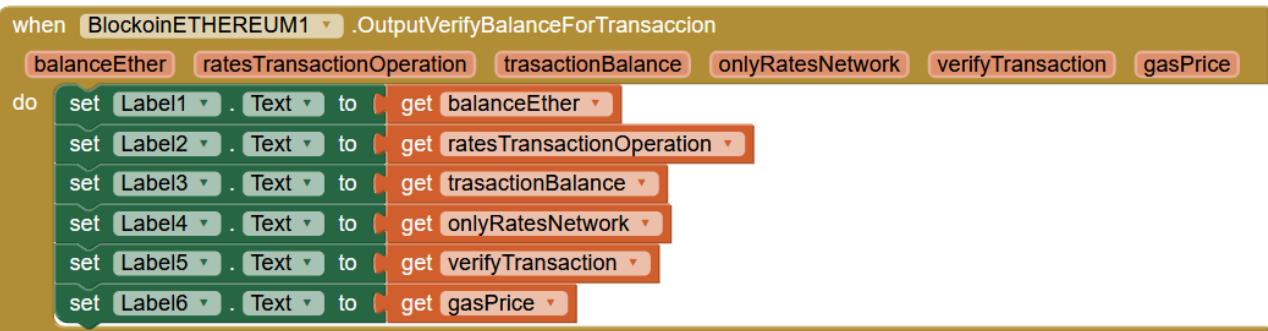
计算标准交易成本的块 - (eth_VerifiBalanceForTransaction)



输入参数：**地址**EthereumSender<String>， 值EthertoSend<String>。

输出参数：事件（OutputVerifyBalanceForTransaction）。

输出：balanceEther<String> , ratesTransactionOperation<String> ,
trasactionBalance<String> , OnlyRatesNetwork<String> , verifyTransaction<String> ,
gasPrice<String>。



说明：提供标准交易费用的详细情况，并参照进入地址。输出参数"verifyTransaction"告诉我们是否可以让交易成为"True"，如果引用的地址没有足够的余额会给我们一个"False"。

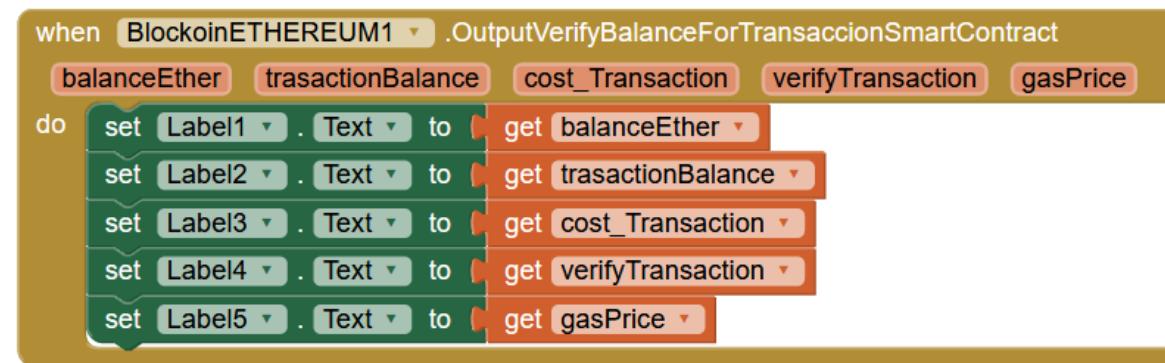
计算标准交易成本的块 - (eth_VerifiBalanceForTransaccionSmartContract)



输入参数：addressEthereum<String>, gasLimit<String>。

输出参数：事件(OutputVerifyBalanceForTransaccionSmartContract)

输出：balanceEther<String> , transactionBalance<String> , cost_Transaction<String> , verifyTransaction<String> , gasPrice<String>。



说明：提供详细的标准交易的费用大概是多少，参考入驻地址，发布**智能合约**。输出参数"verifyTransaction"告诉我们是否可以让交易成为"True"，如果引用的地址没有足够的余额会给我们一个"False"。

balanceEther：引用的地址的余额以以太形式交付。

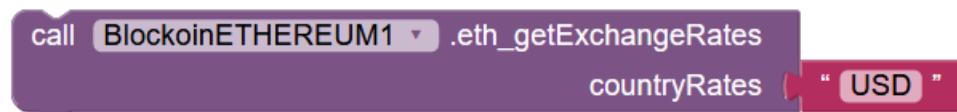
trasactionBalance：交易完成后的余额。

cost_Transaction: 是发布智能合约的交易成本。

验证交易：(balanceEther减去cost_Transaction)。

gasPrice："矿工"使用的GasPrice的当前值，这个值每分钟都会变化。

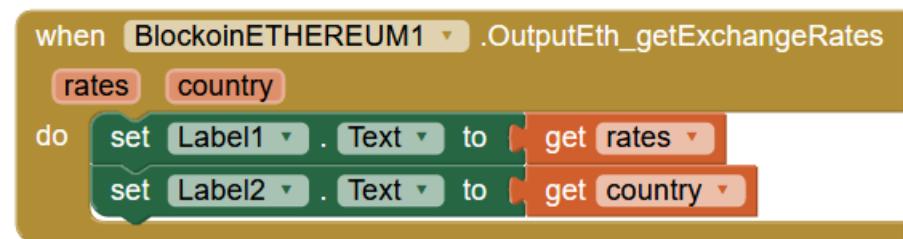
块获取指定国家货币的以太币价格 - (**eth_getExchangeRates**)。



输入参数：**countryRates<String>**。勾选输出参数"国家"，其中包含所有国家的货币类型，选择所需的货币类型。

输出参数：事件(**OutputEth_getExchangeRates**)。

输出：**rate<String>、country<String>**以JSON格式输出世界各国的所有汇率。



说明：它提供了以太币的当前价格，并以参考国家的货币汇率进行计算。

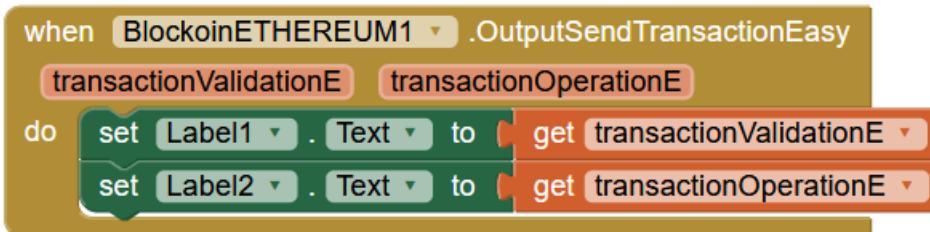
用最小的输入参数来执行标准交易的块--(eth_sendTransactionEasy)。



输入参数 : hexPrivateKeySender<String>, toAddress<String>, valueEther<String>。

输出参数 : 事件 (OutputSendTransactionEasy) 。

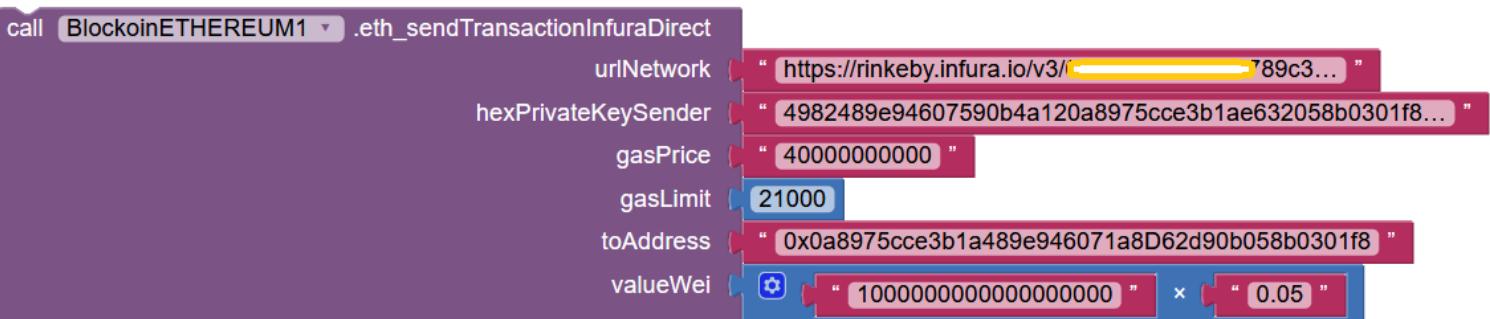
输出 : transactionValidation<String>, transactionOperation<String>。



说明 : 在Ethereum的网络中进行标准交易的功能, 这个功能是立即使用的, 你不需要在INFURA有一个账户, 你只需要3个输入参数, 你只需要有足够的余额来进行所需的交易。

使用官方的Ethereum Web3j库和我们的网络Coinolidation.org直接在Ethereum网络上进行交易。

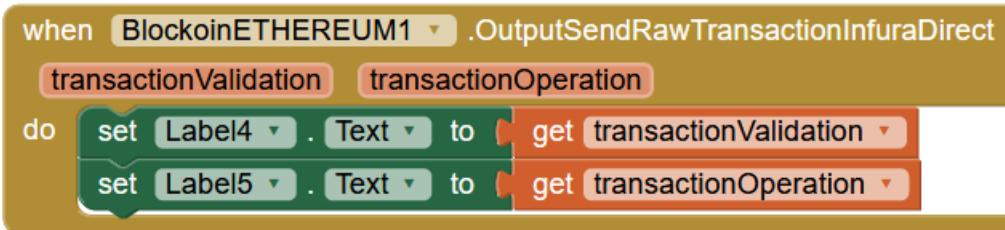
用最小的输入参数--(eth_sendTransactionInfuraDirect)来执行标准交易的块。



输入参数：urlNetwork<String>, hexPrivateKeySender<String>, gasPrice<String> ,
gasLimit<String>, toAddress<String>, valueWEI<String>。

输出参数：事件（OutputSendTransactionInfuraDirect）。

输出：transactionValidation<String> , transactionOperation<String>。



描述：发送一个已经包含隐含数字签名的标准交易的功能，这对于已经有了之前的交易组件知识，并希望根据自己的需求优化这些参数的人来说是很有用的。

11. 标准交易成本和智能合约交易的计算方法

对于标准交易的计算，在Ethereum网络中需要3个参数。

- 1.-天然气价格；
- 2.-气限。
- 3.-以太币（Ethereum数字货币）的当前价值。

GasPrice：通常以GWEI(GigaWEI)为单位。这相当于，如果1个醚有1,000,000,000,000,000威，那么1个GWEI等于1,000,000,000。这个单位作为执行Ethereum网络所有交易的系统的支付，被称为"矿工"，它分布在世界各地。GasPrice

不是一个固定的值，是可变的，可以从分到秒的变化。定义GasPrice价值的人是"矿工"，这取决于Ethereum网络的饱和度。

GasLimit: 这个值通常以WEI为单位，在标准交易中，一个平均的默认值是21,000 WEI，尽管它可以更高或更低，这取决于你想进行的交易类型，在标准交易中，我们使用40,000 WEI的值，以确保我们不会因为低于Gas Limit而被拒绝。

以太币的价值：这个价值也是可变的，是由于全球金融市场的不同参数，这个价值可以从全球范围内一直更新以太币价值的实体获得，称为中心化和去中心化交易所。

重要提示：在交易所Ethereum扩展（EEE）中，我们使用较高的气体限制（40,000），这是由于在没有达到"矿工"建立的最低配额的情况下，交易不受影响，但是，Ethereum网络，如果它将收取的费用，它已经在计算的交易没有使它。出于这个原因，有些用户并没有解释为什么他们的交易没有执行，然而他们会被收取一笔金额或所有交易请求发起时提供的GasLimit，出于这个原因，我们必须始终明确GasLimit和Gas Price的值是什么。

GasPrice可以通过函数`eth_GetRatesGasStation`进行查询，标准交易的GasLimit必须高于21,000 WEI，发布和/或执行智能合约的交易必须至少为500,000 WEI。

标准交易成本。

它由以下公式定义：

成本=(GasLimit×(GasPrice/(1,000,000,000,000))×以太值。

例如：

假设以下数值：

GasLimit=40000，GasPrice=45GWEI，以太价值=406美元。

费用=(40,000×(45,000,000,000/(1,000,000,000,000))×406。

标准交易成本=0.0018乙醚×406美元=0.73美元。

在进行智能合约交易时，必须知道将发布何种智能合约，因为成本与"矿工"处理智能合约的工作量成正比，换句话说，"矿工"处理的计算机系统的处理量比较简单。

在智能合约的情况下，默认值是以500,000 WEI的值启动GasLimit。

需要注意的一点是，当一个人提出GasLimit时，"矿工"不一定会把提出的金额全部拿出来，也就是说，当一个人发送交易时，"矿工"会计算计算工作量，并从GasLimit中拿出来，这可能会小于或等于某些情况下提供的21000WEI的默认GasLimit。

所有的交易都可以在网站www.etherscan.io，在这里我们可以查询到每笔交易的详细情况。

一个标准交易的例子，在发送交易时，气体限额为40,000 WEI，然而“矿工”在计算处理成本时，只取了52.5%，即默认值为21,000 WEI。

The screenshot shows a transaction details page on Etherscan. The transaction hash is 0x7dae251f21c616fb04a94112633c97e04d11c91f4d06dd9b85ed11112f02703a. It was successful and confirmed in block 11234630. The transaction occurred 52 seconds ago on Nov-11-2020 at 06:17:24 AM UTC. The value sent was 0.001084598698481562 Ether (\$0.50). The transaction fee was 0.000672 Ether (\$0.31). The gas price was 0.000000032 Ether (32 Gwei) and the gas limit was 40,000. The gas used was 21,000 (52.5%).

参数	拟议的值	实际值
Gas Limit	40,000	21,000 (52.5%)
Gas Used by Transaction	21,000 (52.5%)	21,000 (52.5%)

交易操作或交易验证

以太坊的交易成本。

拟议的气体限额：40 000 WEI

交易中使用的实际气体极限：21000 WEI。

回到智能合约交易，取50万WEI的GasLimit，如果我们全部使用，我们得到以下成本。

智能合约交易成本。

它由以下公式定义：

成本=(GasLimit×(GasPrice/(1,000,000,000,000))×以太值。

例如：

假设以下数值：

GasLimit=50万， GasPrice=45GWEI， 以太价值=406美元。

费用=(500,000×(45,000,000,000/(1,000,000,000,000))×406。

交易成本发布智能合约=0.0225乙醚×406美元=9135美元。

所有的交易都可以在网站上查询www.etherscan.io。

注意：要想获得交易的总成本，您必须将它们加起来。

总交易成本=Ethereum网络成本+Coinolidation.org费用。

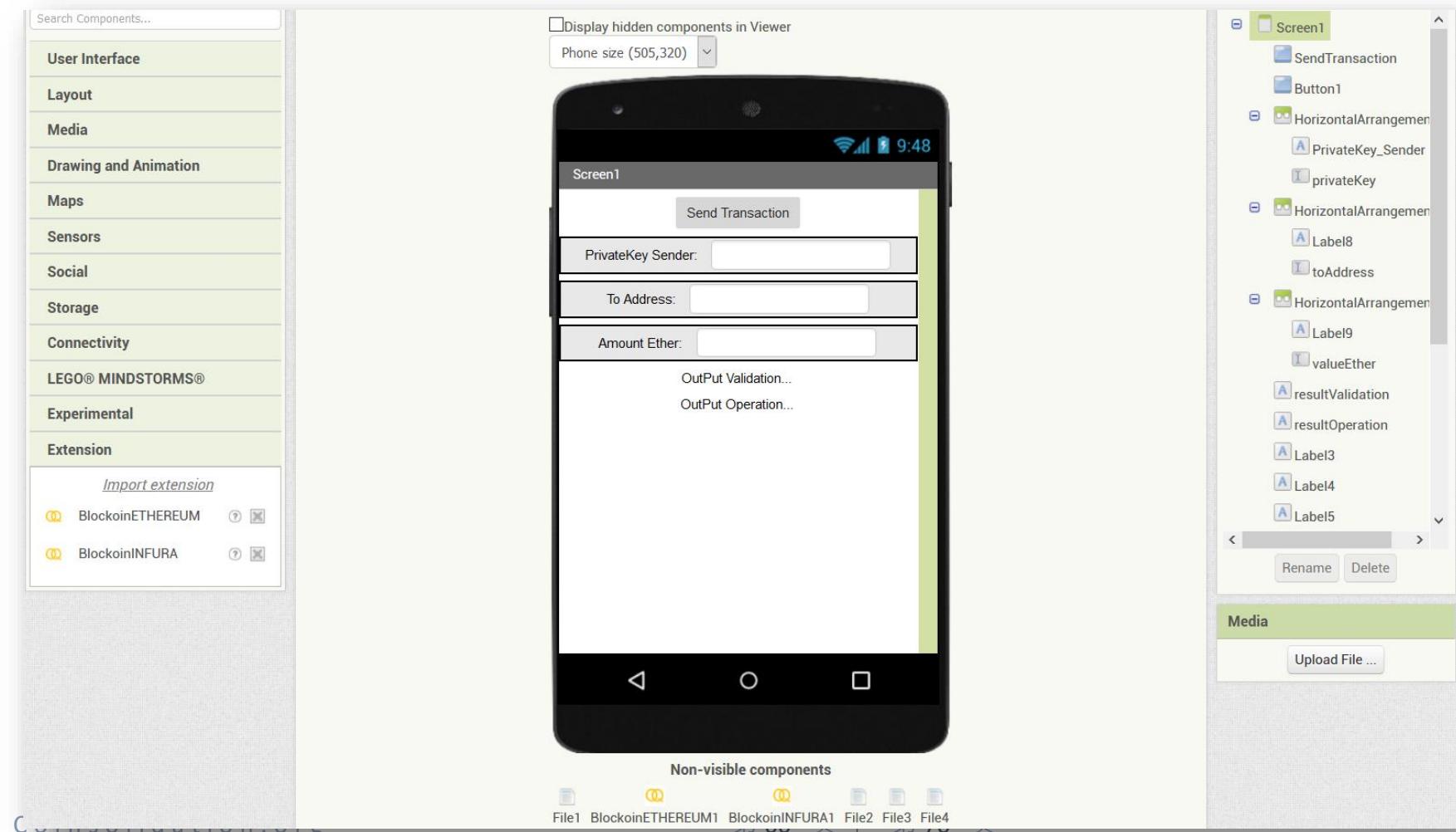
12.Coinolidation.org费用

标准交易：0.5美元+Ethereum网络成本。

交易发布和/或执行智能合约：15美元+Ethereum网络的成本。

13. 在15分钟内创建你的Android应用程序（Exchange）。

在App Inventor（屏幕）中设计。- 五分钟



功能块(eth_SendTransactionEasy)和事件(OutPutSendTransactionEasy)- 5分钟

The image shows two Scratch scripts. The top script, triggered by a button click, sends a transaction using the `BlockoinETHEREUM1` extension's `.eth_sendTransactionEasy` function. The bottom script, triggered by the extension's `OutputSendTransactionEasy` event, handles the validation and operation results.

Top Script (when [SendTransaction v].Click):

```
when [SendTransaction v].Click
do
  call [BlockoinETHEREUM1 v].eth_sendTransactionEasy
    hexPrivateKeySender [privateKey v]
    toAddress [toAddress v]
    valueEther [valueEther v]
```

Bottom Script (when [BlockoinETHEREUM1 v].OutputSendTransactionEasy):

```
when [BlockoinETHEREUM1 v].OutputSendTransactionEasy
  transactionValidationE transactionOperationE
do
  set [resultValidation v] to (get [transactionValidationE v])
  call [File1 v].SaveFile
    text (get [transactionValidationE v])
    fileName "/trasactionValidation.txt"
  set [resultOperation v] to (get [transactionOperationE v])
  call [File1 v].SaveFile
    text (get [transactionOperationE v])
    fileName "/trasactionOperation.txt"
```

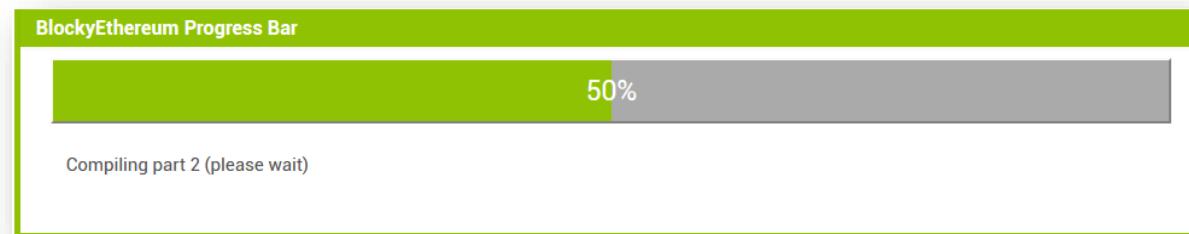
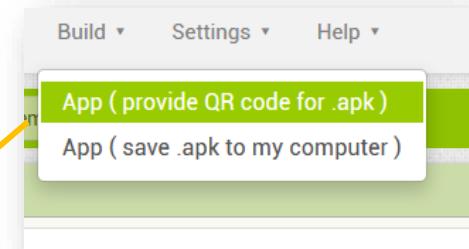
Input Data:

- PrivateKey**: Sender's private key.
- toAddress**: Receiver's address.
- valueEther**: Ether amount to send.

Result Processing:

- Validation Result**: Saved to file `/trasactionValidation.txt`.
- Operation Result**: Saved to file `/trasactionOperation.txt`.

我们编译、生成APK文件安装到安卓设备上。- 5分钟



注意：当交易被执行时，大约需要6到8秒才能释放“发送交易”按钮。由于与Ethereum网络的连接时间。

14. Token CoinSolidation。

代币币种整合是一个项目，主要有三个准则。

想象一下，使用COINSolidation为您的业务拥有自己的加密货币令牌会自动为您免费提供令牌。

首先是创建第一个网络的扩展基于可视化编程方法Blockly，在其简单和直观的使用，可以由任何人使用，没有以前的编程知识，扩展，可以咨询我们的路线图（白皮书）是针对两个基本部门的世界经济，部门的加密货币和/或代币和部门的货币（法币）或共同使用的世界范围内，如美元，欧元欧盟，英镑或任何其他货币的使用。

CoinSolidation项目的第二条准则是创建一种新的算法，以整合当前和未来加密货币的地址。我们正在开发一种新的模型算法，以创建一个地址，将不同区块链的不同地址整合成一个通用地址，请见我们的（白皮书）www.CoinSolidation.org 或 <https://github.com/coinsolidation/whitepaper>。

第三条准则是将量子计算技术应用于CoinSolidation环境的安全，这将与已经开发的QRNG（量子随机数生成器）和PQC（后量子计算）安全算法的扩展应用。这些可以在Github网站的官方扩展库中找到，<https://github.com/coinsolidation>。

CryptoToken CoinSolidation的一般特点。

名称: CoinSolidation

符号 : CUAG

类型 : NFT

发射国 : 爱沙尼亚

官方网站 : www.Coinsolidation.org

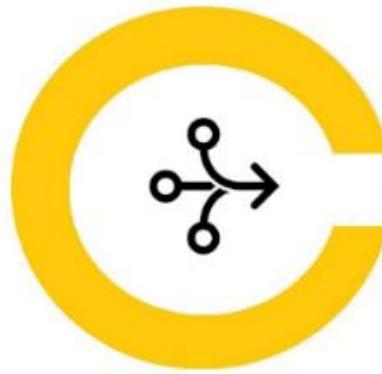
公司 : 币安国际。

发射日期 : 2021年4月31日

创始人 : Guillermo Vidal。

共识算法 : 证明量子。

地址算法。综合通用地址 :



15.软件的许可和使用；

使用许可、条款和条件见 www.coinsolidation.org 或写信至 info@coinsolidation.org。