



Конфигурация и администрирование.

Расширение Ethereum Exchange (EEE).

# руководство пользователя

версия 1.0.0 Beta

Ноябрь 2020 года.

Blockoin.org является зарегистрированной торговой маркой Bankoin Inc, на условиях свободной лицензии и коммерческого использования. Условия использования по адресу: [www.CoinSolidation.org](http://www.CoinSolidation.org)

## Содержание

1. Введение.....	3
2. Что такое блочное программирование?.....	5
3. Что такое расширение?.....	5
4. Что такое BlockoinEthereum и BlockoinINFURA?.....	5
5. Основные концепции, применяемые в платформе Ethereum. ....	6
6. Установка и конфигурация среды расширения и тестирования Ethereum. ....	13
7. Определение и использование блоков (общая функция) .....	21
8. Расширение Ethereum (EEE) функций и событий. ....	22
9. Шаги по созданию крипто-токена или крипто-мони. ....	34
10. Как выставить на продажу новый актив или ваш токен-склеп (Token ERC20).....	35
11. Расчет стандартной стоимости операции и "умная" контрактная операция .....	59
12. Плата за участие в конкурсе Coinsolidation.org.....	63
13. Создайте ваше приложение для Android (Exchange) за 15 минут. ....	64
14. Уплотнение монет Токена.....	67
15. Лицензирование и использование программного обеспечения.....	68

## 1. Введение.

Ethereum является платформой с открытым исходным кодом, децентрализованной в отличие от других цепей блоков, Ethereum может сделать гораздо больше. Она программируема, а это значит, что разработчики могут использовать ее для создания новых типов приложений. Ее цифровая валюта называется "Эфир".

Эти децентрализованные приложения (или "dapps") получают преимущества технологии криптомонтажа и блок-цепочки. Они надежны и предсказуемы, а это значит, что после того, как они "загружены" в Этериум, они всегда будут работать по расписанию. Они могут контролировать цифровые активы для создания новых видов финансовых приложений. Они могут быть децентрализованы, что означает, что их не контролирует ни одна организация, ни одно лицо.

Сейчас тысячи разработчиков по всему миру создают приложения на Ethereum и изобретают новые типы приложений, многие из которых можно использовать и сегодня:

- Крипто-валютные портфели, позволяющие осуществлять дешевые мгновенные платежи с ETH или другими активами
- Финансовые заявки, позволяющие заимствовать, одолживать или инвестировать свои цифровые активы.
- Децентрализованные рынки, позволяющие обмениваться цифровыми активами или даже обмениваться "предсказаниями" о реальных событиях.
- Игры, в которых вы имеете активы в игре и даже можете выиграть реальные деньги.

### Как работает эфир?

Эфир, как и другие крипто-валюты, использует общую цифровую книгу, в которой записываются все операции. Она общедоступна, полностью прозрачна и очень трудно поддается последующим изменениям.

Это называется **блок-цепь**, и она создается в процессе **добычи**.

Шахтеры отвечают за верификацию групп эфирных транзакций для формирования "блоков" и их кодирование путем решения сложных алгоритмов. Эти алгоритмы могут быть более или менее сложными как способ сохранить некоторую согласованность во времени обработки блока (примерно один раз в 14 секунд).

Затем новые блоки соединяются с предыдущей цепочкой блоков, и шахтер получает **вознаграждение**, т.е. фиксированное количество эфирных жетонов. Обычно это 5

эфирных единиц, хотя это число можно считать переменным, если валюта крипта продолжает расти.

### Как работает Этериум?

Блокчейн Ethereum очень похож на биткойн, но его язык программирования позволяет разработчикам создавать программное обеспечение, с помощью которого можно управлять транзакциями и автоматизировать определенные результаты. Это программное обеспечение известно как "умный" контракт.

Если в традиционном договоре описываются условия взаимоотношений, то в "умном" договоре эти условия выполняются путем их написания в коде. Это программы, которые автоматически исполняют контракт после выполнения предопределенных условий, исключая задержку и затраты, которые существуют при ручном выполнении договора.

В качестве простого примера, пользователь Ethereum может создать "умный" контракт на отправку определенного количества эфира другу на определенную дату. Они напишут этот код в строке блока, и когда договор будет заключен (т.е. когда будет достигнута согласованная дата), эфир будет отправлен автоматически.

Эта базовая идея может быть применена к более сложным конфигурациям, и ее потенциал, вероятно, неограничен, с проектами, которые уже достигли значительного прогресса в таких секторах, как страхование, недвижимость, финансовые услуги, юридические услуги и микро-финансирование.

Интеллектуальные контракты также имеют ряд дополнительных преимуществ:

1. Они устраниют фигуру посредника, предлагая пользователю полный контроль и минимизируя дополнительные расходы.
2. Они регистрируются, шифруются и дублируются в цепочке публичных блоков, где все пользователи могут видеть рыночную активность.
3. Исключить время и усилия, необходимые в ручных процессах.

Конечно, "умные" контракты - это все еще очень новая система с множеством деталей, которые нужно отполировать. Код транслируется буквально, поэтому любые ошибки при создании контрактов могут привести к нежелательным результатам, которые нельзя будет изменить.

### DApps против "умных" контрактов

Интеллектуальные контракты имеют сходство с **DApps** (децентрализованными приложениями), но в то же время отделяют их от некоторых важных различий.

Как и интеллектуальные контракты, DApp - это интерфейс, который соединяет пользователя со службой провайдера через децентрализованную одноранговую сеть.

Но, в то время как интеллектуальные контракты требуют создания фиксированного количества участников, у DApps нет ограничений на количество пользователей. Более того, они не ограничиваются только финансовыми приложениями, такими как "умные" контракты: DApp может служить любой цели, о которой вы можете подумать.

## 2. Что такое блочное программирование?

**Blockly** - это **визуальная методология программирования**, основанная на языке программирования JavaScript и состоящая из простого набора команд, которые мы можем комбинировать, как кусочки головоломки. Это очень полезный инструмент для тех, кто хочет **научиться программировать** интуитивно понятным и простым способом, или для тех, кто уже умеет программировать и хочет увидеть потенциал этого вида программирования.

Блокли - это форма программирования, где не требуется никакого фона ни на каком компьютерном языке, это потому, что это просто соединение графических блоков, как если бы мы играли в лего или головоломку, вам просто нужна какая-то логика и все!

Любой человек может создавать программы для мобильных телефонов (смартфонов), не связываясь с теми языками программирования, которые трудно понять, просто сложите блоки в графическом виде простым, легким и быстрым способом.

## 3. Что такое расширение?

Расширение - это модуль, созданный на языке программирования Java, приведенный в файле с расширением .AIX.

В проекте Blockoin.org мы основываемся на создании удобных расширений для финансовой сферы, с помощью которых они могут за несколько минут сделать мобильные приложения, не обладая обширными знаниями в области программирования.

## 4. Что такое BlockoinEthereum и BlockoinINFURA?

BlockoinEthereum и BlockoinINFURA - это бесплатное программное обеспечение (расширения), которое включает в себя следующие технологические решения (алгоритмы), которые можно создавать и использовать в сети Ethereum:

- СОЗДАНИЕ НОВЫХ АККАУНТОВ НА ПЛАТФОРМЕ ETHEREUM.
- КОНСУЛЬТАЦИИ ПО БУХГАЛТЕРСКОМУ БАЛАНСУ, ПЕРЕДАЧА АКТИВОВ (КРИПТОМОНОГА ЭФИР И МАРКЕРЫ)
- СОЗДАНИЕ НОВЫХ ЖЕТОНОВ ERC20 И КРИПТОМОНЕДА-ТОКЕНА.

- СОСТАВЛЕНИЕ, СОЗДАНИЕ, КОНСУЛЬТИРОВАНИЕ И ПУБЛИКАЦИЯ "УМНОГО" КОНТРАКТА
- СОЗДАНИЕ ОФФЛАЙН И ОНЛАЙН-ТРАНЗАКЦИЙ.
- ЗАГРУЗКА ПЕРВИЧНОГО И ОТКРЫТОГО КЛЮЧЕЙ.
- ОБМЕННЫЕ КУРСЫ И КОНСУЛЬТАЦИЯ НА АЗС.
- РАЗРАБОТКА, ТЕСТИРОВАНИЕ И СЕТЕВЫЕ СРЕДЫ (СЕТИ) ЯДРА ETHEREUM
- ВКЛЮЧАЕТ В СЕБЯ 39 ФУНКЦИЙ ФУРЫ.

## 5. Основные концепции, применяемые в платформе Ethereum.

### Что такое блок-цепь?

Блокчейн обычно ассоциируется с Bitcoin и другими крипто-валютами, но это лишь верхушка айсберга, так как он используется не только для цифровых денег, но и может быть использован для любой информации, которая может иметь ценность для пользователей и/или компаний. Эта технология, которая берет свое начало в 1991 году, когда Стюарт Хабер и В. Скотт Сторнетта описали первую работу над цепочкой криптографически защищенных блоков, не была замечена до 2008 года, когда она стала популярной с приходом bitcoin. Но в настоящее время его использование востребовано в других коммерческих приложениях и, по прогнозам, в среднесрочной перспективе будет расти на нескольких рынках, таких как финансовые учреждения или Интернет вещей IoT среди других секторов.

Блок-схема, более известная как блок-схема, представляет собой единую, согласованную запись, распределенную по нескольким узлам (электронным устройствам, таким как ПК, смартфоны, планшеты и т.д.) в сети. В случае с крипто-валютами, мы можем думать об этом как о бухгалтерской книге, в которой регистрируется каждая из транзакций.

Его работа может быть сложной для понимания, если вникнуть во внутренние детали его реализации, но основная идея проста для понимания.

Он хранится в каждом блоке:

- 1.- количество действительных записей или операций,
- 2.- информация, касающаяся этого блока,
- 3.- его связь с предыдущим и следующим блоком через хэш каждого блока – уникальный код, который был бы похож на отпечаток пальца блока.

Поэтому **каждый блок имеет определенное и неподвижное место в цепи**, так как каждый блок содержит информацию из хэша предыдущего блока. Вся цепочка хранится на каждом узле сети, составляющем блок-цепочку, поэтому **точная копия цепочки хранится у всех участников сети**.

## Что такое адрес или учетная запись в платформе Ethereum блок-цепочки?

Это строка из 42 символов в платформе Ethereum, которая представляет собой число в шестнадцатеричной базе, где активы, определенные в Ethereum будут внесены на хранение или отправлены. На других платформах блок-цепочки количество символов учетной записи или адреса может отличаться, например:

0x5d2Acdb34c279Aa6d1e94a77F7b18aB938BFb2bB

## Что такое криптомони?

Это цифровая или виртуальная валюта, предназначенная для функционирования в качестве средства обмена. Она использует криптографию (цифровую безопасность) для обеспечения безопасности и проверки транзакций, а также для контроля за созданием новых блоков конкретного криптомонета.

## Что такое Эфир?

Ether является родной цифровой валютой платформы Ethereum blockchain, децентрализованной платформы, разработанной в 2013 году. Эфир - это устройство, которое можно разделить на более мелкие устройства, называемые WEI, и которое имеет следующую эквивалентность.

1 Эфир = 1,000,000,000,000,000,000 Вэй. (В математическом выражении это  $10^{18}$  ).

## С какими единицами обращаются при использовании эфира?

1	эфирный
$10^3$	Финни
$10^6$	szabo
$10^9$	Шенон или GWEI это используется для GasPrice.
$10^{12}$	баббадж
$10^{15}$	любовное место
$10^{18}$	

## Что такое жетон?

Жетоны - это цифровые активы, которые могут быть использованы в рамках данной экосистемы проекта.

Основное различие между маркерами и крипто-валютами заключается в том, что для работы первых требуется другая платформа блок-цепочки (не их собственная). Ethereum является наиболее распространенной платформой для создания токенов, в основном благодаря своей функции "умного контракта". Токены, созданные на блок-схеме Ethereum, широко известны как ERC-20, хотя существуют и другие, более специализированные типы токенов, такие как ERC-721, используемые в основном для коллекционных ценностей (карты, использование в видеоиграх, произведения искусства и т.д.).

## Что такое газ?

Газ - это стоимость выполнения операции или набора операций в сети Ethereum. Этих операций может быть несколько: от заключения сделки до выполнения интеллектуального контракта или создания децентрализованного приложения.

Другими словами, проще говоря, газ - это устройство для измерения работы, выполняемой в Этериуме.

Как и в физическом мире, в Ethereum также есть рабочие места, которые стоят дороже других: если операция, которую мы хотим выполнить, требует большего использования ресурсов узлами, которые формируют платформу, это заставит газ увеличиться, и наоборот.

Газ служит в первую очередь для выполнения трех функций на платформе Ethereum:

1.- Назначает затраты на выполнение задач; в Ethereum, выполнение задач также имеет затраты. В зависимости от **сложности задачи или скорости, с которой мы хотим, чтобы эта задача была обработана, вычислительная стоимость этой операции будет соответственно выше или ниже**, а количество газа будет увеличиваться или уменьшаться пропорционально.

2.- Обеспечение безопасности системы; Система Ethereum является безопасной системой, и это в значительной степени возможно благодаря газу.

Требуя уплаты комиссии за каждую выполненную транзакцию, платформа гарантирует, что в сети не будут обрабатываться непригодные для использования транзакции. Это помогает сделать блок-цепочку легче, так как это не добавит много бесполезных Мегабайт информации в блок-цепочку.

Кроме того, с помощью газа система также защищена от "спама" и бесконечного использования петель: инструкции по выполнению повторяющихся задач по коду.

Например, если бы Газа не существовало, ничто не помешало бы бесконечному повторению задачи, разрушая систему и делая ее непригодной для использования.

3.- Награда шахтерам; шахтеры являются независимыми внешними системами, распределенными по всему миру, которые отвечают за выполнение транзакций на платформе Ethereum. Когда мы заключаем сделку или исполняем "умный" контракт, мы "платим" определенную сумму за газ.

**Этот газ служит для того, чтобы "платить" шахтерам за используемые ими ресурсы** (оборудование, электричество и время), а **также добавляет вознаграждение** за их работу. Поэтому можно сказать, что газ также помогает поддерживать баланс платформы.

Мы говорим об "оплате" газа - в кавычках - потому что именно на это и уходят операции. Другими словами, вы "оплачиваете" расходы, которые Ethereum тратит на обработку транзакций.

#### **Что такое Цена на газ?**

Газовый блок соответствует выполнению инструкции, например, компьютерному шагу.

Так как же шахтеры "наживаются" на газе, который они приобретают в награду?

Сам по себе газ ничего не стоит и поэтому не может быть заряжен. Для того, чтобы "зарядить" этот отработанный газ, необходимо придать стоимость этим отработанным ресурсам, то есть придать денежную ценность работам, выполняемым шахтерами. Сумма газа, используемая в сделке или интеллектуальном контракте, имеет эквивалентную цену в Ether. Эта цена называется "Цена газа", она обычно назначается шахтерами и варьируется в зависимости от того, насколько занят блок-цепь Ethereum. Обычно он обрабатывается в устройствах GWEI.

#### **Что такое "Газовый лимит"?**

Эти данные указывают на максимальное значение газа, которое может потребляться при совершении сделки, чтобы быть действительным.

Обычно программное обеспечение, используемое для совершения транзакций в сети Ethereum, автоматически вычисляет оценку количества газа, необходимого для проведения транзакции, и сразу же показывает ее нам.

#### **Что такое заправка?**

Это место, где ценности, обрабатываемые шахтёрами, упоминаются для того, чтобы решить на основе консенсуса, какая цена за газ (GasPrice) необходима для осуществления различных видов сделок в блок-поступе Ethereum.

В зависимости от того, сколько GasPrice выбрано, будет взвешено приоритетное время, отведенное на выполнение транзакции, обычно обрабатываются три типа GasPrice: TRADER: ASAP, FAST < 2 минуты, STANDARD < 5 минут. Это средние значения и может варьироваться во времени в зависимости от загруженности (траекторий) основной сети Ethereum.

<https://ethgasstation.info/>

### Что такое Обмен?

Обмен криптоманов - это место встречи, где происходят обмены криптоманов в обмен на деньги или другие криптоманы. В этих онлайн-биржах генерируется рыночная цена, которая обозначает стоимость криптомонет на основе спроса и предложения.

### Что такое обменные курсы?

Это курсы за стоимость эфира в валютах каждой страны. Например, в день создания настоящего руководства стоимость эфира в долларах США составляла 430,94 долл.

### Что такое "Умный контракт"?

В ethereum Умный контракт представляет собой программу на языке программирования под названием Solidity, которая находится в блочной цепи Ethereum, которая имеет инструкции для автоматического выполнения на основе заранее установленных правил, это свойство делает эволюцию во всех существующих блочной цепи, как вы можете создать много Умный контракт адаптированы к различным частным и государственным секторам, что делает более эффективной работы компаний или, где это уместно, адаптироваться к системам или программам всех видов.

### Что такое "nonce"?

Это инкрементный счетчик "шестнадцатеричных чисел", который отслеживает транзакции в каждом направлении или счет, созданный в Ethereum.

### Что такое сделка?

Именно исполнение или передача какого-то не материального актива может быть присвоена заранее установленная стоимость в системе Ethereum, которая впоследствии может быть изменена на материальную стоимость для компании или лица.

### Что такое txHash?

Это шестнадцатеричное число помогает отслеживать результат в деталях каждой транзакции.

### Какие существуют виды сделок?

У вас есть два типа, один из которых транзакция "оффлайн", которая создается без необходимости подключения к основной сети Ethereum может храниться до тех пор, пока вы не решите подключиться к сети Ethereum и выпустить транзакцию, имеют преимущество безопасности, потому что вся транзакция обрабатывается в автономном режиме, что предотвращает любые аномалии, которые могут быть в сетевом подключении. Другая сделка - "он-лайн", которая всегда должна быть подключена к интернету с преимуществами и недостатками безопасности, которые дает соединение.

### Что такое INFURA.io?

Infura.io - это платформа, предоставляющая набор инструментов и инфраструктур, которые позволяют разработчикам легко переносить блок-схемы своих приложений с тестирования на масштабирование, с простым и надежным доступом к Ethereum и IPFS.

### Что такое адрес в сети Ethereum?

Адрес или счет состоит из трех частей, адрес, открытый ключ и частный ключ, эти два ключа представляют собой строку цифр и символов в шестнадцатеричном формате, которые используются для отправки и получения (активный) или эфир (цифровая валюта).

Первичный ключ никогда никому не должен передаваться, так как именно он санкционирует разблокирование остатка (подписывает сделки), хранящегося на счете.

Публичный ключ известен всей общественности и предоставляется любому, так как он является ссылкой для подтверждения того, что сделка является правильной как с точки зрения стоимости, так и того, кому она отправляется.

Примеры:

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

### Что такое Coinsolidation.org?

Это платформа для консолидации крипто-валют, мы создали первые гибридные адреса в централизованном и децентрализованном финансовом мире, используя технологию Blockly, которая является формой визуального программирования без необходимости в продвинутых знаниях программирования, основанного на функциональных расширениях. См. нашу WhitePaper на [сайте www.coinsolidation.org](http://www.coinsolidation.org).

### Какие типы сетей для тестирования и основной сети в блочной цепи Ethereum?

<https://mainnet.infura.io>

Главная сеть, производственная сеть, где все операции осуществляются в режиме реального времени.

<https://ropsten.infura.io>

Тестовая сеть Ropsten позволяет разработчикам блок-цепочки тестировать свою работу в живом окружении, но без необходимости использования реальных ETH-токенов, с алгоритмом, называемым (*Proof-of-Work*).

<https://kovan.infura.io>

Тестовая сеть Кован, которая в отличие от своего предшественника ropsten использует алгоритм под названием Proof of Authority.

<https://rinkeby.infura.io>

Тестовая сеть, использует алгоритм под названием Proof of Authority (Доказательство полномочий). Один из наиболее часто используемых для тестирования.

<https://goerli.infura.io>

Goerli является публичной тестовой сетью для Ethereum, которую движок консенсуса POA (Proof of Authority) поддерживает с различными клиентами. Спам-защита.

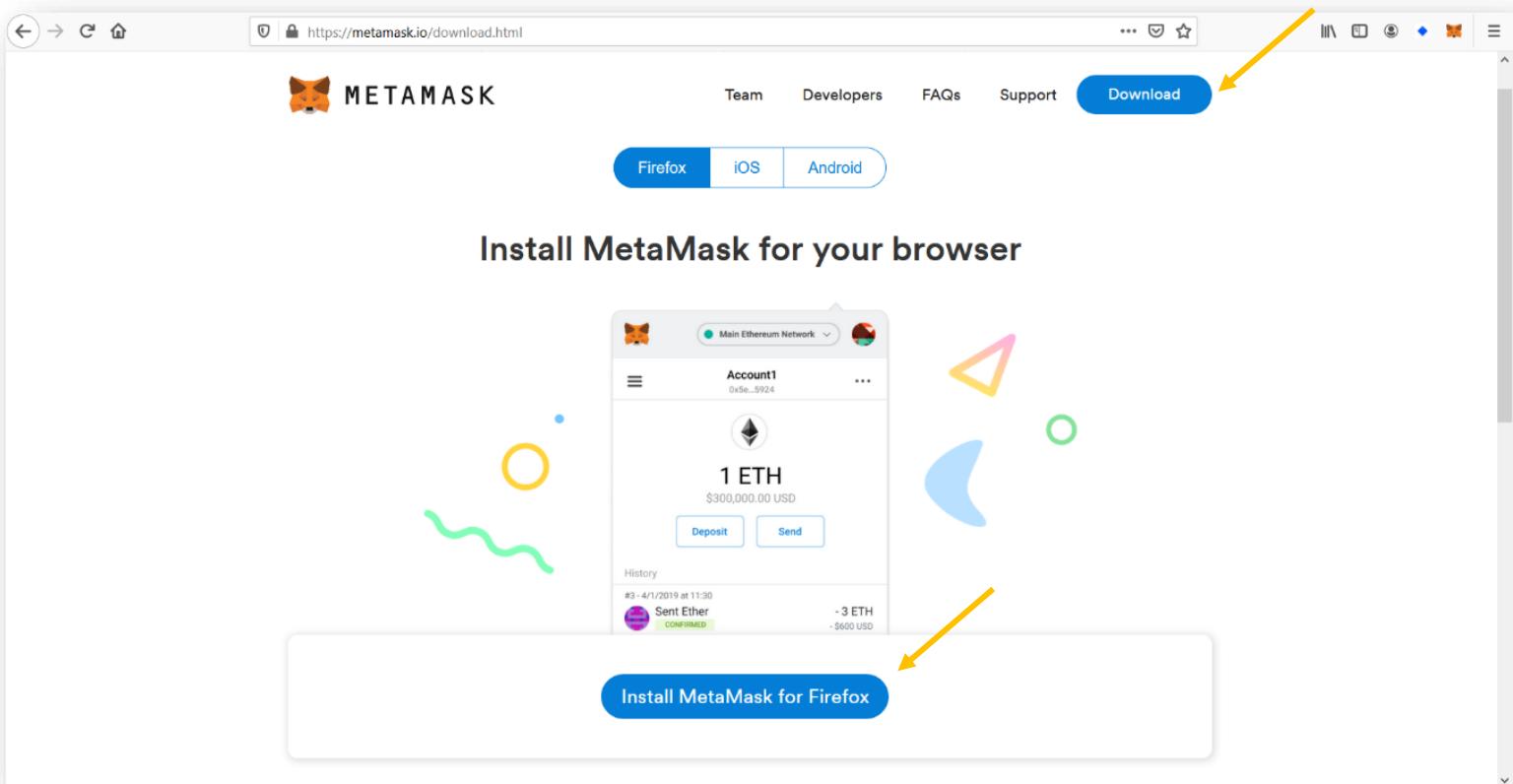
Всего есть 5 сетей на основе блок-цепочки Ethereum, одна для производства или основной и четыре для тестирования, затем мы будем использовать сеть Rinkeby для установки нашей тестовой среды.

## 6. Установка и конфигурация среды расширения и тестирования Ethereum.

Сначала нам нужна тестовая среда. Узнать, как использовать различные функциональные возможности двух расширений, которые будут использоваться для создания, отправки, публикации, просмотра и получения данных из всех транзакций, которые мы совершаем на платформе Ethereum.

Первое, что мы должны сделать, это настроить нашу тестовую среду. Нам нужно будет установить приложение **METAMASK**, которое является кошельком для создания, импорта аккаунтов Ethereum и управления транзакциями из браузера в нашем интернет-браузере, который мы будем использовать Mozilla, а также может быть использован в Chrome.

Зайдите на официальный сайт [www.metamask.io](https://metamask.io) и нажмите на кнопку "Загрузить вниз".



Затем нажмите кнопку "Установить MetaMask for Firefox" и примите опции по умолчанию. После установки мы увидим иконку в правом верхнем углу, где мы увидим уже установленную программу Metamask.

Щелкните по иконке "Метамаска", и появится возможность импортировать существующий аккаунт или создать новый. В нашем случае мы импортируем аккаунт, который мы уже запустили, используя метод "восстановления через семена". Если у вас

его нет, просто создайте новую учетную запись. В этом случае нас попросят назначить нам пароль в любом случае.

Позже мы входим с "паролем" и можем проверить, какой баланс у нас есть, логически, если он новый, то баланс будет нулевым.

The screenshot shows a web browser window with a Google search results page. Overlaid on the browser is the Metamask extension interface. The Metamask interface displays the following information:

- Network: Ethereum Mainnet
- Address: sender 0x4B73...4A58
- Balance: 0.0719 ETH (\$31.29 USD)
- Buttons: Buy, Send, Swap
- Links: Assets, Activity
- Other: Add Token

A yellow arrow points from the text "Позже мы входим с "паролем" и можем проверить, какой баланс у нас есть, логически, если он новый, то баланс будет нулевым." to the "Ethereum Mainnet" dropdown in the Metamask interface.

Теперь мы рассмотрим, как мы собираемся внести несколько эфиров (цифровых монет) на наш тестовый аккаунт Rinkeby.

Вверху у нас есть возможность выбрать тип сети, которую мы будем использовать, по умолчанию, когда вы вводите ее помещает вас в "Ethereum Mainnet", однако, когда вы нажимаете на нее, мы можем просмотреть все дополнительные сети, которые мы можем выбрать, в нашем случае мы выбираем сеть Rinkeby.

The screenshot shows a web browser window with a Google search results page. Overlaid on the browser is the Metamask extension interface. The Metamask interface displays the following information:

- Network: Rinkeby Test Network
- Address: Not connected
- Balance: 1063.5384 ETH (Ropsten Test Network)
- Text: The default network for Ether transactions is Main Net.
- Links: Networks, Ethereum Mainnet, Ropsten Test Network

A yellow arrow points from the text "Вверху у нас есть возможность выбрать тип сети, которую мы будем использовать, по умолчанию, когда вы вводите ее помещает вас в "Ethereum Mainnet", однако, когда вы нажимаете на нее, мы можем просмотреть все дополнительные сети, которые мы можем выбрать, в нашем случае мы выбираем сеть Rinkeby." to the "Rinkeby Test Network" dropdown in the Metamask interface.

Выполните поиск по приложению "Termux", выберите его и запустите процесс установки.

Следующим шагом является внесение эфиров в нашу тестовую учетную запись сети Rinkeri.

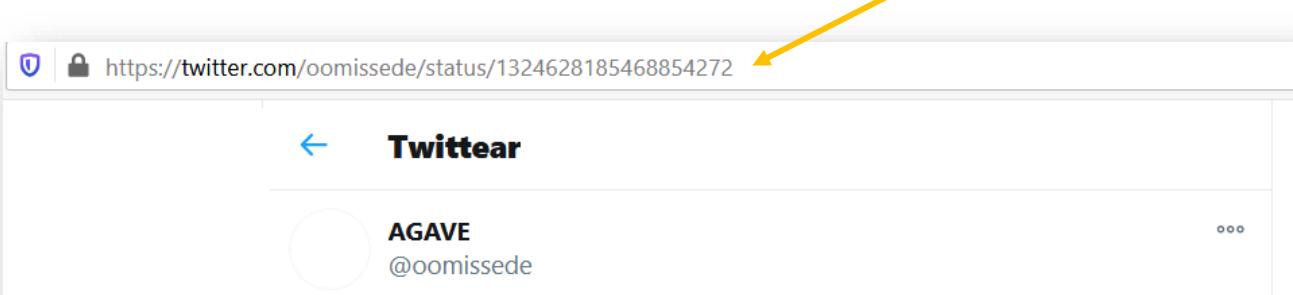
**ВАЖНОЕ ЗАМЕЧАНИЕ:** эфиры (цифровая валюта), которые мы вносим на наш счет со ссылкой на тестовую сеть Rinkery, не имеют значения на рынке криптографии, они будут использоваться нами только для тестирования программного обеспечения. Всегда проверяйте, над какой сетью мы работаем, чтобы избежать ошибок в транзакциях.

Для получения эфира для тестирования необходимо выполнить следующую процедуру.

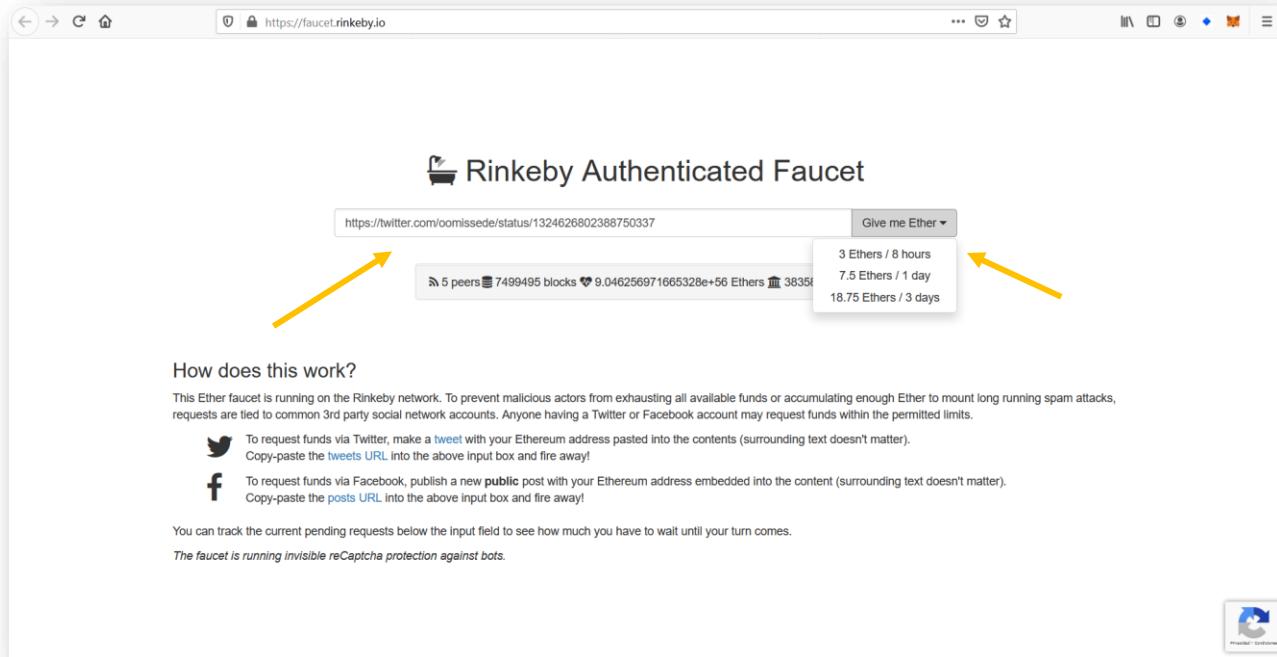
В любом аккаунте twitter, который у вас есть, мы должны будем ввести twitter и создать комментарий, который включает только аккаунт и / или адрес, который у нас есть в Metamask, то мы должны будем скопировать ссылку на комментарий, так как у нас есть это, мы перейдем на следующую ссылку, чтобы закрепить наш аккаунт.

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

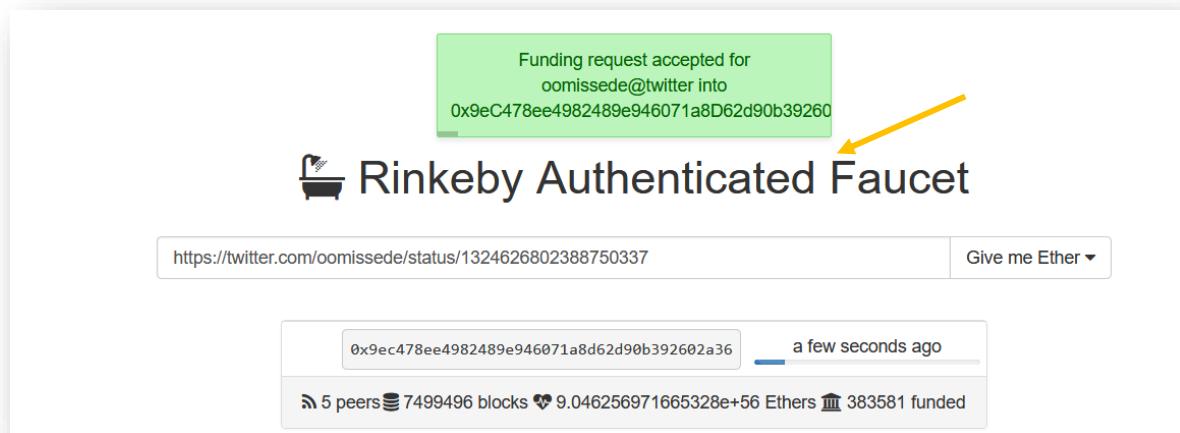
Мы создали комментарий в твиттере и скопировали ссылку из комментария.



На сайте <https://faucet.rinkeby.io/> мы копируем ссылку в твиттере и на кнопке "Give me Ether" выбираем желаемую сумму.



Наконец, если все правильно, мы дадим объявление о том, что депозит был принят, и в зависимости от загруженности сетевой системы Rinkeby депозит будет через несколько минут или может занять больше времени.



Теперь, когда на нашем счету есть Ethers, мы можем начать тестирование на платформе Ethereum.

У нас есть два расширения для взаимодействия с блочной цепью Ethereum.

Расширение **совместного уплотнения**. **BlockoinETHEREUM.aix** содержит функции для выполнения следующих функций:

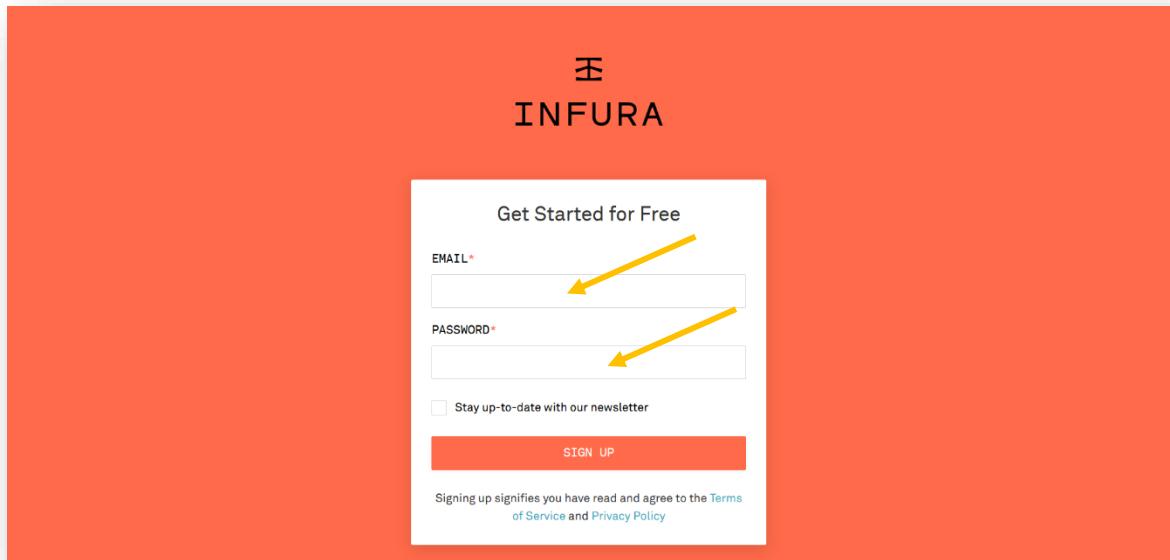
- Создание новых учетных записей (адресов) в блок-цепочке Ethereum (privateKey, publicKey, Address)
- Хранение учетных данных (адресов) в двоичных файлах.
- Импорт учетных записей (адресов) бинарных файлов
- Получение счета (адреса) через privateKey.
- Создание и отправка транзакций между счетами (адресами) "онлайн".
- Создание, подписание и отправка оффлайн транзакций PushRaw.
- Консультация по деталям сделки Tx.
- Проверка баланса по адресам и интеллектуальным контрактам.
- Компилятор для Smart контрактов.
- Создание, публикация и исполнение "умных" контрактов.
- Создание, публикация и исполнение жетона ERC20 (криптомонета).
- Получение ABI кода от "умного" контракта.
- Проверка сетевого подключения.
- Запрос стоимости эфира на криптомаркете в любой стране мира (родная валюта страны) - Обменные курсы.
- Консультация GasPrice.
- Консультация "nonce" конкретного счета.

Удлинение **монетного осаждения**. **BlockoinINFURA.aix** предоставляет нам 40 функциональных возможностей платформы Infura.io. За более подробной

информацией обращайтесь, пожалуйста, к документации json-rpc непосредственно на сайте <https://infura.io/docs>.

Для использования расширения INFURA необходимо создать учетную запись на сайте infura.io, так как нам нужен API KEY, чтобы иметь возможность отправлять запросы в сеть Ethereum, а также иметь возможность использовать тестовые сети Ethereum.

Как мы открываем счет, все просто, как показано ниже.



После создания учетной записи, мы вводим и можем иметь KEY API различных сетей. Мы идем в левую верхнюю часть и нажимаем на Ethereum, затем мы видим проекты, мы **создаем** новый проект и знакомимся с этим проектом.

The screenshot shows the Infura dashboard for the Ethereum project. The sidebar on the left has tabs for 'ETHEREUM' (which is selected), 'FILECOIN', 'DOCS', 'COMMUNITY', and 'SUPPORT'. The main area shows a project named 'COINSOLIDATION' created on 'CREATED SEPTEMBER 12, 2020' with a 'STATUS' of 'Active' and 'REQUESTS TODAY' at 0. There is a line graph showing request trends over the previous 7 days. To the right, there is a 'PLAN' section for 'Core' with '100,000 Requests/Day' and a button to 'CHANGE PLAN'. Below that is a link to 'See how other customers are using our Ethereum API'.

В рамках проекта мы будем иметь данные KEY API и различных сетей, которые мы можем использовать.

В рамках проекта мы можем просмотреть API KEY (PROJECT ID) и выбрать, над какой сетью мы будем работать или полные ссылки ENDPOINTS для выбора.

COINSOLIDATION

REQUESTS SETTINGS

SAVE CHANGES

KEYS

PROJECT ID: 39c3711abdad7f663

PROJECT SECRET: 1ac27e75434133134

ENDPOINTS: Mainnet

Mainnet

Ropsten

Kovan

Rinkeby

Görli

SECURITY

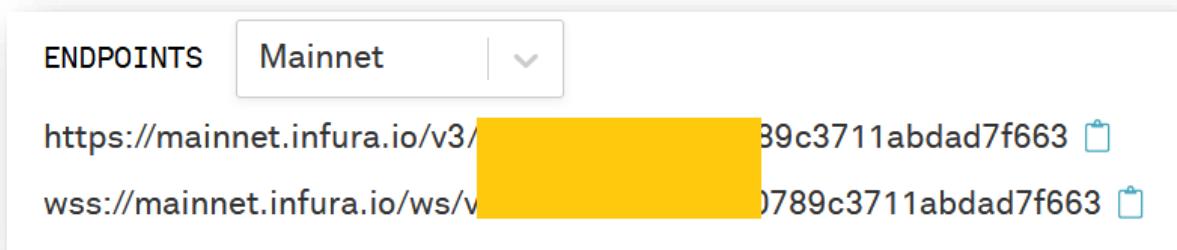
PROJECT SEC

JWT REQUIRED

Require project secret for all requests

Require JWT for all requests

Например, для использования основной сети Ethereum мы бы взяли следующую лигу:

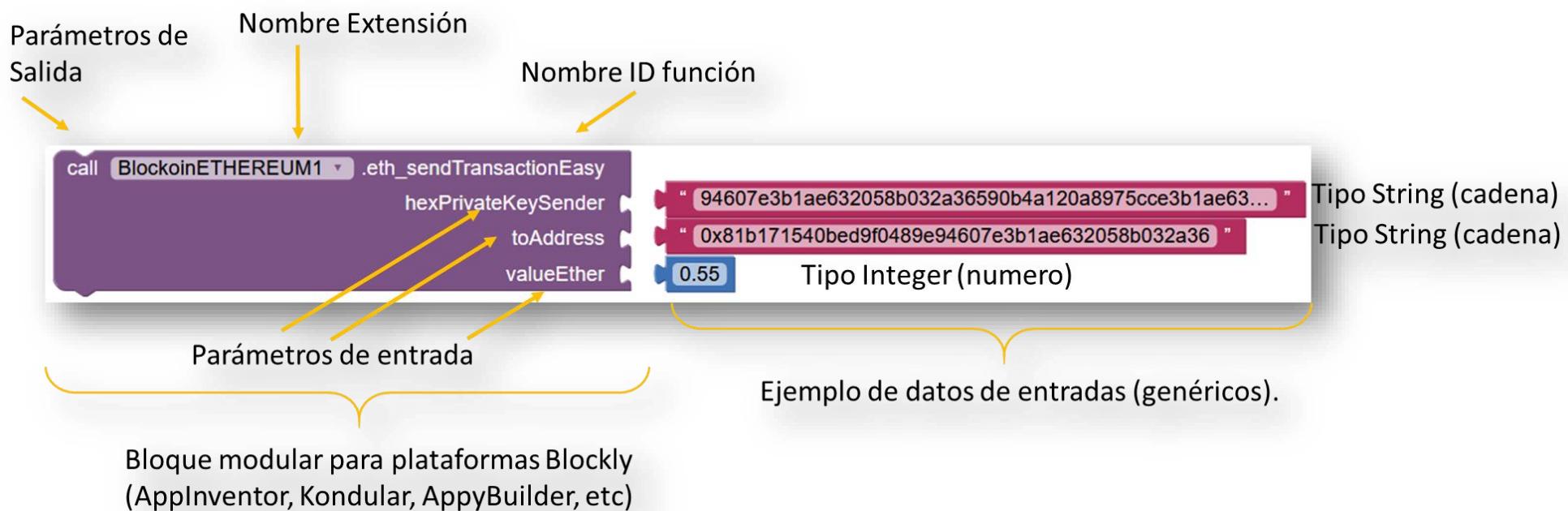


**ПРИМЕЧАНИЕ:** Расширения были протестированы на системах AppInventor, Kondular, Thunkable и AppyBuilder.

## 7. Определение и использование блоков (общая функция)

Мы начнем с объяснения распределения данных, которые будут иметь все блоки, их синтаксис использования и настройки.

В следующем примере мы видим модульный блок и его входные и выходные параметры, а также типы входных данных, эти данные могут быть типа String (строка символов) или Integer (целое или десятичное число). Мы показываем, как он используется и настраиваем его для правильного функционирования.



Каждый блок модуля будет иметь свое описание и будет назван в случае, если он имеет обязательные или необязательные зависимости от других блоков, используемых в качестве входных параметров, будет объявлен процесс интеграции.

## 8. Расширение Ethereum (EEE) функций и событий.

\*\*\*Тестирование сети, которую мы будем использовать **Rinkeby**, в качестве urlNetwork, когда вы хотите совершать реальные транзакции, вам нужно только изменить сеть urlNetwork на **mainnet**.

Блок для проверки подключения к Интернету - (**CheckInternetConnection**).

call BlockoinETHEREUM1 .CheckInternetConnection

Входные параметры: Не применимо.

Выходные параметры: возвращает "True" при наличии соединения или "False" при отсутствии соединения.

Описание: Блок для проверки подключения к Интернету и отправки данных (транзакций).

Блок для генерации **нового** "автономного" адреса - (**GenerateNewAddressEthereum**).

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

Входные параметры: **phraseasHex <String>**.

Выходные параметры: Событие (**OutputGenerateNewAddressEthereum**)

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

when BlockoinETHEREUM1 .OutputGenerateNewAddressEthereum  
privKeyEther pubKeyEther addressEthereum  
do

Описание: Создайте новый этереум адрес (счет) на основе фразы или последовательности чисел. Новый адрес может быть создан без подключения к сети или Интернету - "Offline".

Блок для генерации нового адреса "Online" - (**GenerateNewAddressEthereum**).

call BlockoinETHEREUM1 .GenerateNewAddressEthereumAPI

Входные параметры: Не применимо.

Выходные параметры: возврат в формате данных JSON; privateKey, publicKey, адрес.

Выходной пример:

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

Описание: Создайте новый адрес эфира (аккаунт). Необходимо иметь доступ или подключение к интернету, так как генерация осуществляется через сервис REST API - "Online".

Блокировать для генерации нового "Offline" адреса и сохранения публичных и закрытых ключей в двоичных файлах - (**GenerateNewAddressEthereumStoreKeys**)

call BlockoinETHEREUM1 .GenerateNewAddressEthereumStoreKeys  
 pathFilePrivateKey <String> "/mnt/sdcard/privateKey.bin"  
 pathFilePublicKey <String> "/mnt/sdcard/publicKey.bin"

Входные параметры: **pathFilePrivateKey<String>** , **pathFilePublicKey<String>**.

Выходные параметры: Событие (**OutputGenerateNewAddressEthereumStoreKeys**)

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

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

Описание: Создает новый случайный адрес этереума (учетной записи) и хранит публичный и закрытый ключи в двоичных файлах, которые будут использоваться для импорта и экспорта данных учетной записи. Новый адрес может быть создан без подключения к сети или Интернету - "Offline".

Блок для генерации открытого ключа - (GeneratePublicKeyHexFromPrivateKeyHex).

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

Входные параметры: `hexPrivateKey <String>`.

Выходные параметры: Событие (OutputGeneratePublicKeyHexFromPrivateKeyHex)

Выводы: `адрес<String>` , `publicKeyHex<String>`.

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

Описание: Создает открытый ключ на основе ввода закрытого ключа.

Блок для **получения** баланса по адресу - (GetBalanceAddrEthereum).

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

Входные параметры: `hexPrivateKey <String>`.

Выходные параметры: Событие (OutputGeneratePublicKeyHexFromPrivateKeyHex)

Выходные данные: `баланс<String>` , `total_received<String>` , `total_sent<String>` , `unfirmed_balance <String>` , `final_balance<String>` , `n_tx<String>` , `unfirmed_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
```

**Описание:** Отображает балансовый отчет и подробные данные по счету (адресу).

Блокировка для проверки, активирован ли у вашего мобильного телефона сетевой интерфейс - (**GetDataNetworkConnection**).

call BlockchainETHEREUM1 .GetDataNetworkConnection

Входные параметры: Не применимо

Выходные параметры: Событие (OutputGeneratePublicKeyHexFromPrivateKeyHex)

Выходные данные: interfaseName<String>, isConnected<String>.

when BlockchainETHEREUM1 .OutputGetDataNetworkConnection

interfacename isconnected

do

Описание: Отображает название мобильного интерфейса и сообщает, активирован интерфейс или нет.

Блокировка для подписания сделки "Offline" - (SignerGenericPushRawTransactionOffline)

```
call BlockinETHEREUM1 .SignerGenericPushRawTransactionOffline
    urlNetwork " https://rinkeby.infura.io/v3/...440789c3... "
    hexPrivateKeySender " 9eC478ee49824890b4a120a8975cce3b1ae632058b0301f8... "
    nonceNumber get global nonce
    gasPrice " 250000000000 "
    gasLimit 21000
    toAddress " 0x92a2f157d5aec3fa79f92995fea148616d82c5ef "
    valueWei " 10000000000000000000 " x " 0.01 "
```

Необходимая зависимость(и): Блок (eth\_getTransactionCount), Блок (eth\_SendRawTransactionInfura)

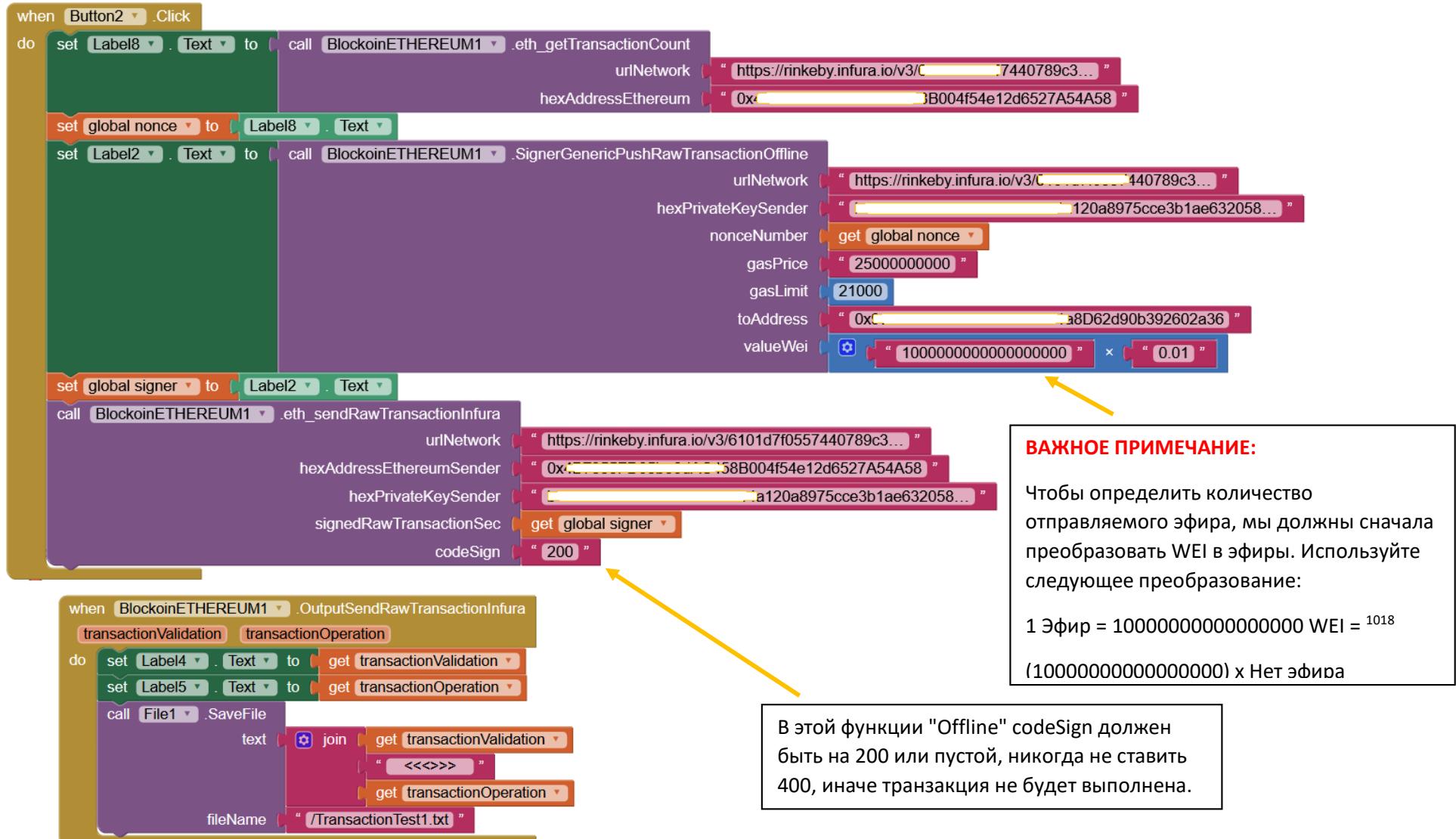
Параметры: urlNetwork <String>, hexPrivateKeySender <String>, nonceNumber <String>, gasPrice <String>, gasLimit <Integer>, toAddress <String>, valueWEI <Integer>.

Выходные параметры:

Выходные данные: Подписанная транзакция для отправки. <Стринг>.

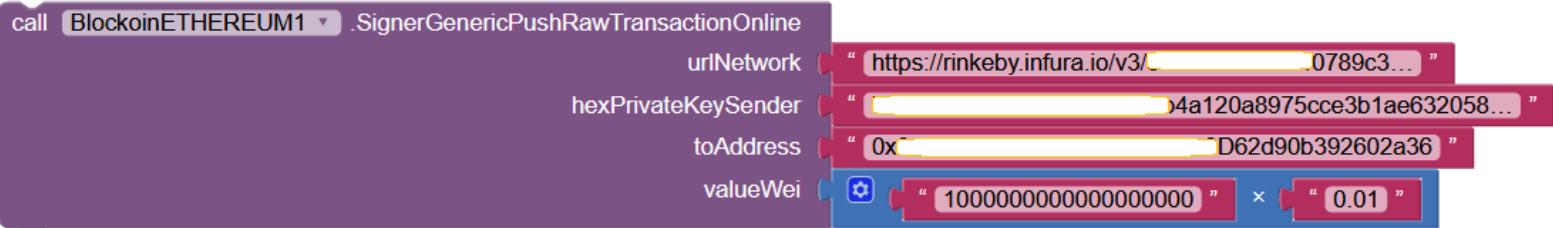
Описание: Готовит новую транзакцию для отправки (зашифрованную и подписанную).  
Это может быть обработано без подключения к сети или Интернету - "Offline".

Пример полного использования с блочными зависимостями (`SignerGenericPushRawTransactionOffline`).



Блокировка для подписания "Онлайн" сделки -

*(SignerGenericPushRawTransactionOnline)*



Обязательная единица(и): Блок (*eth\_SendRawTransactionInfura*).

Входные параметры: *urlNetwork <String>*, *hexPrivateKeySender <String>*, *toAddress <String>*, *valueWEI <Integer>*.

Выходные параметры: События, используемые в следующем порядке (*OutputSignerGenericPushRawTransactionOnline*) и (*OutputSendRawTransactionInfura*).

Выходные данные: **подписанный<String>**, **код<String>**.

when [BlockoinETHEREUM1 v].OutputSignerGenericPushRawTransactionOnline

**signed** **code**

do call [BlockoinETHEREUM1 v].eth\_sendRawTransactionInfura

urlNetwork "https://rinkeby.infura.io/v3/40789c37..."

hexAddressEthereumSender "0x9458B004f54e12d6527A54A58"

hexPrivateKeySender "04a120a8975cce3b1ae632058..."

signedRawTransactionSec get signed

codeSign get code

Выходит блок *eth\_sendRawTransactionInfura*: *transactionValidation<String>*, *transactionOperation<String>*.

when [BlockoinETHEREUM1 v].OutputSendRawTransactionInfura

**transactionValidation** **transactionOperation**

do set Label4 Text to get transactionValidation

set Label5 Text to get transactionOperation

call [File1 v].SaveFile

text join get transactionValidation

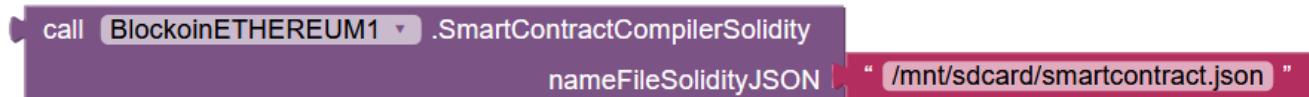
"<>>>"

get transactionOperation

fileName "/TransactionTest1.txt"

Описание: Готовит новую транзакцию для отправки (зашифрованную и подписанную). Требуется подключение к сети или Интернету - "Онлайн".

Блок для **составления** Smart-контракта "Онлайн" - (**SmartContractCompilerSolidity**).



Входные параметры: **nameFileSolidityJSON <String>**.

Выходные параметры: Показывает скомпилированный "умный" контракт, эта функция помогает проверить, хорошо ли он написан, прежде чем публиковать "умный" контракт в сети Ethereum.

Выходы: **скомпилированный код**.

Смарт-контракт должен быть в формате JSON.

Пример базового "умного" контракта на языке Solidity.

```
прочность прагмы ^0.5.0;

рекомендовано договор {
владелец адреса;
функция mortal() { owner = msg.sender; }
функция kill() { если (msg.sender == owner) suicide(owner); } }
контрактная встреча смертельна.
приветствие струны;
функция greeter(string _greeting) public { greeting = _greeting; }
функция greet() возвращает константу (string) {return greeting;}
```

Пример предыдущего "умного" контракта в формате JSON с комментариями.

```
# Проверьте компиляцию прочности через неопубликованный тест.
# Используя пример "приветственной" контрактной солидности, "привет мир"
Этериума.
```

**Файл: smartcontract.json**

```
{
"солидность": "контрактная смерть" {\n /* Определение владельца\n переменной типа address*/\n n address owner;\n /* эта функция\n выполняется при инициализации и устанавливает владельца контракта */\n
```

```

н функция mortal() { owner = msg.sender; }\\      n /* Функция возврата
средств по контракту */\\      n функция kill() { if (msg.sender == owner)
suicide(owner); }\\\\n\\ncontract greeter is mortal {\\      n /* define
variable greeting of the type string */\\      n string greeting;\\      n /*
она выполняется при выполнении договора */\\      n функция greeting(string
_greeting) public {\\n          greeting = _greeting;\\n    }\\      n /*
*/\\      n функция greet() возвращает константу (строку) {\\n          return
greeting;\\n    }\\n  },
"Парамедики". ["Hello BlockCypher Test"]
}

```

**ВАЖНОЕ ЗАМЕЧАНИЕ:** Формат JSON всегда должен иметь разрыв строк в конце каждой строки.

Пример скомпилированного вывода Smartcontract.

```

[
{
  "имя": "u003cstdin\u003e:greeter",
  "сольдность": "контрактная смерть" {\\n /* Определение владельца
переменной типа address*/\\      n address owner;\\      n /* эта функция
выполняется при инициализации и устанавливает владельца контракта */\\
n функция mortal() { owner = msg.sender; }\\      n /* Функция возврата
средств по контракту */\\      n функция kill() { if (msg.sender == owner)
suicide(owner); }\\\\n\\ncontract greeter is mortal {\\      n /* define
variable greeting of the type string */\\      n string greeting;\\      n /*
она выполняется при выполнении договора */\\      n функция greeting(string
_greeting) public {\\n          greeting = _greeting;\\n    }\\      n /*
главная функция */\\      n функция greet() возвращает константу (строку)
{\\n          return greeting;\\n    }\\n  },
  "мусорное ведро": "606060405260405161023e38038061023e8339810160405280510160008054600160a060
020a031916331790558060016000509080519060200190828054600181600116156101000
203166002900490600052602060002090601f016020900481019282601f10609f57805160
ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565b50505
061016e806100d06000396000f35b828001600101855582156076579182015b8281111560
7657825182600050559160200191906001019060b0565b509056606060405260e060020a6
00035046341c0e1b58114610026578063cf3ae321714610068575b005b6100246000543373
ffffffffffffffffff16ff5b6100c9600060609081526001805460
a06020601f600260001961010086881615020190941693909304928301819004028101604
0526080828152929190828280156101645780601f10610139576101008083540402835291
60200191610164565b6040518080602001828103825283818151815260200191508051906
0200190808383829060006004602084601f0104600f02600301f150905090810190601f16
80156101295780820380516001836020036101000a031916815260200191505b509250505
06
  "abi": [
    {
      "постоянная": ложь,
      "входы": [],
      "имя": "убить",
      "выходы": [],
      "тип": "функция"
    }
  ]
}

```

```

},
{
  "постоянная": правда,
  "входы": [],
  "имя": "приветствие",
  "выходы": [
    {
      "имя": "",
      "тип": "строка"
    }
  ],
  "тип": "функция"
},
{
  "входы": [
    {
      "имя": "_greeting",
      "тип": "строка"
    }
  ],
  "тип": "строитель"
}
],
"Парамедики": [
  "Привет, блокшифровка".
]
},
{
  "имя": "смертный",
  "согласие": "контрактная смерть" {
    \n /* Определение владельца
    переменной типа address*/
    n address owner;
    \n /* эта функция
    выполняется при инициализации и устанавливает владельца контракта */
    n функция mortal() { owner = msg.sender; }
    \n /* Функция возврата
    средств по контракту */
    n функция kill() { if (msg.sender == owner)
      suicide(owner); }
    \n\ncontract greeter is mortal {
      \n /* define
      variable greeting of the type string */
      n string greeting;
      \n /*
      она выполняется при выполнении договора */
      n функция greeting(string
_greeting) public {
        \n         greeting = _greeting;
      }
      \n /*
      главная функция */
      n функция greet() возвращает константу (строку)
      \n         return greeting;
    }
    \n
    "мусорное ведро":
    "606060405260008054600160a060020a03191633179055605c8060226000396000f36060
    60405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffffffffffff
    ffffffffffffffffffffff90811691161415605a5760005473fffffffffffff
    ffffffffffffff16ff5b56",
  "abi": [
    {
      "постоянная": ложь,
      "входы": [],
      "имя": "убить",
      "выходы": [],
      "тип": "функция"
    },
    {
      "входы": []
    }
  ]
}

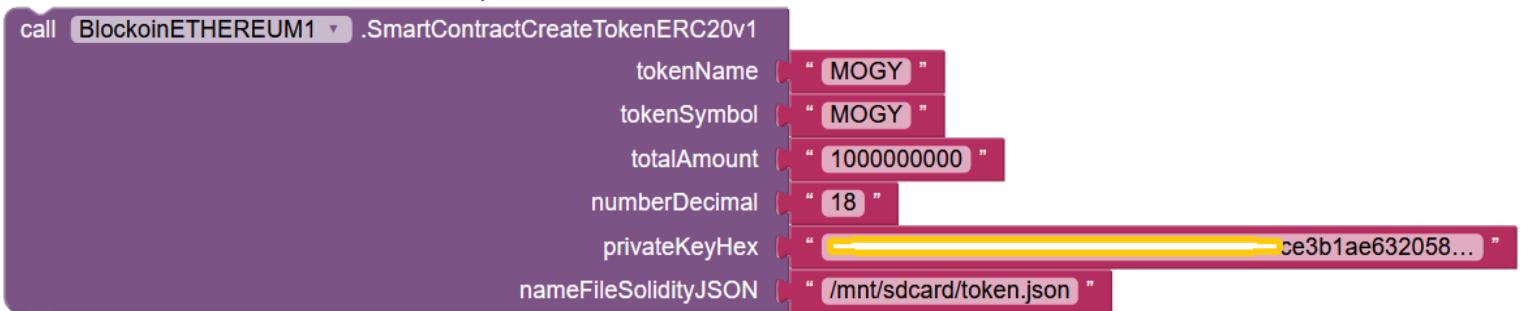
```

```

        "тип": "строитель"
    },
],
"Парамедики": [
    "Привет, блокшифровка".
]
}
]

```

Блок для компиляции, создания и публикации ERC20 Token -

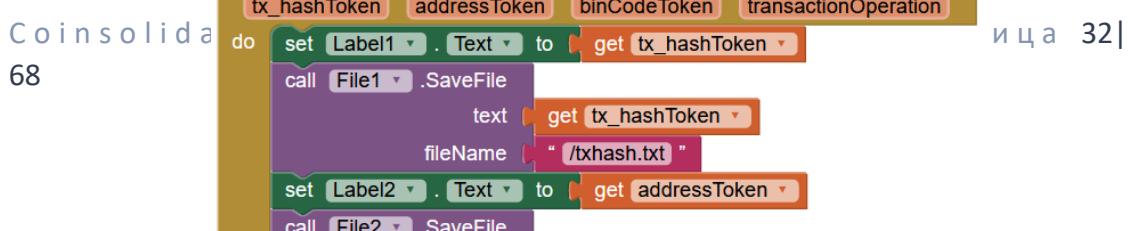


Обязательная единица(и): Блок (`CreateTestingFile`). **ВАЖНО:** Сначала Вы должны использовать этот блок, чтобы убедиться, что путь правильный, это потому, что если Вы не дадите правильный путь в блоке (`SmartContractCreateTokenERC20v1`), то создание токена не будет выполнено, потому что входной параметр "nameFileSolidityJSON" используется для создания временного файла.

Входные параметры: `tokenName <String>`, `tokenSymbol <String>`, `totalAmount <String>`, `numberDecimal <String>`, `privateKeyHex <Integer>`, `nameFileSolidityJSON <String>`. Этот файл является правильным путем для **создания временного** файла, вы должны убедиться, что путь правильный, чтобы проверить, что файл создан, вы можете использовать Block (`CreateTestingFile`) после его использования проверить, что он был создан успешно.

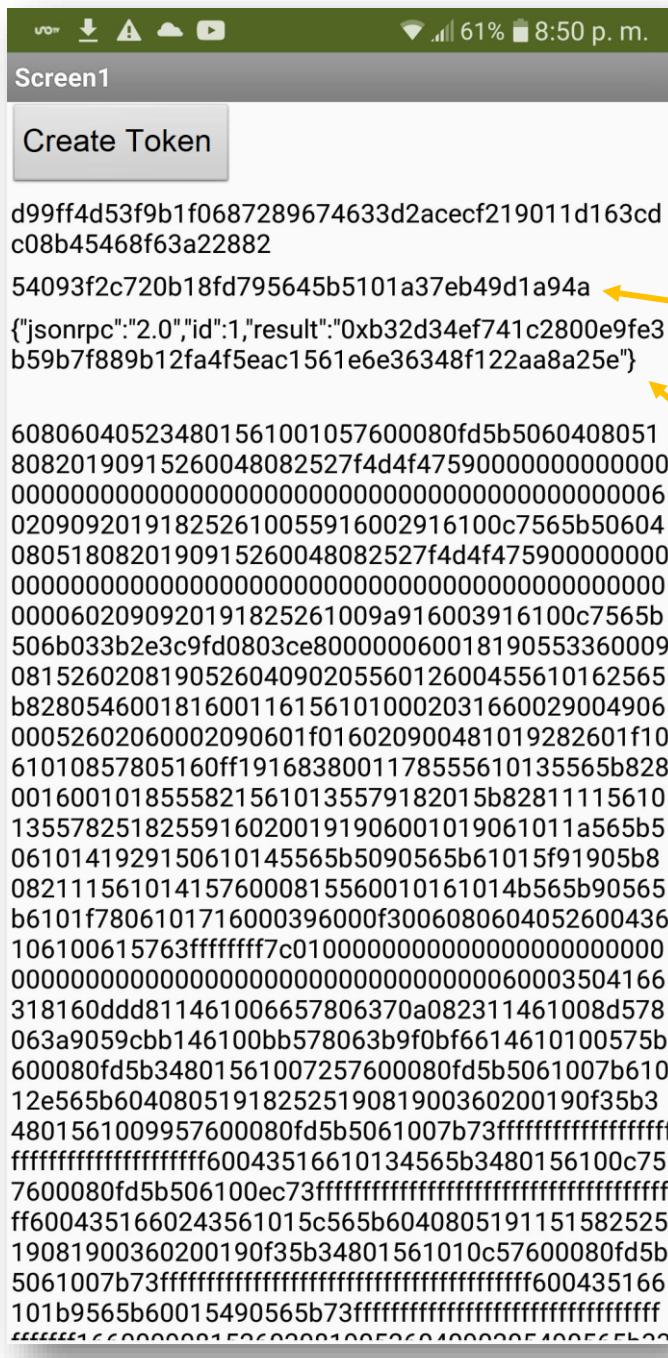
Выходные параметры: Событие (`OutputSmartContractTokenERC20v1`).

Выводы: `tx_hashToken<String>` , `addressToken<String>` , `binCodeToken<String>` , `trasactionOperation<String>`.



Описание: Интеллектуальный контракт "Token ERC20" - актив, опубликованный в сети Ethereum. Версия 1 (v1) уже имеет набор параметров "Предельный расход газа" на уровне 500 000 WEI.

Пример вывода предыдущей функции SmartContractCreateTokenERC20v1.



Транзакция (Tx\_hash) создания сети Ethereum. С этим можно ознакомиться по адресу: [www.etherscan.io](http://www.etherscan.io).

Адрес нового контракта, который ссылается на новый созданный токен ERC20.

Транзакция (Tx\_hash) валидированной операции в системе CoinSolidation.org

Это можно найти по адресу: [www.etherscan.io](http://www.etherscan.io).

Двоичный код (BIN) нового контракта токена, который был обработан в EVM (Ethereum Virtual Machine).

## 9. Шаги по созданию крипто-токена или крипто-мони.

Шаг 1.

[Coinsolidation.org](http://Coinsolidation.org)

Убедитесь, что временный путь на мобильном устройстве, где был создан Smart контракт, действителен и файл может быть успешно создан. Это делается с помощью Block (**CreateTestingFile**). Убедитесь, что тестовый файл, указанный во входном файле "pathTestFile", создан, обычно путь задается: [/mnt/sdcard/name\\_file.txt](#)

Шаг 2 (необязательно).

На данном этапе мы проверим, достаточно ли средств на счете (адресе), с которого будет списана операция, для выполнения операции по созданию и публикации "умного" контракта. Это можно проверить с помощью Блока (**eth\_VerifiBalanceForTransaccionSmartContract**). Использование этого блока не является обязательным, поскольку блоки, генерирующие **SmartContractCreateTokenERC20v1** или **SmartContractCreateTokenERC20v2**, уже содержат эту проверку изнутри.

Шаг 3.

Выберите, какой блок использовать для создания маркера ERC20, у вас есть два варианта:

a.- Блок **SmartContractCreateTokenERC20v1** уже имеет неявное значение Gas Limit со значением 500 000 WEi.

b.- Блок **SmartContractCreateTokenERC20v2** имеет возможность настроить Gas Limit в соответствии с потребностями конечного пользователя или разработчика. Следует отметить, что при очень низком пределе газа менее 350 000 Вэй, вполне возможно, что Smart contrit.

Шаг четвертый.

Используйте блоки **SmartContractCreateTokenERC20v1** или **SmartContractCreateTokenERC20v2**, убедившись, что при использовании входной переменной "nameFileSolidityJSON" она равна входной переменной "pathTestFile" блока (**CreateTestingFile**), уже проверенной на шаге 1.

Шаг пятый.

Перед выполнением создания маркера ERC20 с любым из блоков **SmartContractCreateTokenERC20v1** или **SmartContractCreateTokenERC20v2** рекомендуется сохранить значения (результаты) Event для сохранения результатов (tx\_hashToken, addressToken, binCodeToken, transactionOperation). См. пример вывода предыдущей функции **OutPutSmartContractCreateTokenERC20v1**.

Шаг шестой.

Выполните создание жетона ERC20 и опубликуйте его для продажи. См. раздел 10.

## 10. Как выставить на продажу новый актив или маркер склепа (Token ERC20).

С тех пор как мы создали ERC20 - Cryptomoney Token (см. SmartContractCreateTokenERC20v1 Block или с помощью SmartContractCreateTokenERC20v2 Block), мы должны загрузить его на какой-нибудь Exchange-сервер, чтобы любой человек в мире мог его купить. Exchange - это сайт в Интернете, на котором публикуются новые маркеры.

Обмены делятся на два типа: централизованные и децентрализованные. Главное отличие состоит в том, что одним из них управляет и проверяет какой-то международный орган (централизованный), а у децентрализованных нет никого, кто бы работал с аудиторами. Хотя это может дать больше уверенности в себе, реальность такова, что в последнее время децентрализованные структуры стали более сильными и используются в основном без серьезных проблем.

Одна из лучших практик при работе с любыми активами заключается не в том, чтобы все они находились на одном счете, а в том, чтобы распределить их по нескольким счетам для обеспечения безопасности как частных, так и публичных ключей.

В нашем случае мы будем использовать децентрализованный Exchange, но уже с неплохой историей, мы будем использовать [www.forkdelta.app](http://www.forkdelta.app).

На данный момент мы уже должны установить приложение для браузера (Mozilla или Chrome) METAMASK [www.metamask.io](http://www.metamask.io), потому что Exchange, чтобы посетить вашу страницу [www.forkdelta.app](http://www.forkdelta.app), должен соединиться с учетной записью, которая у нас есть Ethereum.

Важным моментом является то, что наш Ethereum аккаунт, который у нас уже есть в METAMASK, должен иметь баланс более или менее 10 USD, это потому, что когда мы опубликуем наш новый ERC20 Token, который мы создали с помощью SmartContractCreateTokenERC20v1 Block или с помощью SmartContractCreateTokenERC20v2 Block, мы должны будем заплатить за транзакцию, чтобы опубликовать его на бирже.

Для того, чтобы использовать Exchange [www.forkdelta.app](http://www.forkdelta.app), у нас должны быть следующие токен-данные, которые мы хотим опубликовать для продажи на Exchange.

Адрес новой жетоны ERC20, которую мы создали ранее.

**0x54093F2C720b18Fd795645b5101A37EB49d1A94a**

Количество десятицентов, которые мы используем в наших прикосновениях.

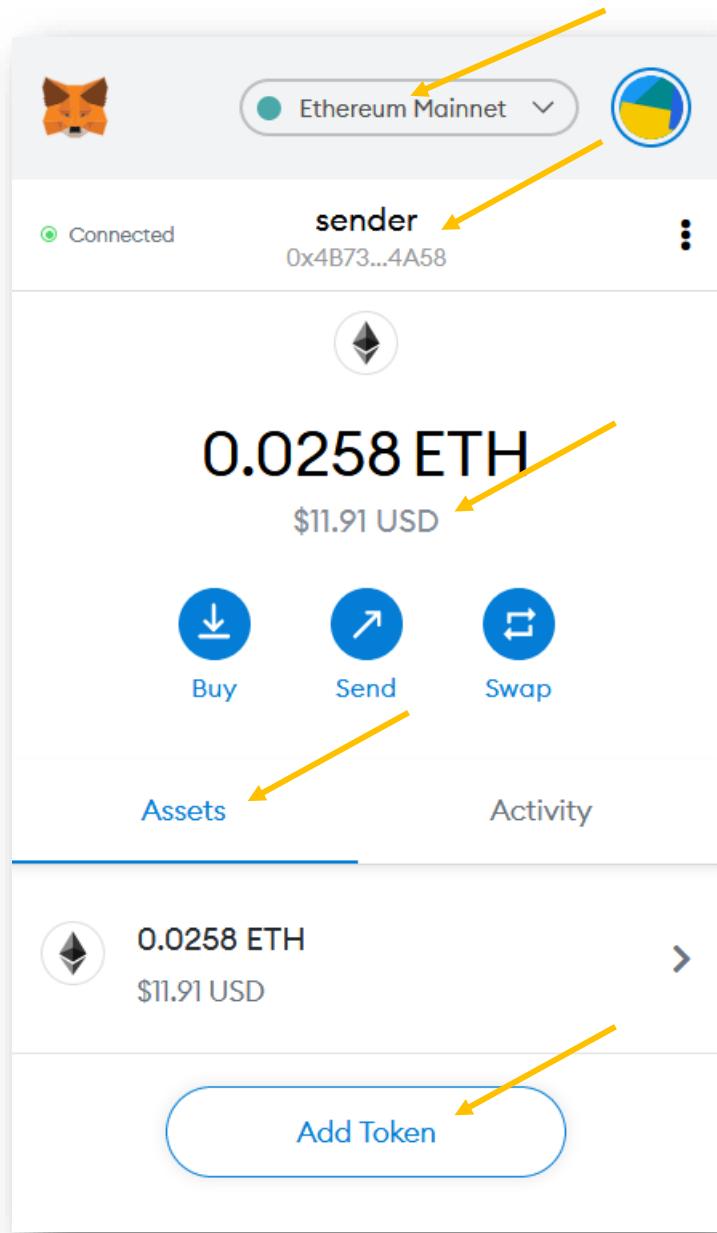
**18**

Имя, которое идентифицирует токен.

**MOGY**

Начнем с проверки того, что созданная нами лексема имеет упомянутые выше параметры, и что это те же параметры, которыми она была изначально создана.

Перейдем в **METAMASK** и сначала убедимся, что мы находимся на том аккаунте, с помощью которого создали токен. Затем перейдите в нижнюю часть и нажмите на кнопку "Добавить жетон".



Теперь мы добавим наш новый токен, чтобы увидеть ваш текущий баланс. После нажатия на кнопку "Пользовательский токен" в верхней части страницы введите запрашиваемую информацию в следующие поля и затем нажмите на кнопку "Далее". После этого появится наш новый токен с вашим балансом. Если у Вас нет баланса,

проверьте, находитесь ли Вы на счету, с которого Вы создали токен. Если Вы не находитесь на счету, с которого Вы создали токен, у Вас будет нулевой баланс, потому что Вы еще не купили ни одного токена.

**Add Tokens**

Custom Token

Token Contract Address

MOGY

Decimals of Precision

Cancel

Next

sender / MOGY

1000000000 MOGY

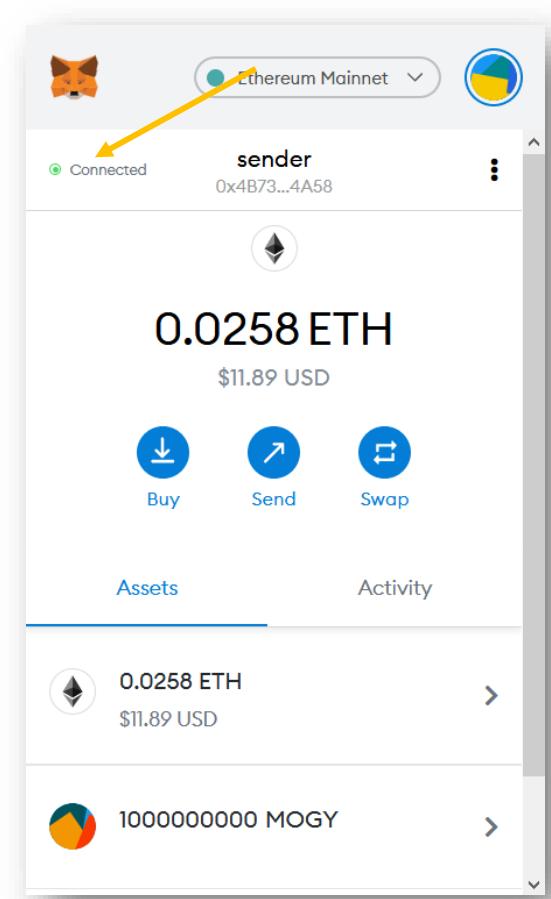
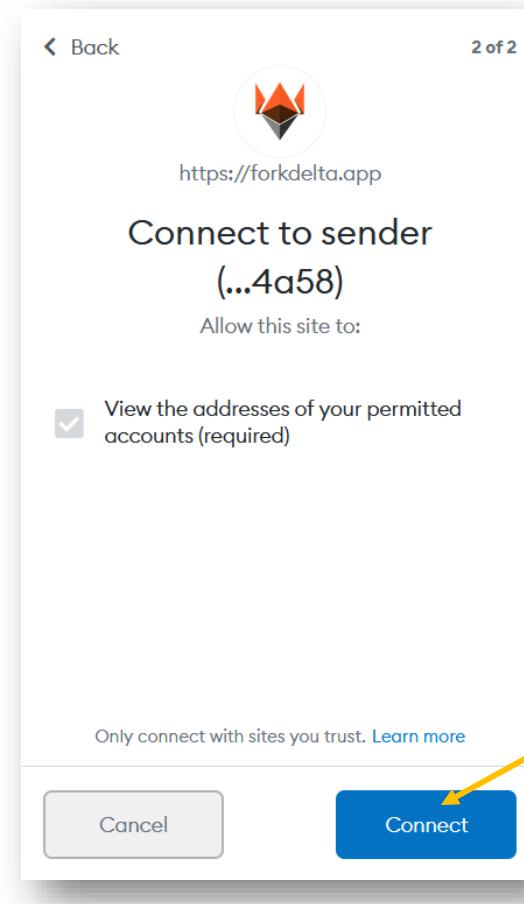
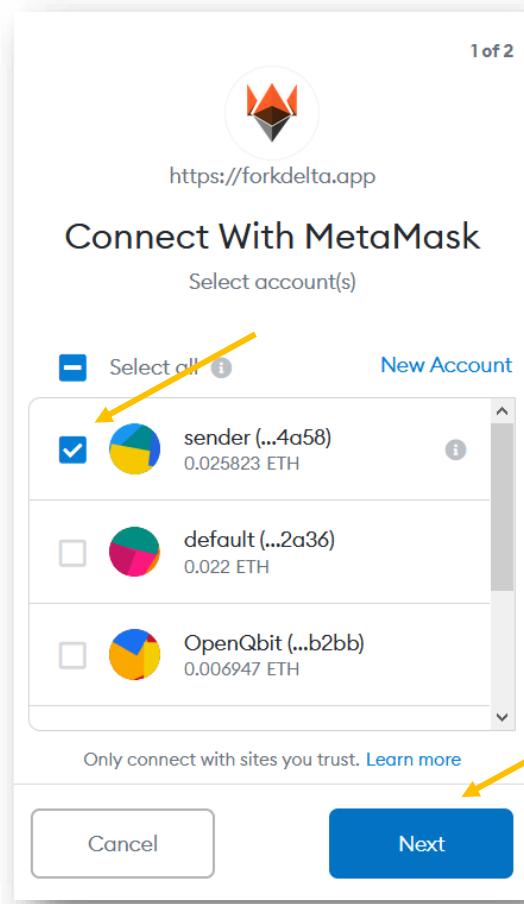
Send

Swap

You have no transactions

Как только мы загрузим наш новый токен для продажи на бирже [www.forkdelta.app](https://forkdelta.app).

Когда вы находитесь на сайте Exchange, вас попросят подключиться к сайту <https://forkdelta.app>. Нажмите кнопку "Далее" ниже, затем "Подключиться" и, наконец, вы сможете проверить, что вы подключены к сайту forkdelta.app.



Пришло время выпустить новый токен на Exchange <https://forkdelta.app>.

Нажмите на верхнее меню "DAI" и перейдите в конец прокрутки и выберите опцию "Другое".

The screenshot shows the ForkDelta exchange interface. At the top left, there is a dropdown menu with 'DAI' selected. A yellow arrow points from the text above to this dropdown. Below it, another yellow arrow points to the 'Other' option in a sub-menu. The main interface includes a balance section, a volume section, a search bar, and a deposit form. The central part features an 'Order Book' and a 'Price Chart'. To the right is a 'Trades' section showing recent trade history. At the bottom, there is a 'My Transactions' section with tabs for Important, Trades, Orders, and Funds, along with social media links and a 'Tweets' section.

Затем мы регистрируем новый токен с уже известными нам данными.

The screenshot shows the ForkDelta app interface. A modal window titled "Other token" is open, prompting for token registration details:

- Address:** 0x54093F2C720b18Fd795645b5101A37EB49d1A94a
- Name:** MOGY
- Decimals:** 18

Yellow arrows highlight the "Name" and "Decimals" fields. The background features a "Balance" section with DAI and ETH amounts, a "Volume" section with market data, and a "Trades" section showing recent trade history. A "New Order" form is also visible in the background.

В данный момент новый маркер появится в верхней левой части Exchange, мы только загрузили его на Exchange, нам нужно его опубликовать, чтобы все могли его купить и увидеть. Теперь нам нужно внести немного эфира (0.015) с нашего счета на Exchange.

The image consists of two side-by-side screenshots of the ForkDelta mobile application. Both screenshots show the 'Balance' section of the app.

**Left Screenshot:**

- The top navigation bar shows the ForkDelta logo and the token 'MOGY'.
- The 'Balance' section has three tabs: 'Deposit', 'Withdraw', and 'Transfer'. The 'Deposit' tab is selected.
- Under the 'Deposit' tab, there are three columns: 'Token' (labeled 'MOGY'), 'Wallet' (labeled '1000000000.000'), and 'ForkDelta' (labeled '0.000').
- Below these columns are two input fields: 'Amount' (with 'ETH' and '0.026') and another 'Amount' field (with '0.000').
- A blue 'Deposit' button is located at the bottom right of the deposit section.
- A note at the bottom says: 'Make sure MOGY is the token you actually want to trade.'
- The 'Volume' section below shows a search bar and a list of tokens with their current prices.
- At the bottom is a search bar with a magnifying glass icon and the placeholder 'Escribe aquí para buscar'.

**Right Screenshot:**

- The top navigation bar shows the ForkDelta logo and the token 'MOGY'.
- The 'Balance' section has three tabs: 'Deposit', 'Withdraw', and 'Transfer'. The 'Deposit' tab is selected.
- Under the 'Deposit' tab, there are three columns: 'Token' (labeled 'MOGY'), 'Wallet' (labeled '1000000000.000'), and 'ForkDelta' (labeled '0.000').
- Below these columns are two input fields: 'Amount' (with 'ETH' and '0.026') and another 'Amount' field (labeled '0.015').
- A blue 'Deposit' button is located at the bottom right of the deposit section.
- A note at the bottom says: 'Use this to deposit from your personal Ethereum wallet ("Wallet" column) to the current smart contract ("ForkDelta" column). Make sure MOGY is the token you actually want to trade.'
- The 'Volume' section below shows a search bar and a list of tokens with their current prices.
- At the bottom is a search bar with a magnifying glass icon and the placeholder 'Escribe aquí para buscar'.

Так как мы сделали депозит, мы можем внести столько же маркеров, сколько мы хотим на Exchange [www.forkdelta.app](https://www.forkdelta.app).

Чтобы внести определенное количество маркеров, нам нужно создать заказ на покупку, это делается внизу, где написано "Новый заказ", и мы нажимаем на опцию "Продать". Затем мы вводим количество жетонов, которое мы хотим сделать доступным для любого покупателя в мире, цену в Ether, которую мы хотим продать каждому (цена единицы), и параметр "Истечения" - это количество времени, которое мы хотим, чтобы эти жетоны были проданы:

Истечение Время = 14 секунд Введенное значение X.

Пример: 14 секунд X 10000 = 140 000 секунд = 1,62 дня.

The screenshot shows the ForkDelta app interface for the MOGY token. On the left, there's a 'Balance' section with tabs for Deposit, Withdraw, Transfer, Token, Wallet, and ForkDelta. It shows 1000000000.000 MOGY and 1.057 ETH. Below this is a 'Volume' section with a search bar and a list of tokens: ASTRO, REQ, SXDT, SNOV, POE. On the right, the 'Order Book' section for MOGY/ETH shows no orders. A 'New Order' form is open, with yellow arrows pointing to the following fields: 'Buy' (highlighted), 'Sell' (highlighted), 'MOGY' input field containing '10000' (highlighted), 'MOGY/ETH' input field containing '0.05' (highlighted), 'ETH' input field containing '500.000' (highlighted), and the 'Expires' input field containing '10000' (highlighted). At the bottom of the 'New Order' form is a large orange 'Sell' button.

Там, ваши новые жетоны в продаже, любой может зайти и купить их. Иногда он не распознает новый токен, поэтому их приходится продавать из приложения Metamask.

Пример консультации на сайте [www.etherscan.io](https://www.etherscan.io) созданного нового токена.

The screenshot shows the Etherscan interface for a transaction. The transaction hash is 0xd99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882. The status is Success. The transaction was included in block 11240201, which has 224 block confirmations. The timestamp is 47 mins ago (Nov-12-2020 02:49:56 AM +UTC) | Confirmed within 30 secs. The transaction originated from address 0x4b7355fd05be6dac458b004f54e12d6527a54a58 and went to a contract address [Contract 0x54093f2c720b18fd795645b5101a37eb49d1a94a Created]. The value sent 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%).

**Transaction Details**

Sponsored: Crypto.com - The Crypto Super App - Buy Crypto at True Cost with 0% fee on credit card purchase. [Get App Now.](#)

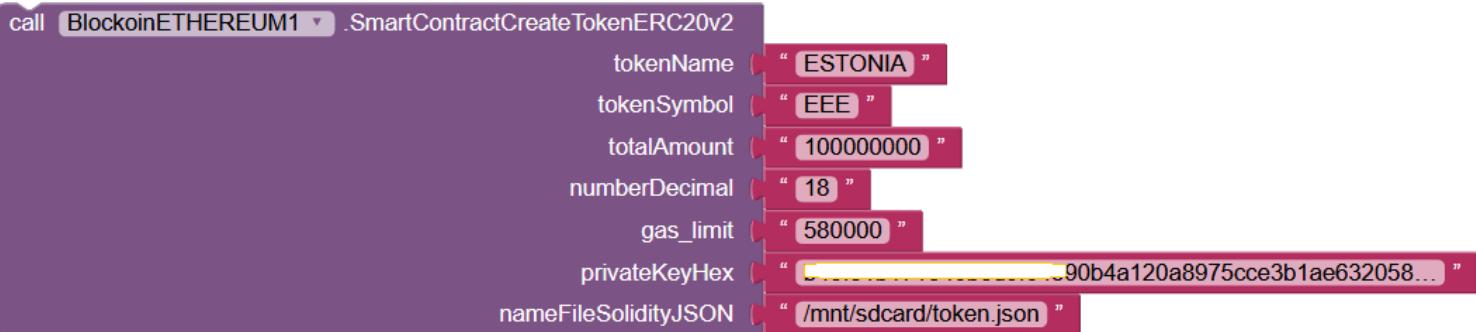
Overview	Status	Comments
② Transaction Hash:	0xd99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882	
② Status:	Success	
② Block:	11240201	224 Block Confirmations
② Timestamp:	47 mins ago (Nov-12-2020 02:49:56 AM +UTC)	Confirmed within 30 secs
② From:	0x4b7355fd05be6dac458b004f54e12d6527a54a58	
② To:	[Contract 0x54093f2c720b18fd795645b5101a37eb49d1a94a Created]	
② Value:	0 Ether (\$0.00)	
② Transaction Fee:	0.011014691 Ether (\$5.06)	
② Gas Price:	0.000000041 Ether (41 Gwei)	
② Gas Limit:	500,000	
② Gas Used by Transaction:	268,651 (53.73%)	

Транзакция (Tx\_hash) создания сети Ethereum.

Контрактный адрес вновь созданного токена.

Сеть Ethereum стоит всего лишь.  
Не включает операционные расходы +

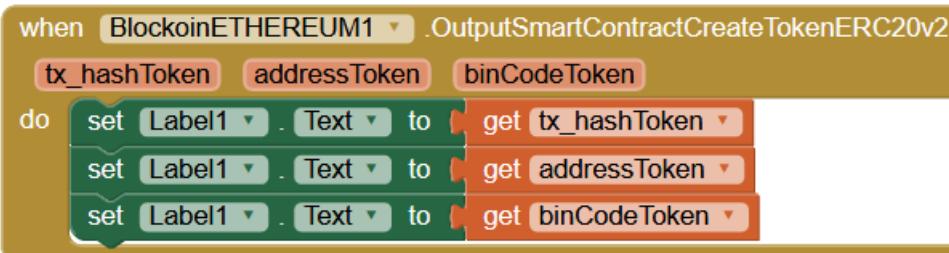
Блок для компиляции, создания и публикации ERC20 Token -  
**(SmartContractCreateTokenERC20v2)**



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

Выходные параметры: Событие (`OutputSmartContractTokenERC20v2`).

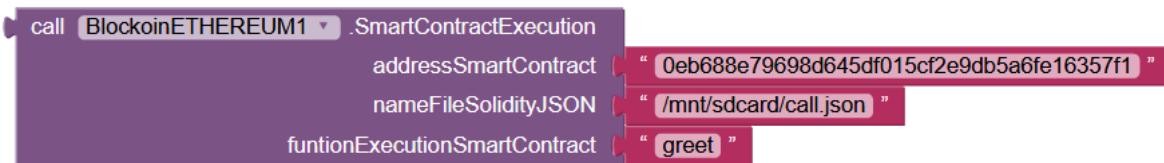
Выводы: `tx_hashToken<String>`, `addressToken<String>`, `binCodeToken<String>`.



Описание: Интеллектуальный контракт "Token ERC20" - активно публикуется в сети Ethereum. Версия 2 (v2) может быть оптимизирована, потому что в зависимости от "умного" контракта, как быстро вы хотите сделать публикацию в сети ethereum. Рекомендуется, чтобы минимальное значение для публикации без сбоев или задержек составляло 350 000 WEI.

Разница между функциями создания `TokenERC20v1` и создания `TokenERC20v2` заключается в том, что в случае версии 1 параметр `GasLimit` уже предварительно сконфигурирован и во 2-й версии может быть сконфигурирован под потребности пользователя или разработчика.

## Блокировка вызова или выполнения токена ERC20 - ([SmartContractExecution](#))



Входные параметры: addressSmartContract<String>, nameFileSolidityJSON<String>, funtionExecutionSmartContract<String>.

Выходные параметры: Выполнение функции, указанной в соответствующем "умном" договоре.

Выходные данные: **выполнение "умного" контракта.**

**ПРИМЕЧАНИЕ:** Для выполнения, файл должен быть создан в формате JSON, который содержит параметры первичного ключа адреса, по которому вы хотите выполнить "Смарт-контракт", вам необходимо ввести лимит газа (WEI).

Пример JSON-файла для выполнения функций скомпилированного Smart-контракта на примере упомянутой выше функции компилятора. Имя файла может быть произвольным.

Файл: call.json

```
{
  "Личный": "3са40...",
  "газ_лимит": 20000
}
```

Пример вывода на исполнение функции "greet" контракта Smart:

```
{
  "Gas_limit": 20,000,
  "address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",
  "результаты": [
    "Привет, блокшифровка".
  ]
}
```

ERC20 Token Property Display Block - (**SmartContractGetCreationTx**)

```
call BlockoinETHEREUM1 .SmartContractGetCreationTx
      txSmartContract " d99ff4d53f9b1f0687289674633d2acecf219011d163cdc0..." "
```

Входные параметры: **txSmartContract<String>**

Выходные параметры: Отображает свойства Smart контрактов, на которые ссылается (**Tx\_hash**).

Описание: Показывает основные функции и ABI код ссылочного контракта Smart.

Пример свойств токена ERC20, пример tokenName "MOGY", ранее созданного с помощью функции (**martContractCreateTokenERC20v1**)

```
{
  "block_hash": "cadb8914c43ed28fb370c9e1b6f5d8818ba31a1e944d1f7049441b982902dea8",
  "Блок_высота": 11240201,
  "Блок_индекс": 170,
  "hash": "d99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882",
  "адреса": [
    "4b7355fd05be6dac458b004f54e12d6527a54a58"
  ],
  "итого": 0,
  "гонорары": 110146910000000,
  "размер": 958,
  "Gas_limit": 500,000,
  "Gas_used": 268651,
  "цена газа": 41000000000,
  "контракт_создание": верно,
  "relayed_by": "200.77.24.87",
  "confirmed": "2020-11-12T02:49:56Z",
  "received": "2020-11-12T02:50:13.185Z",
  "видишь": 0,
}
```



Блок для отображения кода компиляции токена ERC20 - ([SmartContractGetcodeABI](#))

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

Входные параметры: addressSmartContract<String>.

Выходные параметры: Показывает ссылочный код Smart contract.

Описание: Показывает ABI код ссылочного контракта Smart.

Пример ABI-кода типового контракта Smart:

```

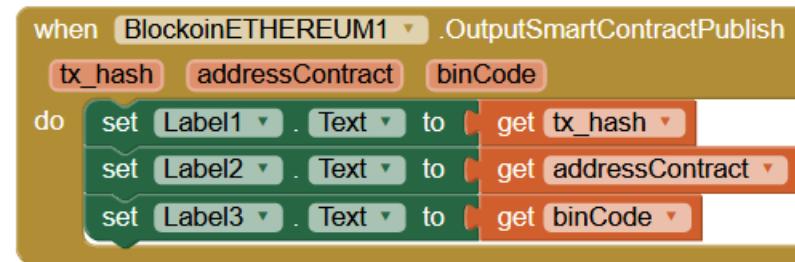
{
    "солидность": "контрактная смерть" {\n /* Определение владельца
переменной типа address*/\n    n address owner;\n    n /* эта функция
выполняется при инициализации и устанавливает владельца контракта */\n    n
функция mortal() { owner = msg.sender; }\n    n /* Функция возврата
средств по контракту */\n    n функция kill() { if (msg.sender == owner)
suicide(owner); }\n\ncontract greeter is mortal {\n    n /* define
variable greeting of the type string */\n    n string greeting;\n    n /*
она выполняется при выполнении договора */\n    n функция greeting(string
_greeting) public {\n        greeting = _greeting;\n    }\n    n /*
главная функция */\n    n функция greet() возвращает константу (строку)
{\n        return greeting;\n    }\n}\n    ",\n    "мусорное ведро":\n"606060405260405161023e38038061023e8339810160405280510160008054600160a060
020a031916331790558060016000509080519060200190828054600181600116156101000
203166002900490600052602060002090601f016020900481019282601f10609f57805160
ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565b50505
061016e806100d06000396000f35b828001600101855582156076579182015b8281111560
7657825182600050559160200191906001019060b0565b509056606060405260e060020a6
00035046341c0e1b58114610026578063cfaf321714610068575b005b6100246000543373
ffffffffffffffffff08116911614156101375760005473fff
ffffffffffff081526001805460a06020601f600260001961010086881615020190941693909304928301819004028101604
0526080828152929190828280156101645780601f10610139576101008083540402835291
60200191610164565b6040518080602001828103825283818151815260200191508051906
0200190808383829060006004602084601f0104600f02600301f150905090810190601f16
80156101295780820380516001836020036101000a031916815260200191505b509250505
06
    "Эби":\n" [{"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"}],\n    "creation_tx_hash":\n"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",\n    "created": "2016-07-20T01:54:50Z",\n    "address": "0eb688e79698d645df015cf2e9db5a6fe16357f1"}\n*Перед использованием следующего Блока (SmartContractPublish) необходимо использовать\nБлок (SmartContractCompilerSolidity) для проверки правильности написания Smart-контракта.
```

Блок для публикации токена ERC20 в сети Ethereum - (**SmartContractPublish**).



Входные параметры: **nameFileSolidityJSON<String>**

Выходные параметры: Отображает свойства ссылающегося "умного" контракта. В событии (**OutputSmartContractPublish**).



Описание: Публикация в сети ethereum договора Smart, ссылка на который содержится в JSON-файле.

Файл-пример: **PublishSmartContract.json**

```
{
    "солидность": "контрактная смерть" /* Определение владельца
переменной типа address*/\n    n address owner;\n    n /* эта функция
выполняется при инициализации и устанавливает владельца контракта */\n    n функция mortal() { owner = msg.sender; }\n    n /* Функция возврата
средств по контракту */\n    n функция kill() { if (msg.sender == owner)
suicide(owner); }\n\ncontract greeter is mortal {\n    n /* define
variable greeting of the type string */\n    n string greeting;\n    n /*
она выполняется при выполнении договора */\n    n функция greeting(string
_greeting) public {\n        greeting = _greeting;\n    }\n    n /*
главная функция */\n    n функция greet() возвращает константу (строку)
{\n        return greeting;\n    }\n}\n\n",
"Параметрики": ["Hello Test"],
"опубликовать": ["приветствие"],
"Личный": "3ca40...",
"газ_лимит": 500000
}
```

Как показано в коде выше, это тот же самый JSON код, который использовался в примере функции компиляции в том же самом коде, параметры которого были добавлены в конце JSON файла:

**Парамы:** Неявные параметры Smart contr.

**Публикация:** Название того, как будет опубликован "Умный контракт".

**Личный:** Первичный ключ счета, который будет исполнять "умный" контракт, должен иметь баланс.

**Gas\_limit:** это баланс в WEI, который вы хотите потратить на публикацию "Умного" контракта.

Пример вывода при выполнении (публикация Smart contract) Smart contract заключен в сети ethereum. В нем отображается осуществленная транзакция "**create\_tx\_hash**" и присвоенный адрес "адреса" созданного Smart контракта.

```
[  
 {  
     "имя": "greeter",  
     "согласие": "контрактная смерть" {\n        /* Определение владельца  
переменной типа address*/\n        n address owner;\n        n /* эта функция  
выполняется при инициализации и устанавливает владельца контракта */\n        n функция mortal() { owner = msg.sender; }\n        n /* Функция возврата  
средств по контракту */\n        n функция kill() { if (msg.sender == owner)  
suicide(owner); }\n    }\n    n\n    ncontract greeter is mortal {\n        n /* define  
variable greeting of the type string */\n        n string greeting;\n        n /*  
она выполняется при выполнении договора */\n        n функция greeting(string  
_greeting) public {\n            n\n            greeting = _greeting;\n        }\n        n /*  
главная функция */\n        n функция greet() возвращает константу (строку)  
{\n            n\n            return greeting;\n        }\n    }  
    ",  
    "мусорное ведро":  
"606060405260405161023e38038061023e8339810160405280510160008054600160a060  
020a031916331790558060016000509080519060200190828054600181600116156101000  
203166002900490600052602060002090601f016020900481019282601f10609f57805160  
ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565b50505  
061016e806100d06000396000f35b828001600101855582156076579182015b828111560  
7657825182600050559160200191906001019060b0565b509056606060405260e060020a6  
00035046341c0e1b58114610026578063cfae321714610068575b005b6100246000543373  
ffffffffffffffffff908116911614156101375760005473fff  
ffffffffffff908116911614156101375760005473fff  
a06020601f600260001961010086881615020190941693909304928301819004028101604  
0526080828152929190828280156101645780601f10610139576101008083540402835291  
60200191610164565b6040518080602001828103825283818151815260200191508051906  
0200190808383829060006004602084601f0104600f02600301f150905090810190601f16  
80156101295780820380516001836020036101000a031916815260200191505b509250505  
06  
    "abi": [  
        {  
            "постоянная": ложь,  
            "входы": [],  
            "имя": "убить",  
            "выходы": [],  
            "тип": "функция"  
        },  
        {  
            "постоянная": правда,  
            "входы": [],  
            "имя": "приветствие",  
            "выходы": [  
                {  
                    "имя": "",  
                    "тип": "строка"  
                }  
            ]  
        }  
    ]  
}
```

```

        }
    ],
    "тип": "функция"
},
{
    "входы": [
        {
            "имя": "_greeting",
            "тип": "строка"
        }
    ],
    "тип": "строитель"
}
],
"Gas_limit": 500,000,
"creation_tx_hash": "61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
"address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",
"Парамедики": [
    "Hello Test"
]
},
{
    "имя": "смертный",
    "сольдность": "контрактная смерть" /* Определение владельца переменной типа address*/\n    n address owner;\n    n /* эта функция выполняется при инициализации и устанавливает владельца контракта */\n    n функция mortal() { owner = msg.sender; }\n    n /* Функция возврата средств по контракту */\n    n функция kill() { if (msg.sender == owner) suicide(owner); }\n\ncontract greeter is mortal {\n    n /* define variable greeting of the type string */\n    n string greeting;\n    n /* она выполняется при выполнении договора */\n    n функция greeting(string _greeting) public {\n        n         greeting = _greeting;\n    }\n    n /* главная функция */\n    n функция greet() возвращает константу (строку)\n    {\n        n         return greeting;\n    }\n}\n\n    "мусорное ведро":\n"606060405260008054600160a060020a03191633179055605c8060226000396000f36060\n60405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffffffffffff\nffffffffffffffffffff90811691161415605a5760005473fffffffffffff\nfffffffffffff16ff5b56",
    "abi": [
        {
            "постоянная": ложь,
            "входы": [],
            "имя": "убить",
            "выходы": [],
            "тип": "функция"
        },
        {
            "входы": [],
            "тип": "строитель"
        }
],
"Gas_limit": 500,000,
"Парамедики": ["Hello Test"]

```

```
}
```

```
]
```

Тестовый блок для **создания** файла - (**createTestingFile**)

```
call BlockoinETHEREUM1 .createTestingFile  
pathTestFile " /mnt/sdcard/token.json "
```

Входные параметры: **pathTestFile<String>**

Выходные параметры: Создается тестовый файл по указанному пути.

Описание: Это служит для проверки правильности пути создания временного файла при использовании Блока (**SmartContractCreateTokenERC20v1**) или Блока (**SmartContractCreateTokenERC20v2**).

Блок для получения тарифов на газ - (**eth\_RatesGasStationInfo**).

```
call BlockoinETHEREUM1 .eth_RatesGasStationInfo
```

Входные параметры: **nameFileSolidityJSON<String>**

Выходные параметры: Отображает свойства ссылающегося "умного" контракта. В случае (**OutputEth\_GasStationInfo**) значения, переданные в GWEI.

Цена на газ - это стоимость, которая будет выплачена системам, осуществляющим транзакции в сети Ethereum. Эти системы широко известны как "шахтеры", а значения Цены на газ зависят от того, насколько быстро (по времени и приоритету) выполняется транзакция в сети Ethereum.

Поставляемые значения являются функцией следующего времени выполнения. Это время приблизительное и может варьироваться в зависимости от того, как вы выполняете требования (транзакции) в сети Ethereum.

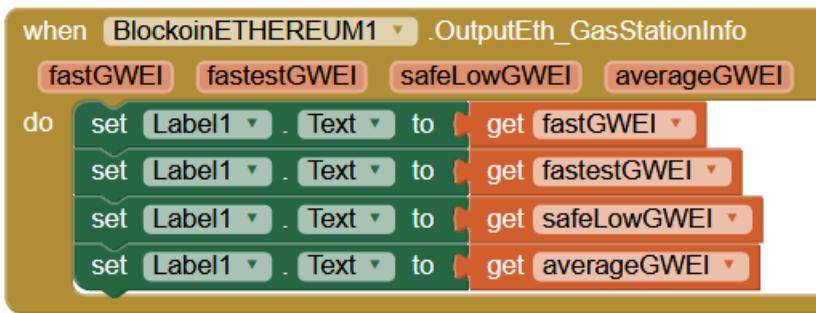
**Быстро** < 2 минуты.

**Самый быстрый** < 30 секунд.

**SafeLow** < 30 минут.

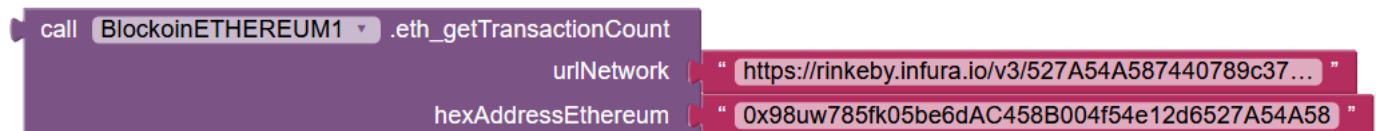
**В среднем** < 15 минут.

Операции, совершаемые с расширением биржевого Ethereum Extension (EEE), всегда используют цену газа = среднюю.



Описание: Получает обновленную цену газа в момент запроса для создания новой транзакции.

Блок для получения числа "nonce" - (`eth_getTransactionCount`).



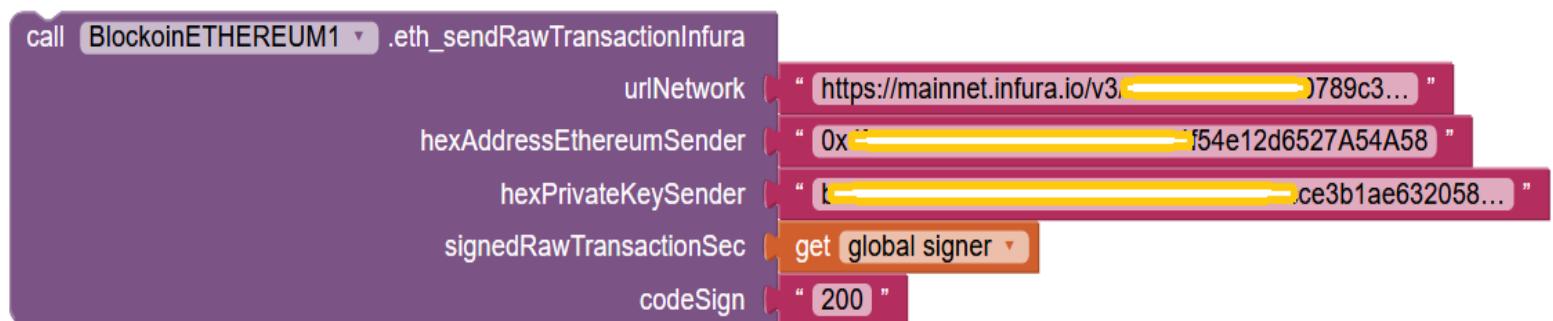
Входные параметры: `urlNetwork<String>`, `hexAddressEthereum<String>`.

Выходные параметры: отображает в шестнадцатеричном формате последовательный номер "nonce" адреса, на который делается ссылка.

Незначительное" число - это инкрементальное число, которое отслеживает количество транзакций, осуществленных с определенного адреса.

Описание: Получает "nonce" номер адреса.

Блок отправки подписанной транзакции - (`eth_SendRawTransactionInfura`).

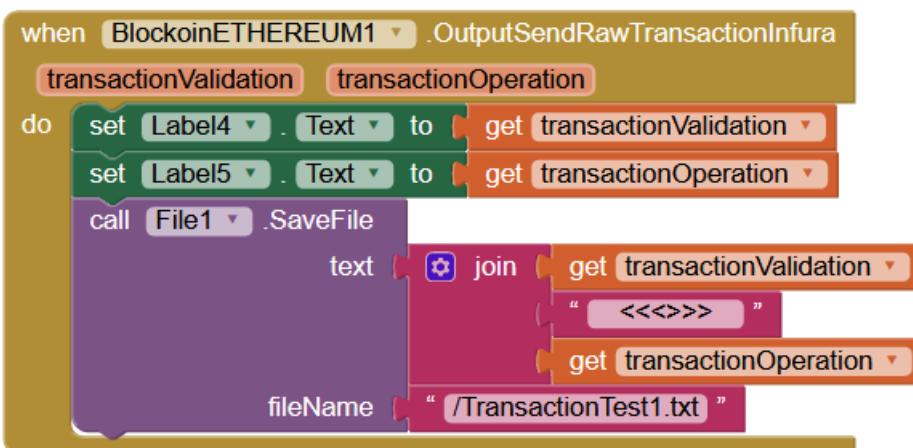


Необходимые зависимости: Блок (`SignerGenericPushRawTransactionOffline`). Блок (`SignerGenericPushRawTransactionOffline`).

Входные параметры: `urlNetwork <String>`, `hexAddressEthereumSender <String>`, `hexPrivateKeySender <String>`, `SignedRawTrasactionSec <String>`, `codeSign <String>`.

Выходные параметры: Событие (`OutputSendRawTransactionInfura`)

Выводы: `transactionValidation<String>`, `transactionOperation<String>`.



Описание: Он выдает два шестнадцатеричных значения в результате транзакций. Стоимость проверки транзакции (`transactionValidation value`) - это транзакция, осуществленная в сети Ethereum, которая содержит невынужденную стоимость Ethereum. Стоимость сделкиОперация - это сделка, совершаемая в сети Coinsolidation, стоимость которой составляет 0,5 цента США за каждую сделку, исходя из стоимости эфира на момент совершения сделки. Эта стоимость предназначена для оплаты услуг сети Coinsolidation.org и инвестируется в обслуживание, поддержку и создание расширений для криптографического сектора и сектора активов по всему миру.

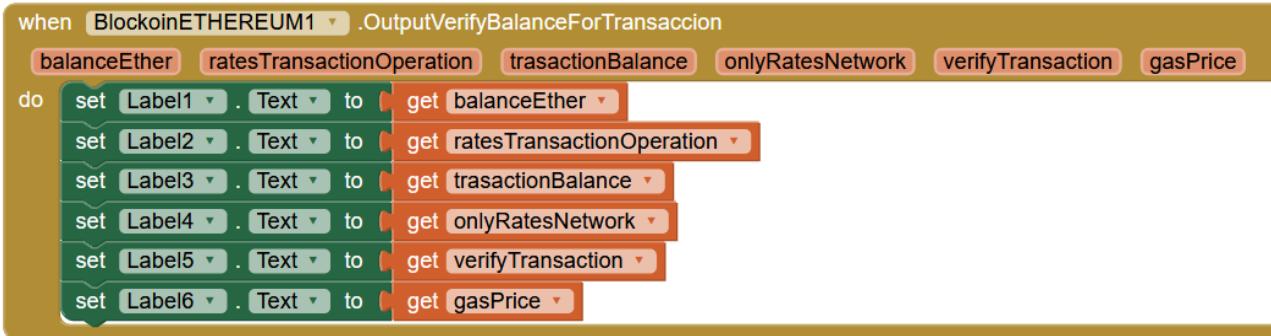
Блок для расчета стоимости стандартной сделки - (`eth_VerifiBalanceForTransaction`)



Входные параметры: `addressEthereumSender<String>`, `valueEthertoSend<String>`.

Выходные параметры: Событие (OutputVerifyBalanceForTransaction).

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



Описание: Предоставляет подробную информацию о том, какова будет стоимость стандартной сделки со ссылкой на адрес ввода. Выходной параметр "verifyTransaction" сообщает нам, можно ли сделать транзакцию "True" или если адрес, на который ссылаются, не имеет достаточного баланса, выдаст нам "False".

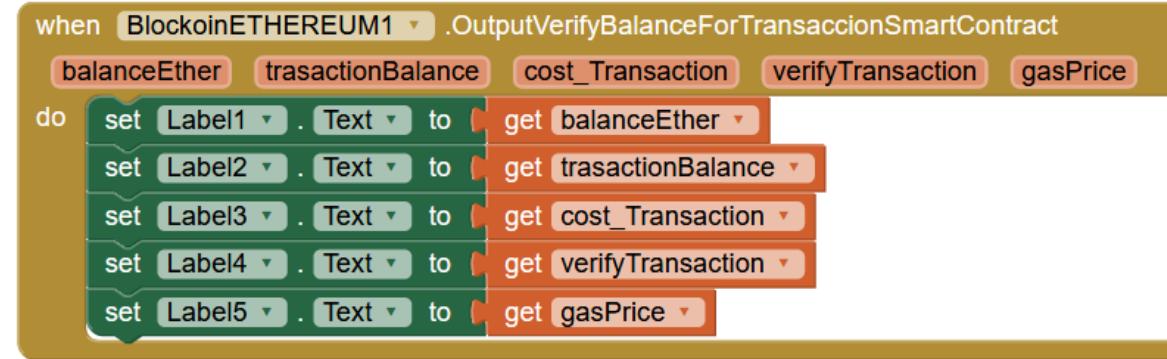
Блок для расчета стоимости стандартной сделки -  
(**eth\_VerifiBalanceForTransaccionSmartContract**)



Входные параметры: **addressEthereum<String>**, **gasLimit<String>**.

Выходные параметры: Событие (OutputVerifyBalanceForTransaccionSmartContract)

Выводы: **balanceEther<String>**, **trasactionBalance<String>**, **cost\_Transaction<String>**,  
**verifyTransaction<String>**, **gasPrice<String>**.



**Описание:** Предоставляет подробную информацию о приблизительной стоимости стандартной операции по размещению "умного" контракта со ссылкой на адрес, по которому был заключен контракт. Выходной параметр "verifyTransaction" сообщает нам, можно ли сделать транзакцию "True" или если адрес, на который ссылаются, не имеет достаточного баланса, выдаст нам "False".

**BalanceEther:** Баланс адреса, на который делается ссылка, доставляется в эфирах.

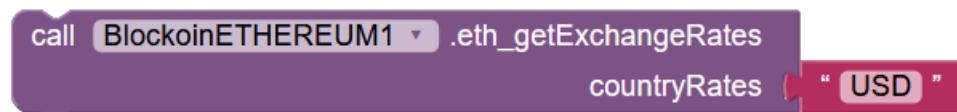
**trasactionBalance:** Баланс после совершения сделки.

**cost\_Transaction:** Стоимость сделки по публикации интеллектуального контракта.

**verifyTransaction:** (balanceEther минус cost\_Transaction).

**GasPrice:** Текущее значение GasPrice, используемое "Шахтерами", может меняться каждую минуту.

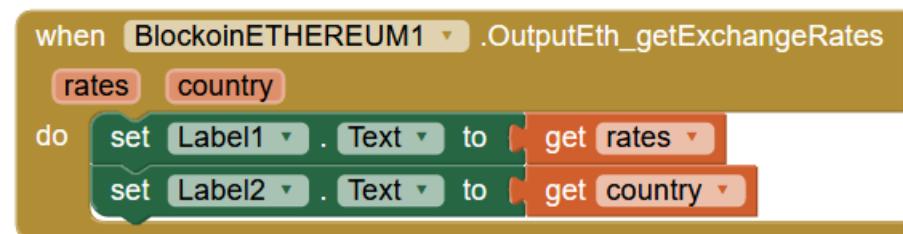
Блок для получения цены Ether в валюте указанной страны - (**eth\_getExchangeRates**).



Входные параметры: **countryRates<String>**. Проверьте выходной параметр "страна", где он содержит все типы валют страны, чтобы выбрать нужный.

Выходные параметры: Событие (**OutputEth\_getExchangeRates**).

Вывод: **показатели<String>, страны<String>** вывод в формате JSON все показатели стран мира.



Описание: Он поставляет текущую цену эфира по обменному курсу валюты соответствующей страны.

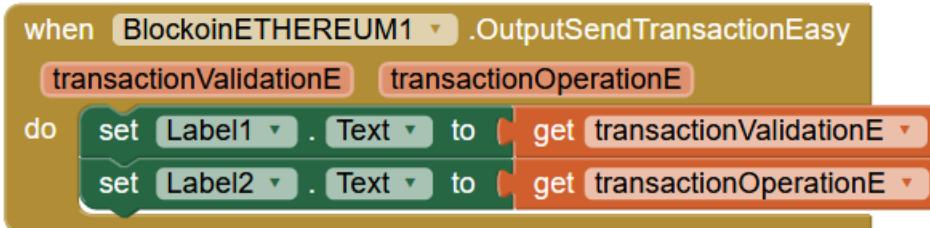
Блок для выполнения стандартной операции с минимальными входными параметрами - (`eth_sendTransactionEasy`).



Входные параметры: `hexPrivateKeySender<String>`, `toAddress<String>`, `valueEther<String>`.

Выходные параметры: Событие (`OutputSendTransactionEasy`)

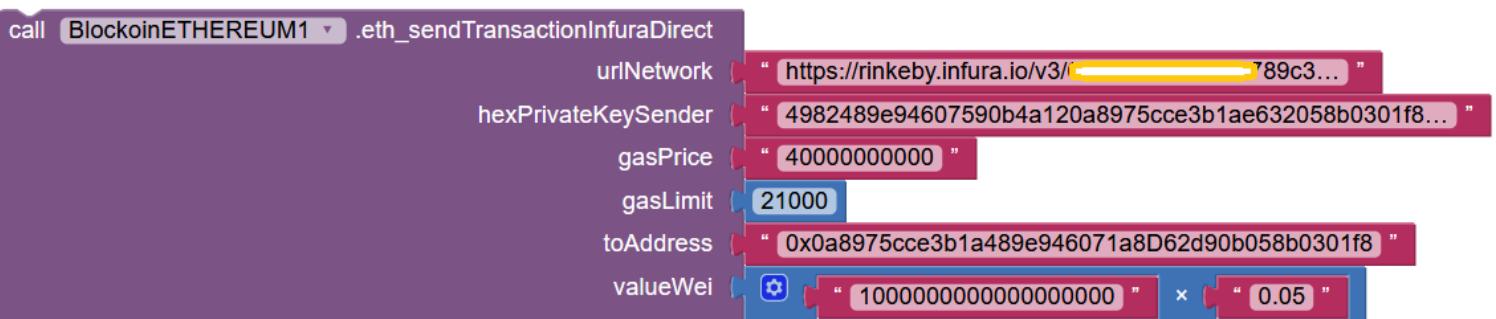
Выводы: `transactionValidation<String>`, `transactionOperation<String>`.



Описание: Функция для выполнения стандартной транзакции в сети Ethereum, эта функция используется сразу же, вам не нужно иметь учетную запись в INFURA и вам нужно только 3 входных параметра, вам нужно только иметь достаточно баланса, чтобы сделать желаемую транзакцию.

Транзакции размещаются в сети Ethereum непосредственно через официальную библиотеку Ethereum Web3j и нашу сеть Coinsolidation.org.

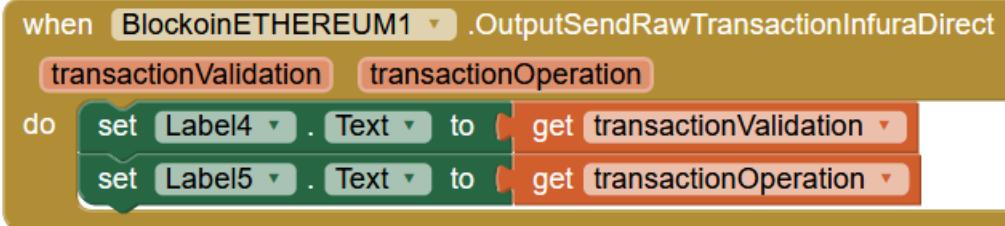
Блок для выполнения стандартной транзакции с минимальными входными параметрами - (`eth_sendTransactionInfuraDirect`).



Входные параметры: `urlNetwork<String>`, `hexPrivateKeySender<String>`, `gasPrice<String>`, `gasLimit<String>`, `toAddress<String>`, `valueWEI<String>`.

Выходные параметры: событие (`OutputSendTransactionInfuraDirect`)

Выводы: `transactionValidation<String>`, `transactionOperation<String>`.



Описание: Функция посылать стандартный трансакт, который уже содержит неявную цифровую подпись, это полезно для людей, которые уже имеют предыдущие знания о компонентах трансакции и хотят оптимизировать эти параметры в соответствии с их потребностями.

## 11. Расчет стандартной стоимости операции и "умная" контрактная операция

Для расчета стандартной транзакции в сети Ethereum необходимы 3 параметра.

- 1.- Цена на газ.
- 2.- Газовый лимит.
- 3.- Текущая стоимость эфира (цифровая валюта Ethereum).

**Цена газа:** Обычно она дается в единицах GWEI (GigaWEI), что соответствует, если 1 эфир имеет 1,000,000,000,000 wei, то 1 GWEI равен 1,000,000,000. Это устройство служит для оплаты систем, которые выполняют все транзакции сети Ethereum и называются "шахтеры", оно распространяется по всему миру. GasPrice не является фиксированным значением и является переменным и может изменяться от минуты к минуте или секунде. Те, кто определяет значение GasPrice, являются "шахтерами", и это зависит от того, насколько насыщена сеть Ethereum.

**GasLimit:** это значение обычно дается в единицах WEI, а в стандартной сделке среднее значение по умолчанию равно 21 000 WEI, хотя оно может быть больше или меньше в зависимости от того, какой тип сделки вы хотите сделать, в стандартной сделке мы используем значение 40 000 WEI, чтобы убедиться, что у нас нет отказа в том, что мы не подпадаем под Gas Limit.

**Значение Ether:** Это значение также является переменным и из-за различных параметров глобального финансового рынка, это значение может быть получено от организаций, которые всегда обновляли значение Ether во всем мире, называемых централизованными и децентрализованными биржами.

**ВАЖНОЕ ПРИМЕЧАНИЕ:** В Exchange Ethereum Extension (EEE) мы используем более высокий лимит газа (40,000), это связано с тем, что в случае не достижения минимальной квоты, установленной "горняками", ТРАНСАКТИВАНИЕ НЕ ЭФФЕКТИВНО, однако сеть Ethereum, если она будет взимать расходы, которые она сделала при расчете транзакции, не делая этого, По этой причине некоторые пользователи не объясняют, почему их транзакция не выполняется, однако с них будет снята сумма или весь GasLimit, который был предложен при запросе на транзакцию, по этой причине мы всегда должны четко понимать, каковы будут значения GasLimit и Цены на газ.

С функцией **eth\_GetRatesGasStation** можно ознакомиться в GasPrice, а GasLimit для стандартных транзакций должен быть выше 21 000 WEI, а транзакции для публикации и/или выполнения "умного" контракта должны быть не менее 500 000 WEI.

Стандартная стоимость операции.

Она определяется по следующей формуле:

Стоимость = (GasLimit x (GasPrice / (1,000,000,000,000))) x Эфирное значение.

Пример:

Оцените следующие значения:

**GasLimit** = 40,000, **GasPrice** = 45 GWEI, **Ether Value** = \$406.

Стоимость = (40,000 x (45,000,000 / (1,000,000,000))) x 406

**Стандартная стоимость операции** = 0,0018 эфира x 406 USD = 0,73 USD

В случае "умного" контракта необходимо знать, какой тип "умного" контракта будет опубликован, так как стоимость прямо пропорциональна той рабочей нагрузке, которую "горняки" должны будут выполнить для обработки "умного" контракта, другими словами, объем обработки в компьютерных системах, с которыми "горняки" справляются проще.

В случае "умного" контракта значением по умолчанию будет запуск GasLimit со значением 500 000 WEI.

Важный момент, который необходимо принять во внимание, заключается в том, что когда предлагается GasLimit, "горняки" не обязательно будут учитывать всю предложенную сумму, т.е. когда человек отправляет сделку, "горняки" подсчитывают вычислительные усилия и вычитают то, что будет вычтено из GasLimit, что в некоторых случаях может быть меньше или равно стандартному GasLimit в размере 21 000 WEI, предлагаемому по умолчанию.

Со всеми сделками можно будет ознакомиться на сайте [www.etherscan.io](http://www.etherscan.io), где мы можем проконсультироваться с деталями каждой сделки.

Пример стандартной сделки, где сделка была отправлена с газовым лимитом в 40 000 WEI, однако "шахтеры" при расчете, сколько будет стоить обработка, приняли только 52,5%, т.е. значение по умолчанию составляет 21 000 WEI.

The screenshot shows the Etherscan Transaction Details page for a specific Ethereum transaction. The transaction hash is 0x7dae251f21c616fb04a94112633c97e04d11c91f4d06dd9b85ed11112f02703a. The transaction was successful and has 2 block confirmations. It was timestamped 52 seconds ago on November 11, 2020, at 06:17:24 AM UTC. The transaction originated from 0x4b7355fd05be6dac458b004f54e12d6527a54a58 and was sent to 0x5d2acdb34c279aa6d1e94a77f7b18ab938fb2bb. The value transferred was 0.001084598698481562 Ether (\$0.50), and 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 actual gas used was 21,000 (52.5% of the limit). A yellow arrow points from the text "Предлагаемый лимит газа: 40" to the gas limit value. Another yellow arrow points from the text "Фактический лимит газа, использованный в сделке: 21" to the gas used value.

Parameter	Value	Notes
Transaction Hash	0x7dae251f21c616fb04a94112633c97e04d11c91f4d06dd9b85ed11112f02703a	
Status	Success	
Block	11234630	2 Block Confirmations
Timestamp	52 secs ago (Nov-11-2020 06:17:24 AM +UTC)	Confirmed within 38 secs
From	0x4b7355fd05be6dac458b004f54e12d6527a54a58	
To	0x5d2acdb34c279aa6d1e94a77f7b18ab938fb2bb	
Value	0.001084598698481562 Ether (\$0.50)	
Transaction Fee	0.000672 Ether (\$0.31)	
Gas Price	0.000000032 Ether (32 Gwei)	
Gas Limit	40,000	
Gas Used by Transaction	21,000 (52.5%)	

Возвращаясь к "умной" контрактной сделке и взяв 500 000 WEI GasLimit, мы получаем следующую стоимость, если используем все это.

Умная стоимость операции по контракту.

Она определяется по следующей формуле:

Стоимость = (GasLimit x (GasPrice / (1,000,000,000,000)) x Эфирное значение.

Пример:

Оцените следующие значения:

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

Стоимость =  $(500,000 \times (45,000,000 / (1,000,000,000))) \times 406$

**Стоимость операции публикуется Умный контракт** = 0,0225 эфир x 406 USD = 9 135 USD

Со всеми сделками можно ознакомиться на сайте [www.etherscan.io](http://www.etherscan.io).

**ПРИМЕЧАНИЕ:** Чтобы получить общую стоимость операции, вы должны сложить их:

**Итого Стоимость операции** = Стоимость сети Ethereum + Комиссия за объединение в сеть Coinsolidation.org.

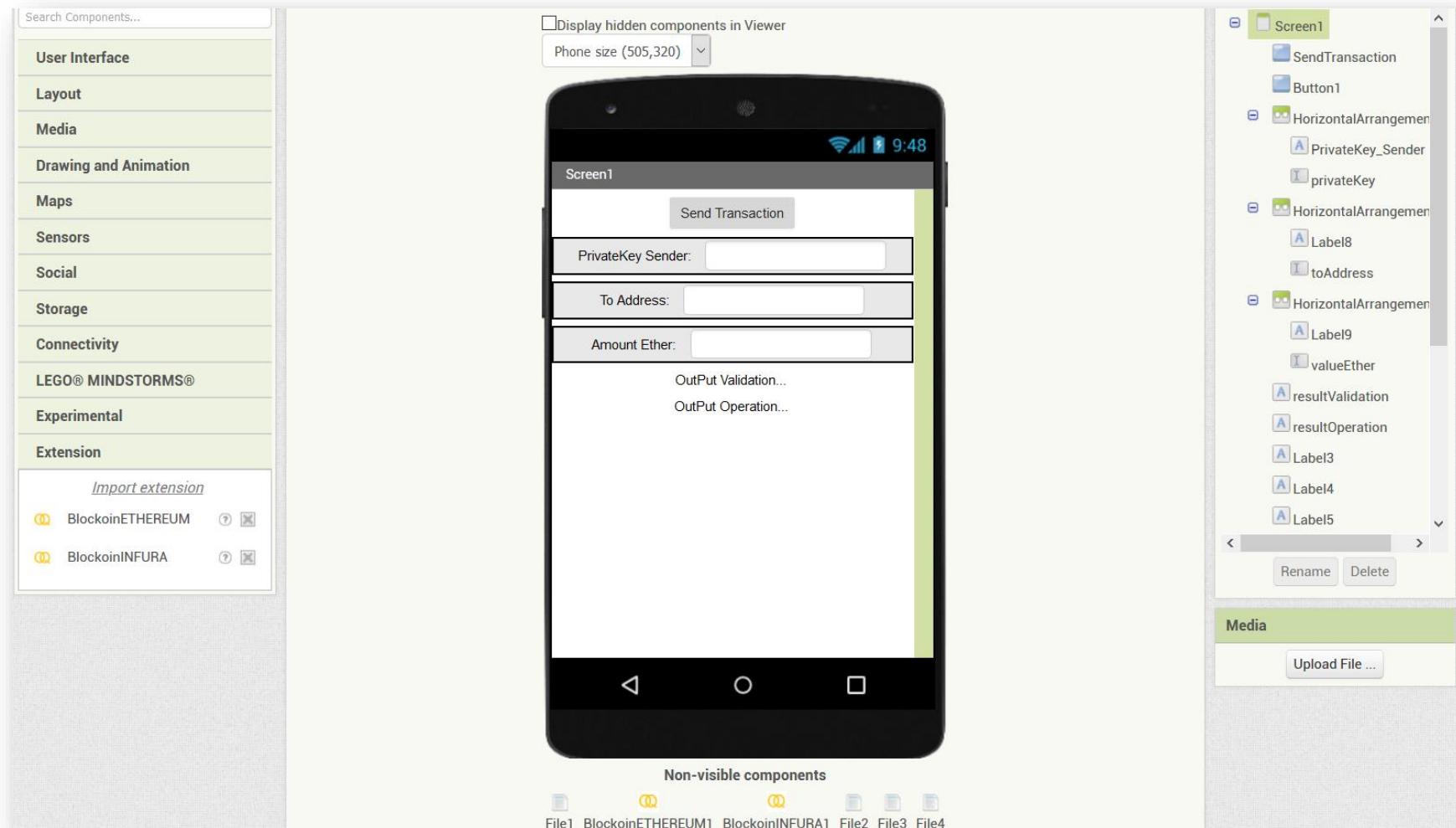
## 12. Плата за участие в конкурсе Coinsolidation.org

Стандартная сделка: 0,5 цента США + стоимость сети Ethereum.

Публикация и/или исполнение "умного" контракта: \$15 USD + стоимость сети Ethereum.

### 13. Создайте ваше приложение для Android (Exchange) за 15 минут.

Дизайн в App Inventor (Экран). - 5 минут.



Функциональные блоки (`eth_SendTransactionEasy`) и событие (`OutPutSendTransactionEasy`) - 5 минут

```

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

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

```

Вводите данные:

Первичный ключ к адресу отправителя.

**Адрес:** шестнадцатеричный адрес получателя.

**valueEther:** Дайте количество эфира, которое будет отправлено.

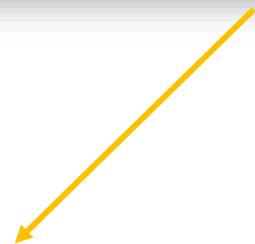
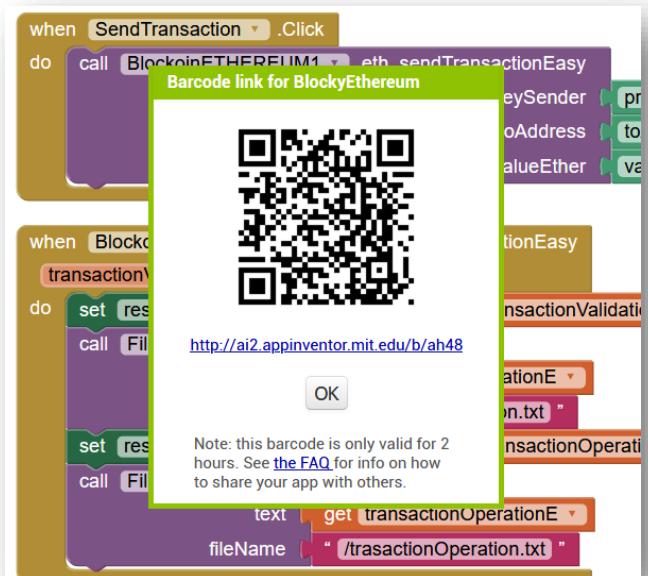
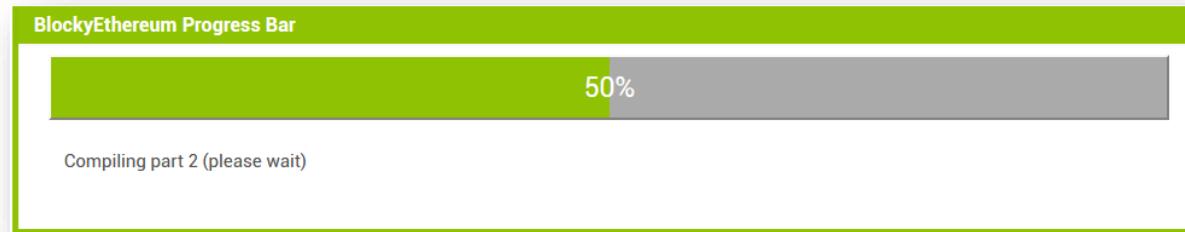
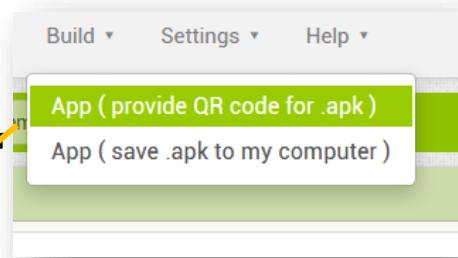
Сохраните результаты в текстовых файлах:

Функция File1: Файл **трассацииValidation.txt**

Сохраните результаты в текстовых файлах:

Функция File2: **Трассация файловОценка.txt**

Мы компилируем, генерируем APK файл, чтобы установить его на Android устройство. - 5 минут



**ПРИМЕЧАНИЕ:** Когда транзакция будет выполнена, потребуется примерно от 6 до 8 секунд, чтобы отпустить кнопку "Отправить транзакцию". Из-за времени подключения к сети Ethereum.

## 14. Уплотнение монет Токена.

Консолидация монет Токена - это проект с тремя основными направлениями.

Представьте, что у вас есть собственная криптовалюта Token для развития вашего бизнеса. С Coinsolidation это будет легко и просто.

Первый заключается в создании первой сети расширений, основанных на визуальной методологии программирования Blockly, в которой благодаря ее простому и интуитивному использованию может быть использован любым человеком, не обладающим предварительными знаниями в области программирования, расширения, с которыми можно ознакомиться в нашей Дорожной карте (Белая книга), направлены на два фундаментальных сектора мировой экономики, сектор крипто-валют и/или жетонов и сектор валют (fiat) или общего использования во всем мире, таких как доллар США, евро ЕС, фунт стерлингов или любой другой валюты, используемой в настоящее время.

Вторым руководством в проекте CoinSolidation является создание нового алгоритма для консолидации адресов текущих и будущих криптомуонтажей. Мы разрабатываем новый алгоритм моделирования для создания адреса, объединяющего различные адреса из разных блоков в универсальный адрес, см. нашу (Белую книгу) по адресу [www.Coinsolidation.org](http://www.Coinsolidation.org) или <https://github.com/coinsolidation/whitepaper>.

Третьим руководством является применение технологии Quantum Computing в безопасности среди CoinSolidation, которая будет применяться с уже разработанными расширениями алгоритмов безопасности QRNG (Quantum Random Number Generator) и PQC (Post-Quantum Computing). Их можно найти в официальном репозитории расширений на сайте Github, <https://github.com/coinsolidation>.

## Общие характеристики CryptoToken COINsolidation

Название: COINsolidation

Символ: CUAG

Страна запуска: Эстония

Официальный сайт: [www.Coinsolidation.org](http://www.Coinsolidation.org)

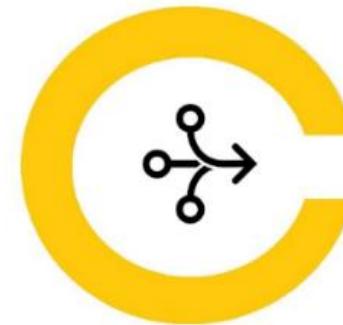
Компания: Международная организация по консолидации монет.

Дата запуска: 04/30/2021 г.

Создал: Guillermo Vidal.

Алгоритм консенсуса: Доказательство Кванта.

Алгоритм адресации: Консолидированный универсальный адрес.



## 15. Лицензирование и использование программного обеспечения.

Лицензирование, условия использования см. [www.coinsolidation.org](http://www.coinsolidation.org) или пишите по адресу [info@coinsolidation.org](mailto:info@coinsolidation.org).