



EX^{change}_{tensions}

設定と管理。

Ethereum Exchange Extension (EEE)。

ユーザーズガイド

バージョン 1.0.0.0 ベータ

2020年11月に

Blockoin.orgは、Bankoin Inc.の登録商標であり、自由使用および商用ライセンスに基づいています。利用条件：www.CoinSolidation.org

内容

1.	序章です。	2
2.	Blocklyプログラミングとは？	5
3.	延長とは何ですか？	5
4.	BlockoinEthereumとBlockoinINFURAとは？	6
5.	Ethereumプラットフォームに適用される基本的な概念。	6
6.	拡張機能とEthereumテスト環境のインストールと設定	14
7.	ブロックの定義と使い方（汎用機能）	23
8.	Exchange Ethereum Extension (EEE) の機能とイベント。	24
9.	CryptoTokenまたはCryptomoney Tokenを作成する手順です。	36
10.	新しい資産や自分のクリプトトークンを売りに出す方法(トークンERC20)	38
11.	標準取引コストの計算とスマートコントラクト取引	63
12.	Coinsolidation.org 料金	67
13.	15分でAndroidアプリ（Exchange）を作成します。	68
14.	トークンコインソリダクション。	71
15.	ソフトウェアのライセンスと使用。	72

1. 序章です。

Ethereumはオープンソースのプラットフォームであり、他のブロックチェーンとは異なり分散化されているため、Ethereumはより多くのことを行うことができます。プログラム可能なので、開発者はこれを使って新しいタイプのアプリケーションを作成することができます。そのデジタル通貨は「イーサ」と呼ばれています。

これらの分散型アプリケーション（または「ダップス」）は、暗号モンタージュやブロックチェーン技術の恩恵を受けることができます。彼らは信頼性が高く、予測可能であり、一度Ethereumに「ロード」されると、常にスケジュール通りに実行されることを意味します。彼らはデジタル資産をコントロールして、新しいタイプの金融アプリケーションを作成することができます。彼らは分散化することができます、つまり、誰か一人の実体や人が彼らを支配することはありません。

現在、世界中の何千人もの開発者がEthereum上でアプリケーションを作成し、新しいタイプのアプリケーションを発明しています。

- ETHまたは他の資産で安価で即時の支払いを可能にする暗号通貨のポートフォリオ
- デジタル資産を借りたり、貸したり、投資したりできる金融アプリケーション
- 分散型の市場で、デジタル資産を交換したり、現実世界の出来事についての「予測」を交換したりすることができます。
- ゲーム内に資産を持っていて、リアルマネーを獲得することもできるゲーム。

エーテルの仕組みは？

イーサは、他の暗号通貨と同様に、すべての取引が記録される共有デジタルブックを使用しています。公開されていて、完全に透明で、後から修正するのは非常に困難です。

これはブロックチェーンと呼ばれるもので、マイニングによって構築されます。

鉱夫は、エーテル取引のグループを検証して「ブロック」を形成し、複雑なアルゴリズムを解いてコーディングする役割を担っています。これらのアルゴリズムは、ブロック処理時間にある程度の整合性を保つための方法として、多かれ少なかれ困難である可能性があります（約14秒に1つ）。

その後、新しいブロックは前のブロックチェーンにリンクされ、問題の鉱夫は報酬、すなわち一定数のエーテルトークンを受け取ります。通常は5エーテル単位ですが、クリプト通貨が上昇を続けるとこの数字は変動すると見られます。

Ethereumの仕組みは？

Ethereumブロックチェーンはビットコインと非常に似ていますが、そのプログラミング言語により、開発者は取引を管理し、特定の結果を自動化するためのソフトウェアを作成することができます。このソフトはスマートコントラクトと呼ばれています。

従来の契約が関係の条件を記述している場合、スマートな契約は、コードで記述することで、それらの条件が満たされていることを確認します。これらは、予め定められた条件が満たされると自動的に契約を実行するプログラムであり、手動で契約を実行する際に発生する遅延やコストを排除しています。

簡単な例を挙げると、Ethereumユーザーがスマートコントラクトを作成して、一定の日に設定された量のエーテルを友人に送ることができます。彼らはこのコードをブロック文字列に書き込んで、契約が完了したとき（つまり合意した日付に達したとき）にエーテルが自動的に送信されるようにします。

この基本的な考え方は、より複雑な構成にも適用でき、その可能性はおそらく無限大であり、保険、不動産、金融サービス、法律サービス、マイクロファイナンスなどの分野すでに顕著な進展を遂げているプロジェクトがあります。

スマートコントラクトには、さらにいくつかのメリットがあります。

1. 彼らは仲介者の図を排除し、ユーザーにトータルコントロールを提供し、余分なコストを最小限に抑えます。
2. それらはパブリックブロックチェーンに登録され、暗号化され、複製され、すべてのユーザーが市場活動を見ることができます。
3. 手動プロセスに必要な時間と手間を省く

もちろん、スマートコントラクトはまだ新しいシステムで、細かい部分を磨く必要があります。コードは文字通りに翻訳されているので、契約書作成中にエラーが発生すると、変更できない望まない結果になる可能性があります。

DApps vs. スマートコントラクト

インテリジェントコントラクトは、DApps（分散型アプリケーション）との類似点を共有していますが、いくつかの重要な違いとは切り離されています。

スマートコントラクトと同様に、DAppは分散型ピアネットワークを介してプロバイダからのサービスにユーザーを接続するインターフェースです。しかし、スマートコントラクトは一定の参加者数を作成する必要がありますが、DAppsはユーザー数の制限がありません。さらに、スマートコントラクトのような金融アプリケーションに限定されるものではありません。

2. Blocklyプログラミングとは？

BlocklyはJavaScriptのプログラミング言語をベースにしたビジュアルなプログラミング手法で、パズルのピースのように組み合わせができるシンプルなコマンド群で構成されています。直感的でシンプルな方法でプログラミングを学びたい人や、すでにプログラミングを知っていて、この手のプログラミングの可能性を見てみたい人にはとても便利なツールです。

Blocklyは、コンピュータ言語のいずれかの種類のバックグラウンドを必要としないプログラミングの形態です。

誰もが理解するのが難しいこれらのプログラミング言語をいじらずに携帯電話（スマートフォン）用のプログラムを作成することができます、ちょうど作成するためのシンプルで簡単かつ迅速な方法でグラフィカルな方法でブロックをまとめてください。

3. 延長とは何ですか？

拡張子とは、拡張子.AIXのファイルで与えられるJavaプログラミング言語で作られたモジュールのことです。

Blockoin.orgプロジェクトでは、プログラミングの膨大な知識を持たずに数分でモバイルアプリケーションを作るためには得ることができる金融分野のためのユーザー・フレンドリーな拡張機能を作成することに基づいています。

4. BlockoinEthereumとBlockoinINFURAとは？

BlockoinEthereumとBlockoinINFURAは、Ethereumのネットワークで使用されている作成することができるようになるには、次の技術的なソリューション（アルゴリズム）が含まれています無料で使用するソフトウェア（拡張機能）です。

- エセラム・プラットフォーム上の新しいアカウントの作成。
- バランスシート・コンサルテーション、資産移転（現金・預金）について
- 新しいERC20トークンとクリプトモネダトークンの誕生です。
- スマートコントラクトの集計、作成、コンサルティング、公表
- オフラインおよびオンライン取引の作成。
- プライマリキーとパブリックキーの読み込み。
- 為替レートとガソリンスタンドのコンサルティング。
- 開発、テスト、および地球環境 中核ネットワーク環境 (ネットワーク)
- 39のINFURAの機能が含まれています。

5. Ethereumプラットフォームに適用される基本的な概念。

ブロックチェーンとは？

ブロックチェーンは一般的にビットコインや他の暗号通貨と関連付けられていますが、デジタルマネーだけでなく、ユーザー・企業にとって価値のあるあらゆる情報に利用できるため、これらは氷山の一角に過ぎません。 1991年にスチュアート・ハーバーとW・スコット・ストルネットが暗号的に保護されたブロックの連鎖の最初の作品を記述したことによる起源を持つこの技術は、ビットコインの登場とともに普及する2008年まで注目されていませんでした。しかし、現在は他の商用アプリケーションでの利用が求められており、中期的には金融機関やIoT（モノのインターネット）など、いくつかの市場で成長すると予測されています。

ブロックチェーンとは、ブロックチェーンという言葉で知られるように、ネットワーク上の複数のノード（PC、スマートフォン、タブレットなどの電子機器）に分散された、合意された単一の記録のことです。暗号通貨の場合は、それぞれの取引が記録されている会計帳簿を考えることができます。

その動作は、その実装の内部的な詳細にまで踏み込むと複雑なものになりますが、基本的な考え方方はシンプルです。

各ブロックに格納されています。

1. 有効な記録または取引の数。

2. - そのブロックに関する情報。

3. 前のブロックと次のブロックとのリンクは、各ブロックのハッシュを介して、ブロックの指紋のようになる一意のコード。

したがって、各ブロックは前のブロックのハッシュからの情報を含んでいるため、各ブロックはチェーン内で特定の移動不可能な場所を持っています。チェーン全体がブロックチェーンを構成する各ネットワークノードに保存されるため、チェーンの正確なコピーがすべてのネットワーク参加者に保存されます。

ブロックチェーン Ethereum プラットフォーム内のアドレスやアカウントとは？

これは、Ethereum プラットフォームにおける 42 文字の文字列で、Ethereum で定義された資産が預け入れられたり、送られたりする 16 進数ベースの数字を表しています。他のブロックチェーン プラットフォームでは、例えば、アカウントやアドレスの文字数が異なる場合があります。

0x5d2Acdb34c279Aa6d1e94a77F7b18aB938BFb2bB

クリptoモニーとは？

それは、交換の媒体として機能するように設計されたデジタルまたは仮想通貨です。暗号（デジタルセキュリティ）を利用して、取引の安全性を確保したり、検証したり、特定の暗号通貨の新しい単位の作成を制御したりします。

エーテルとは？

イーサは、2013年に開発された分散型プラットフォーム「Ethereum」のブロックチェーンプラットフォームのネイティブデジタル通貨です。エーテルとは、WEIと呼ばれる小さな単位に分割できる単位で、次のような等価性を持っています。

1エーテル = 1,000,000,000,000,000,000,000魏。(数学的表現では 10^{18})。

エーテルを使用する場合、どのような単位で扱われますか？

1	エーテル
10^3	フィニー
10^6	沙保
10^9	シャノンかGWEIこれはGasPriceに使われています。
10^{12}	ババゲー
10^{15}	愛の花
10^{18}	ウェイ

トークンとは？

トークンは、与えられたプロジェクトのエコシステム内で使用できるデジタル資産です。

トークンと暗号通貨の主な違いは、前者は動作するために（独自のものではなく）別のブロックチェーン・プラットフォームを必要とすることです。Ethereumは主にスマートコントラクト機能があるため、トークンを作成する際に最も一般的なプラットフォームです。Ethereumブロックチェーン上で作成されたトークンは一般的にERC-20トークンとして知られていますが、他にも主に収集可能な資産（カード、ビデオゲームでの使用、アートワークなど）に使用されるERC-721トークンなど、より専門的なタイプのトークンがあります。

ガスとは？

ガスとは、Ethereumネットワーク上で操作や一連の操作を行うためのコストのことです。これらの操作は、トランザクションの作成からスマートコントラクトの実行、または分散型アプリケーションの作成まで、いくつかの種類があります。

つまり、もっと簡単に言うと、Ethereumで行われている作業を計測するための単位がGasです。

物理的な世界と同様に、Ethereumでは、他のものよりもコストがかかるジョブもあります：私たちが実行したい操作は、プラットフォームを形成するノードがリソースをより多く使用する必要がある場合は、これはガスも同様に増加し、その逆もあります。

ガスは、主にEthereumプラットフォーム上で3つの機能を実行するために機能します。

1. - タスクの実行にコストを割り当てる; Ethereumでは、タスクの実行にもコストがかかります。そのタスクの難易度や処理してほしい速度に応じて、その操作にかかる計算コストが高くなったり低くなったりして、それに比例してガスの数も増えたり減ったりする。
2. - システムを確保する; Ethereumのシステムは安全なシステムであり、これはGasのおかげで大きく可能です。

実行された各取引に対して手数料を支払うことを要求することで、プラットフォームは、使用不可能な取引がネットワーク上で処理されないことを保証します。これにより、ブロックチェーンに無駄なメガバイトの情報がたくさん追加されることがなくなり、ブロックチェーンを軽くすることができます。

さらに、Gasでは、システムは「スパム」やループの無限の使用からも保護されています：コードで反復的なタスクを実行するための命令。

例えば、ガスが存在しなければ、タスクが無限に繰り返され、システムが崩壊して使えなくなることを防ぐことはできません。

3. - 鉱夫への報酬；鉱夫とは、世界中に分散している独立した外部システムで、Ethereumプラットフォーム内の取引の実行を担当しています。取引をしたり、スマートコントラクトを締結したりすると、一定額のガスを「支払う」ことになります。

このガスは、彼らが使用した資源（ハードウェア、電気、時間）のために鉱山労働者に「支払う」のに役立ち、また、彼らの仕事に報酬を追加します。そのため、ガスは台のバランスを整える役割も果たしていると言えるでしょう。

私たちは、ガスを「支払う」という話をしています。言い換えれば、取引を処理するために Ethereumにかかる費用を「支払う」ということです。

ガス料金とは？

ガスユニットは、コンピュータのステップなどの命令の実行に対応しています。

では、鉱夫たちは報酬として得たガスをどうやって「現金化」しているのでしょうか？

ガス自体には価値がないので課金できません。この使用済みガスを「有料化」するためには、この消費された資源に価値を与えること、つまり採掘者の仕事に金銭的な価値を与えることが必要です。取引やスマートコントラクトで使用されるガスの量は、イーサに相当する価格を持っています。この価格は「ガス価格」と呼ばれ、通常は採掘者によって割り当てられ、Ethereumブロックチェーンがどれだけ混雑しているかによって変動します。通常はGWEI単位での取り扱いとなります。

ガスリミットとは？

このデータは、トランザクションが有效地に消費できるガスの最大値を示しています。

通常、Ethereumネットワーク上で取引を行う際に使用するソフトウェアは、取引を行うために必要なGasの見積もりを自動で計算して、すぐに見せてくれますが、このソフトウェアを使用することで、取引を行うために必要なGasの見積もりを自動で計算してくれます。

ガソリンスタンドとは？

それは、Ethereumブロックチェーンの中で様々な種類の取引を行うために必要なGasPriceをコンセンサスで決めるために、採掘者が扱う値を参照する場所です。

どの程度のGasPriceが選択されているかに応じて、トランザクションの実行に与えられた優先時間が重み付けされ、通常は3種類のGasPriceが扱われます：TRADER：ASAP、FAST < 2分、

STANDARD<5分。これらは平均値であり、Ethereumのメインネットワークのワークロード（トラクション）によって時間が変動する可能性があります。

<https://ethgasstation.info/>

取引所とは何ですか？

クリプトマニアック交換所とは、不換紙幣や他のクリプトマニアックと交換することで、クリプトマニアックの交換が行われる集会所です。これらのオンライン取引所の家では、市場価格は、需要と供給に基づいて暗号の値をマークする生成されます。

為替レートとは？

例えば、本マニュアル作成日現在のイーサの米ドルでの価値は430.94ドルとなります。

スマートコントラクトとは？

Ethereumではスマート契約は、事前に確立されたルールに基づいて自動的に実行される命令を持っているブロックチェーンEthereum内にあるSolidityと呼ばれるプログラミング言語のプログラムで、このプロパティは、異なる民間および公共部門に適応した多くのスマート契約を作成することができますように、すべての既存のブロックチェーンでの進化を行います。

nonce」とは何か？

これは、Ethereumで作成されたアカウントや各方向の取引を記録しておくためのインクリメンタルな「16進数」カウンターです。

取引とは何ですか？

それは、Ethereumシステム内で事前に確立された価値を与えられ、後に企業や個人のために有形の価値に変更することができる、ある種の非有形資産の実行または譲渡です。

txHashとは？

それは、各トランザクションの詳細な結果を追跡するのに役立ちます16進数です。

どのような取引があるのでしょうか？

あなたは2つのタイプがありますが、1つは、トランザクション"オフライン"これは、Ethereumのネットワークに接続することを選択するまで、Ethereumのメインネットワークへの接続を持っている必要がなく、保存することができます作成され、トランザクションを解放し、全体のトランザクションは、ネットワーク接続である可能性があります任意の異常を防ぐオフラインで処理されるため、セキュリティの利点を持っています。もう一つの取引は、接続がもたらすセキュリティ上のメリットとデメリットで、常にインターネットに接続する必要がある「オンライン」のものです。

INFURA.ioとは？

Infura.ioは、EthereumとIPFSへのシンプルで信頼性の高いアクセスで、開発者がアプリケーションのブロックチェーンをテストからスケーリングまで簡単に行えるようにするツールとインフラのセットを提供するプラットフォームです。

Ethereumネットワーク上のアドレスとは？

アドレスやアカウントは、3つの部分、アドレス、公開鍵と秘密鍵で構成され、これらの2つのキーは、送受信（アクティブ）またはエーテル（デジタル通貨）に使用される16進数形式の数字と文字の文字列です。

主な鍵は、口座に保持されている残高の解放（取引のサイン）を承認するものであるため、誰とも共有されるべきではありません。

公開鍵は、価値の面でも、誰に送るかの面でも、取引が正しいことを確認するための参照であるため、公開されている全公開人に知られており、誰とでも共有されています。

例。

{

"private": "429a043ea6393b358d3542ff2aab9338b9c0ed928e35ec0aed630b93adb14a1c",

```
"public":  
"049b4b7e72701a09d3ee09165bba460f2549494a9d9fd7a95aac57c2827eac162fd9e105b  
2461cd6594ca8ca6a8daf10fe982f918be1b0060c87db9cfbcd289a8",  
"address": "88ab6dcecc3603c7042f4334fc06db8e8d7062d5"  
}  
}
```

Coinsolidation.orgとは？

それは、暗号通貨の統合のプラットフォームであり、我々は、機能拡張に基づいてプログラミングの高度な知識を必要とせずに視覚的なプログラミングの形であるBlockly技術を使用して、中央集権と分散型金融の世界で最初のハイブリッドアドレスを作成しました。ホワイトペーパーは www.coinsolidation.org でご覧ください。

ブロックチェーンEthereumのテスト用ネットワークやメインネットワークにはどのような種類があるのでしょうか？

<https://mainnet.infura.io>

メインネットワーク、すべての取引がリアルタイムで行われる生産ネットワーク。

<https://ropsten.infura.io>

Ropstenのテストネットワークは、ブロックチェーン開発者がライブ環境で自分の仕事をテストすることを可能にしますが、実際のETHトークンを必要とせずに、（Proof-of-Work）と呼ばれるアルゴリズムを使用しています。

<https://kovan.infura.io>

その前身のropstenとは異なり、権威の証明と呼ばれるアルゴリズムを使用しているKovanのテストネットワーク、。

<https://rinkeby.infura.io>

テストネットワークは、権威の証明と呼ばれるアルゴリズムを使用しています。テストによく使われるものの一つ。

<https://goerli.infura.io>

Goerliは、POA（Proof of Authority）コンセンサスエンジンが様々なクライアントでサポートしているEthereumの公開テストネットワークです。スパム対策。

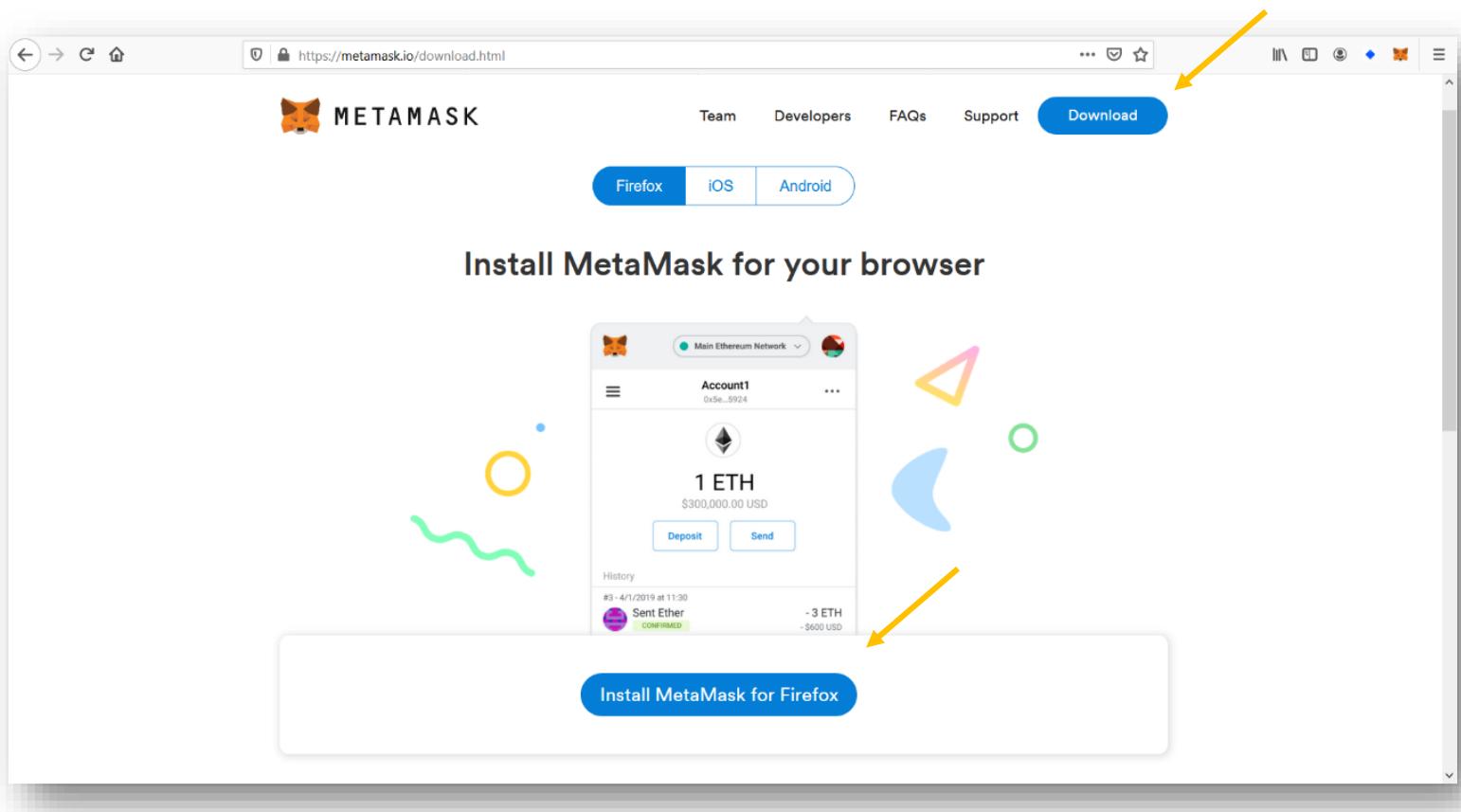
合計では、ブロックチェーンのEthereumに基づいて5つのネットワークがありますが、1つの生産またはメインとテストのための4つのために、我々は私たちのテスト環境をインストールするRinkebyのネットワークを使用します。

6. 拡張機能とEthereumテスト環境のインストールと設定

まずはテスト環境が必要です。Ethereumプラットフォーム上で行うすべての取引の作成、送信、公開、レビュー、データ取得に使用する2つの拡張機能の異なる機能の使用方法を学ぶことができます。

まずはテスト環境を整えることから始めます。今回はMETAMASKというアプリをインストールすることになりますが、これはEthereumアカウントの作成やインポート、取引管理をブラウザから行うためのウォレットで、使用するインターネットブラウザはMozillaで、Chromeでも使用できます。

公式サイト（www.metamask.io）に移動し、「Download」ボタンをクリックしてください。

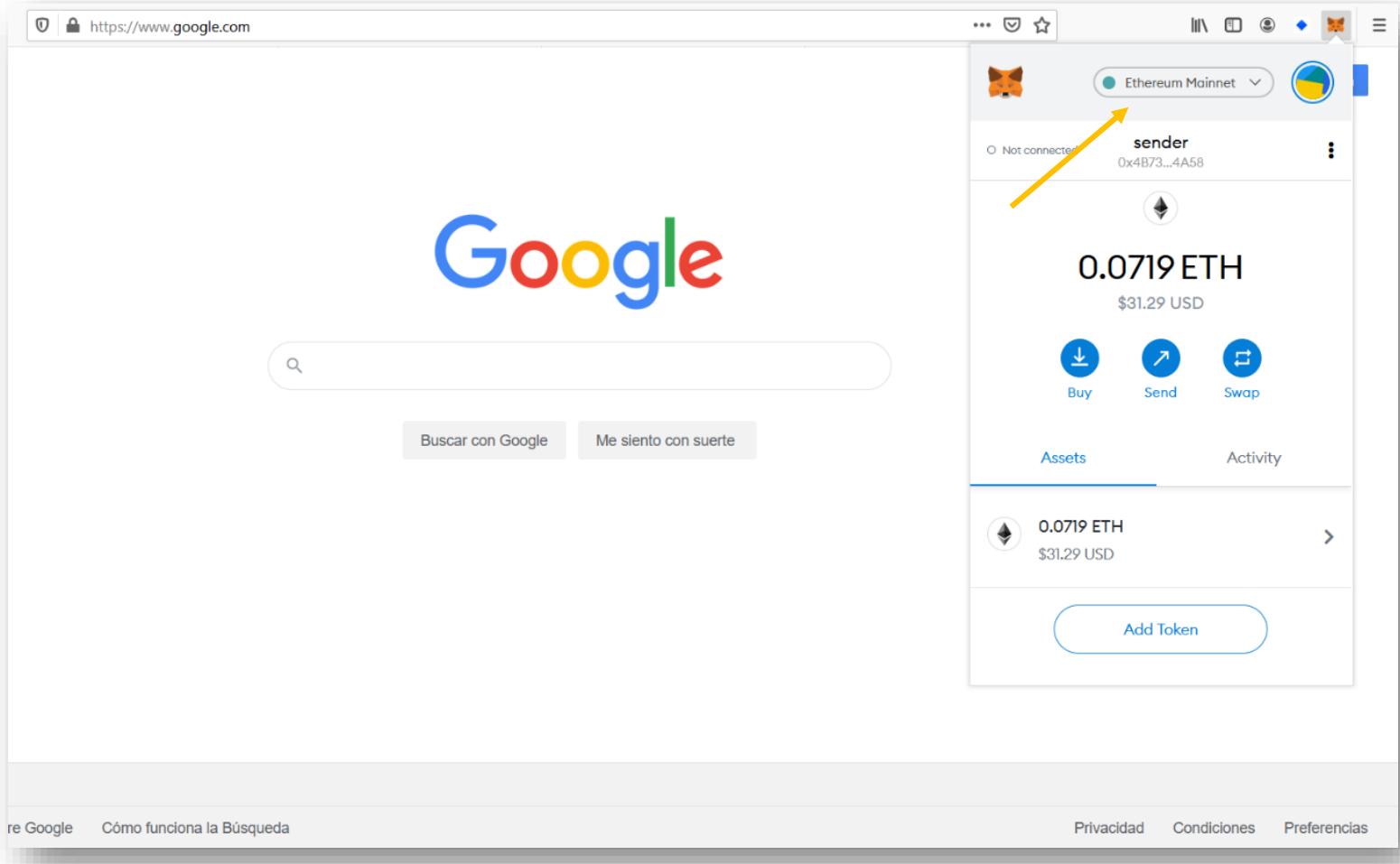


そして「Install MetaMask for Firefox」ボタンをクリックし、デフォルトのオプションを受け入れます。インストール後、右上のアイコンが表示されますが、そこにはすでにメタマスクソフトウェアがインストールされていることがわかります。

メタマスクのアイコンをクリックすると、既存のアカウントをインポートするか、新しいアカウントを作成するかのオプションが表示されます。このケースでは、"restore through seed"という方法を使用

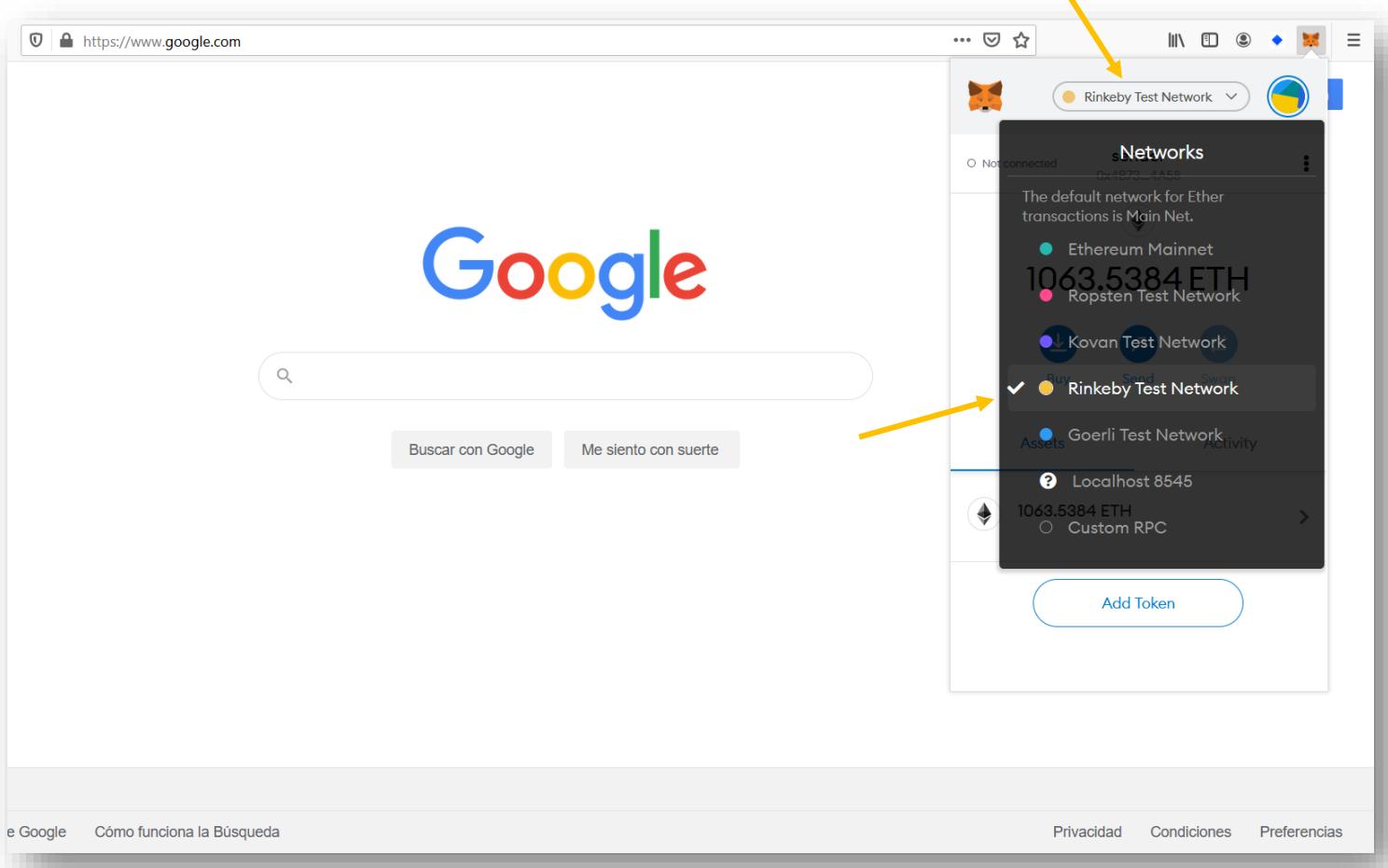
して、すでに実行しているアカウントをインポートします。お持ちでない場合は、新しいアカウントを作成してください。この場合、どちらの場合もパスワードの割り当てを求められます。

後で我々は "パスワード"で入力し、我々は持っているどのようなバランスをチェックすることができ、論理的には、それが新しい場合は、バランスはnullになります。



我々は今、我々は私たちのRinkebyテストアカウントにいくつかのエーテル（デジタルコイン）を入れようとしている方法を検討するために進みます。

上部には、我々が使用するネットワークの種類を選択するオプションがあり、デフォルトでは、あなたがそれを入力すると、"Ethereum Mainnet"であなたを配置しますが、あなたがクリックすると、我々は我々が選択することができますすべてのオプションのネットワークを確認することができますが、私たちのケースでは、我々はRinkebyネットワークを選択します。



次のステップは、私たちのリンクリーネットワークを参照したテストアカウントにエーテルを入金することです。

重要な注意: Rinkeyのテストネットワークに参照されているアカウントに入金されたエーテル（デジタル通貨）は、暗号市場では価値がなく、ソフトウェアのテストにのみ使用されます。取引でエラーが出ないように、どのネットワークで作業しているかを常に確認しておきましょう。

検査用のエーテルを得るために、以下の手順を実行する必要があります。

あなたが持っている任意のtwitterアカウントで我々はtwitterを入力し、唯一のアカウントと/またはアドレスを含むコメントを作成する必要があります我々は、我々はこれを持っているので、我々は我々のアカウントをアンカーに次のリンクに移動しますコメントのリンクをコピーする必要がありま
すメタマスクで持っている。

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

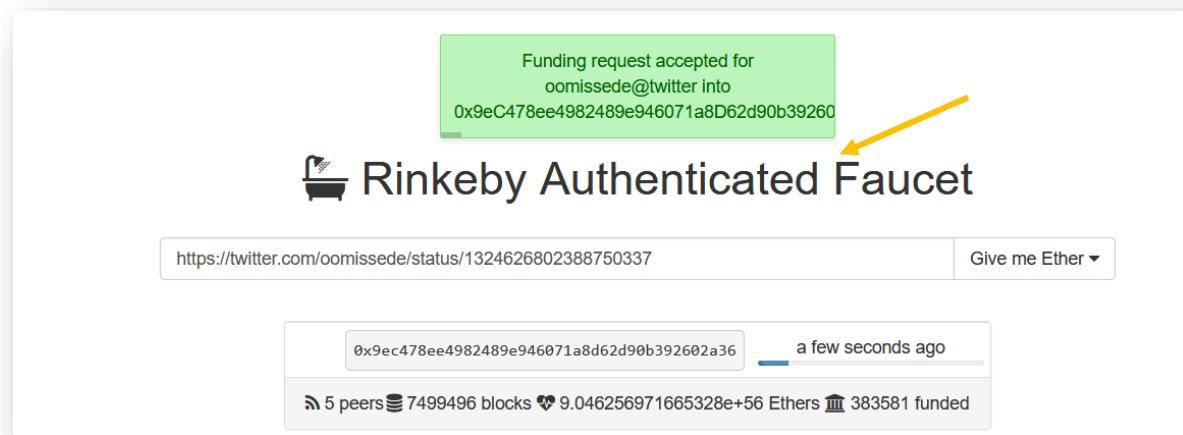
twitterのコメント欄を作って、コメント欄のリンクをkopipasteしてみました。

A screenshot of a Twitter post from user AGAVE (@oomissede). The post contains a copied link: 0x88ac6dcecc3603c7042f4334fc06db8e8d7062d5. A yellow arrow points to the URL in the browser's address bar: https://twitter.com/oomissede/status/1324628185468854272.

サイト (<https://faucet.rinkeby.io/>) では、twitterのリンクをコピーして、ボタン上の"Give me Ether"で希望の金額を選択します。

A screenshot of the Rinkeby Authenticated Faucet website. A yellow arrow points to the input field where the Twitter link was pasted. Another yellow arrow points to the dropdown menu for selecting the amount of Ether to receive: "3 Ethers / 8 hours", "7.5 Ethers / 1 day", and "18.75 Ethers / 3 days".

最後に、すべてが正しい場合は、我々は入金が受理されたことを発表し、Rinkebyネットワークシステムのワークフローに応じて、入金は数分以内になるか、またはより長くかかる可能性があります
◦



これでイーサーズのアカウントを持っているので、Ethereumプラットフォームでのテストを始めることができます。

Ethereumブロックチェーンと対話するための拡張機能が2つあります。

共闘の延長線上にあるBlockoinETHEREUM.aixには、以下の機能を実行するための機能が含まれています。

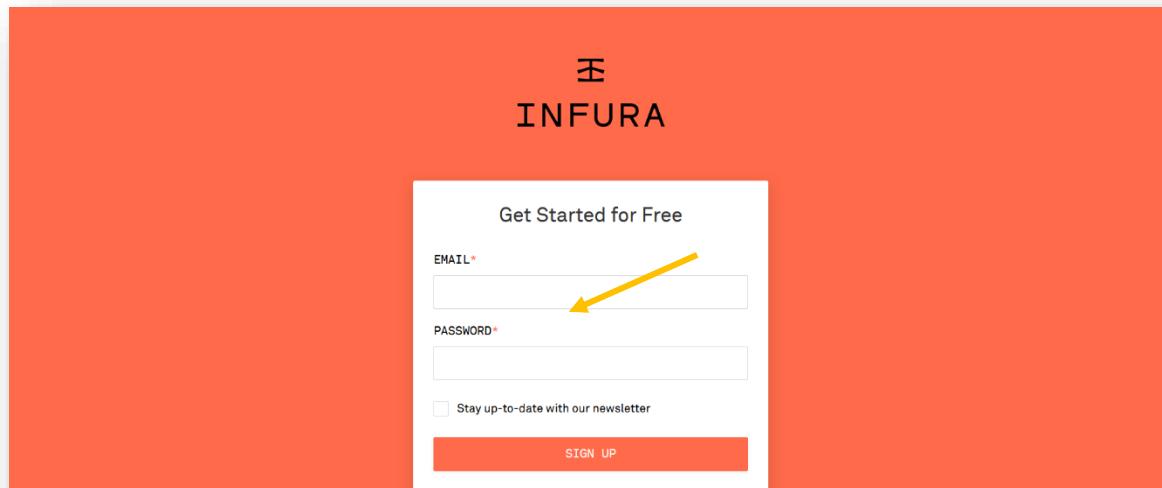
- ブロックチェーンにおける新規アカウント（アドレス）の作成 Ethereum (privateKey、publicKey、アドレス)
- アカウントデータ（アドレス）をバイナリファイルで保存すること。
- バイナリファイルのアカウント（アドレス）のインポート
- privateKeyでアカウント（アドレス）を取得します。

- 口座（アドレス）間の取引を「オンライン」で作成・送信
- オフラインPushRawトランザクションの作成、署名、送信
- 取引内容の相談 Tx.
- 住所やスマート契約のバランスチェック
- スマートコントラクトのためのコンパイラ。
- スマートコントラクトの作成・発行・実行
- トークンERC20（クリプトモニートークン）の作成・公開・実行
- スマートコントラクトからABIコードを取得
- ネットワーク接続の検証。
- 世界のどの国のクリプトマーケットでもエーテルの価値のクエリ（その国の自国通貨）- 為替レート。
- ガスプライスの相談。
- 特定口座の「nonce」の相談。

コインソシデーションの延長線上にあるBlockoinINFURA.aixは、Infura.ioプラットフォームの40の機能を提供してくれます。詳細は、<https://infura.io/docs> にあるJSON-RPCのドキュメントを直接参照してください。

INFURA拡張機能を使用するにはinfura.ioサイトでアカウントを作成する必要があります。これは、Ethereumネットワークにクエリを送信したり、Ethereumテストネットワークを使用できるようにするためのKEY APIが必要だからです。

口座開設の方法は以下のように簡単です。



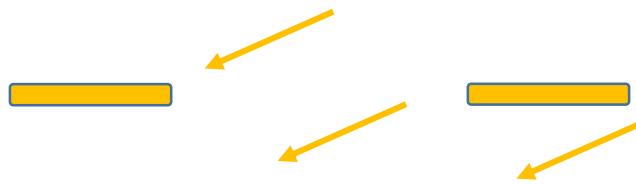
アカウントが作成されると、我々は入力し、我々は異なるネットワークのKEY APIを持つことができます。左上のEthereumをクリックして、プロジェクトが表示されるので、新しいプロジェクトを作成して、そのプロジェクトの自己紹介をします。

The screenshot shows the Infura.io dashboard for the Ethereum network. On the left, there's a vertical sidebar with buttons for 'ETHEREUM' (highlighted with a yellow arrow), 'FILECOIN', 'DOCS', 'COMMUNITY', and 'SUPPORT'. The main area is titled 'Ethereum' and shows a 'PROJECTS' section with one project named 'COINSOLIDATION' (created on September 12, 2020). The project card includes a status indicator ('Active'), a requests chart for the previous 7 days, and a plan section for 'Core' with 100,000 requests/day. There's also a link to see how other customers are using the Ethereum API. In the top right corner, there's an 'UPGRADE' button and a user profile icon.

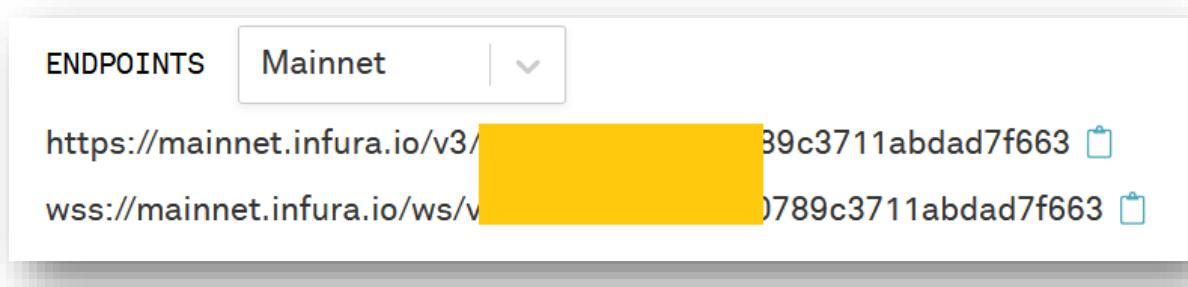
プロジェクト内では、KEY APIのデータと使用可能な様々なネットワークがあります。

プロジェクト内では、API KEY (PROJECT ID)を確認し、どのネットワークで作業するか、またはENDPOINTSの完全なリンクを選択することができます。

The screenshot shows the settings page for the 'COINSOLIDATION' project. The top navigation bar has tabs for 'REQUESTS' and 'SETTINGS' (highlighted with a yellow arrow). Below that is a 'SAVE CHANGES' button. The main area is titled 'COINSOLIDATION' and has a 'KEYS' section. It displays the 'PROJECT ID' (6101d7f0557440789c3711abdad7f663) and 'PROJECT SECRET' (6cf6b9d051ac4db1ac27e75434133134). Under the 'ENDPOINTS' dropdown (highlighted with a yellow arrow), it shows 'Mainnet' and a URL 'https://main...'. The sidebar on the left is identical to the one in the previous screenshot, with buttons for 'ETHEREUM' (highlighted with a yellow arrow) and 'FILECOIN'.



例えば、Ethereumのメインネットワークを使用するには、次のようなリグを取ることになります。

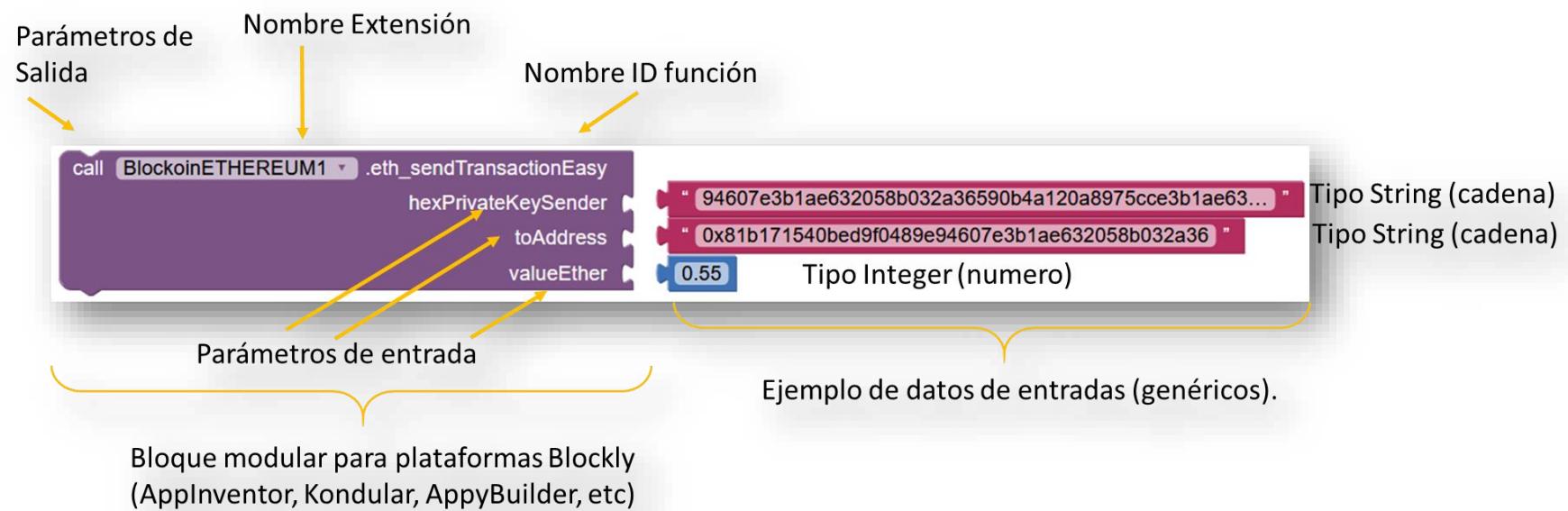


注：この拡張機能は、AppInventor、Kondular、Thunkable、AppyBuilderシステムでテスト済みです。

7. ブロックの定義と使い方（汎用機能

まずは、すべてのブロックが持つであろうデータの分布、使用の構文、構成について説明します。

次の例では、入力データのタイプと同様に、モジュールブロックとその入出力パラメータを見ることができ、これらのデータのタイプはString（文字列）またはInteger（整数または10進数）です。どのように使用されているかを示し、正しく機能するように設定します。



各モジュールブロックはその説明を持ち、入力パラメータとして使用される他のブロックの必須または任意の依存関係がある場合には名前が付けられ、統合プロセスが発表されます。

8. Exchange Ethereum Extension (EEE) の機能とイベント。

***実際のトランザクションを行いたい場合は、urlNetworkのネットワークを変更するだけで、メインネットのurlNetworkのネットワークを変更する必要があります。

インターネット接続を確認するためのブロック - (CheckInternetConnection)。

call BlockoinETHEREUM1 .CheckInternetConnection

入力パラメータ：該当しません。

出力パラメータ：接続がある場合は"True"を、接続がない場合は"False"を返します。

説明：インターネットの接続を確認し、データ（トランザクション）を送信するためのブロック。

新しい「オフライン」アドレスを生成するブロック - (GenerateNewAddressEthereum)

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

入力パラメータ：phraseHex <String>。

出力パラメータ：イベント(OutputGenerateNewAddressEthereum)

出力します。PrivateKey<String>、PublicKey<String>、addressEthereum<String>。

when BlockoinETHEREUM1 .OutputGenerateNewAddressEthereum
privKeyEther pubKeyEther addressEthereum
do

説明: フレーズや数字の列に基づいて、新しいethereumアドレス(アカウント)を作成します。新しいアドレスは、ネットワークやインターネット接続なしで作成することができます - "オフライン"。

新しい「オンライン」アドレスを生成するブロック - (GenerateNewAddressEthereum)

call BlockoinETHEREUM1 .GenerateNewAddressEthereumAPI

入力パラメータ：該当しません。

出力パラメータ：privateKey、publicKey、addressのJSONデータ形式で返します。

出力例。

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

説明: ethereumアドレス(アカウント)を新規作成します。生成はREST APIサービス - "Online"を介して行われるので、インターネットへのアクセスや接続が必要です。

新しい「オフライン」アドレスを生成し、公開鍵と秘密鍵をバイナリファイルに保存するブロック
(GenerateNewAddressEthereumStoreKeys)

call BlockoinETHEREUM1 .GenerateNewAddressEthereumStoreKeys

pathFilePrivateKey

“ /mnt/sdcard/privateKey.bin ”

pathFilePublicKey

“ /mnt/sdcard/publicKey.bin ”

入力パラメータ：pathFilePrivateKey<String> , pathFilePublicKey<String>.

出力パラメータ：イベント(OutptertGenerateNewAddressEthereumStoreKeys)

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

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

説明: 新しいランダムなethereumアドレス(アカウント)を作成し、公開鍵と秘密鍵をバイナリファイルに保存し、アカウントデータのインポートとエクスポートに使用します。新しいアドレスは、ネットワークやインターネット接続なしで作成することができます - "オフライン"。

公開鍵を生成するブロック - (GeneratePublicKeyHexFromPrivateKeyHex)。

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

入力パラメータ: hexPrivateKey <String>.

出力パラメータ : イベント(OutputGeneratePublicKeyHexFromPrivateKeyHex)

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

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

説明: 秘密鍵のエントリに基づいて公開鍵を作成します。

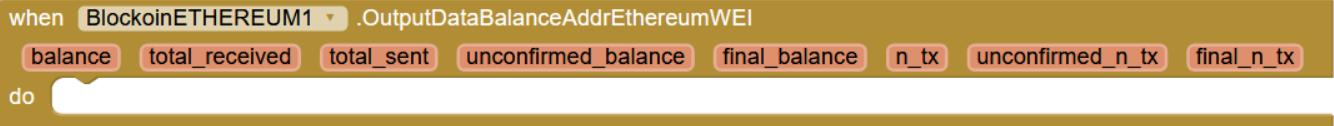
アドレスの残高を取得するブロック - (GetBalanceAddrEthereum)。

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

入力パラメータ: hexPrivateKey <String>.

出力パラメータ : イベント(OutputGeneratePublicKeyHexFromPrivateKeyHex)

出力: balance<String> , total_received<String> , total_sent<String> ,
 unconfirmed_balance<String> , final_balance<String> , n_tx<String> ,
 unconfirmed_n_tx<String> , final_n_tx<String>。



説明：貸借対照表と詳細勘定科目（住所）のデータを表示します。

モバイルにネットワーク・インターフェイスが有効になっているかどうかを確認するためにブロックします
 - (GetDataNetworkConnection)。

call [BlockoinETHEREUM1 v].GetDataNetworkConnection

入力パラメータ：該当しません。

出力パラメータ：イベント(OutputGeneratePublicKeyHexFromPrivateKeyHex)

出力: interfacename<String> , isconnected<String>。

```

when [BlockoinETHEREUM1 v].OutputGetDataNetworkConnection
  interfacename
  isconnected
do

```

説明：モバイルインターフェースの名前を表示し、インターフェースがアクティブになっているかどうかを配信します。

トランザクション「オフライン」に署名するためのブロック
 (SignederGenericPushRawTransactionOffline)

```

call [BlockoinETHEREUM1 v].SignederGenericPushRawTransactionOffline
  urlNetwork "https://rinkeby.infura.io/v3/440789c3..."
  hexPrivateKeySender "9eC478ee49824890b4a120a8975cce3b1ae632058b0301f8..."
  nonceNumber get global nonce
  gasPrice "125000000000"
  gasLimit 21000
  toAddress "0x92a2f157d5aec3fa79f92995fea148616d82c5ef"
  valueWei [1000000000000000000] * "0.01"

```

必要な依存関係: ブロック (eth_getTransactionCount)、ブロック (eth_SendRawTransactionInfura)

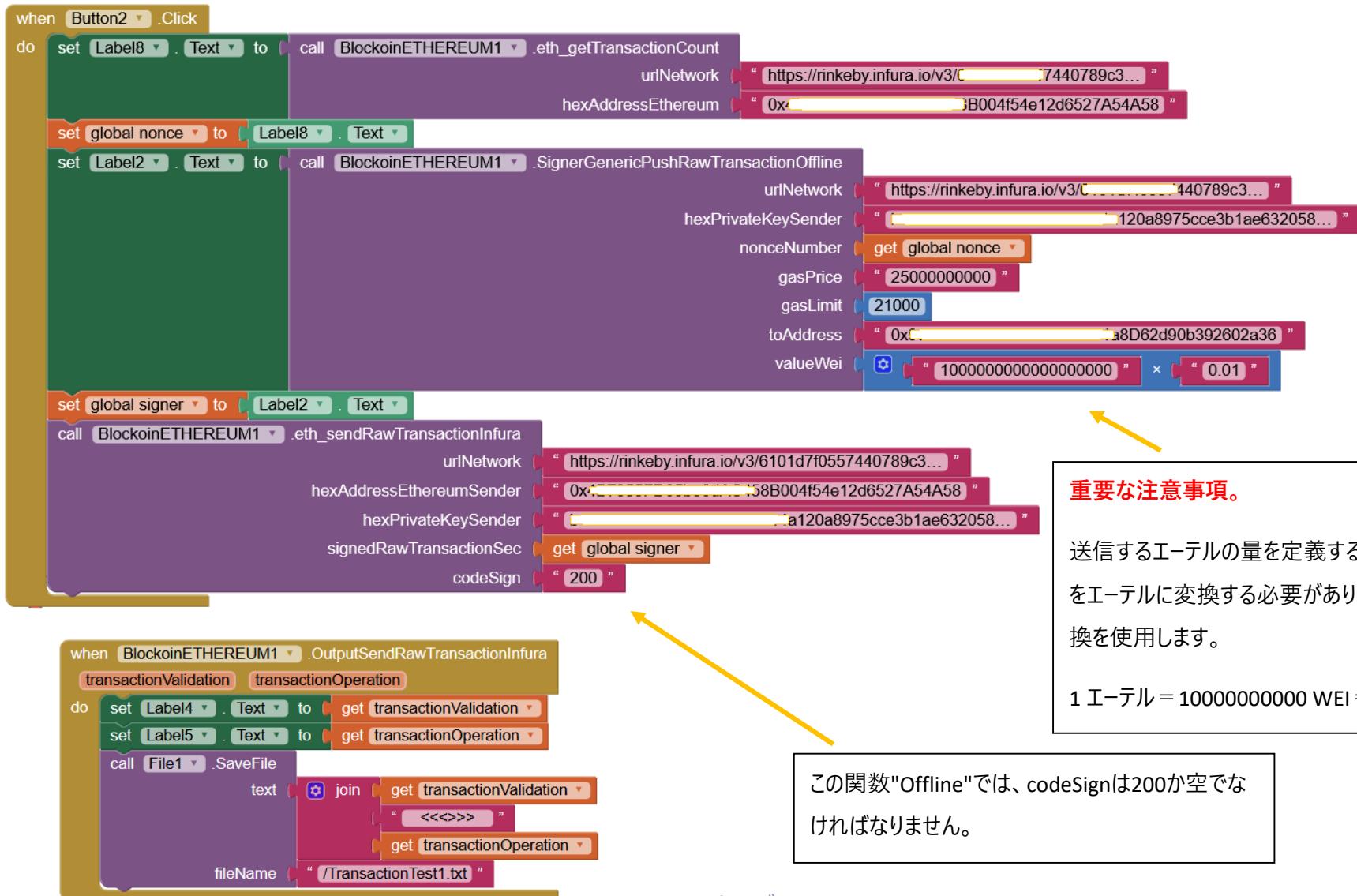
エントリ項目: urlNetwork <String>、hexPrivateKeySender <String>、nonceNumber <String>、gasPrice <String>、gasLimit <Integer>、toAddress <String>、valueWEI <Integer>。

出力パラメータ。

出力：送信される署名済みのトランザクション。<文字列>です。

説明: 送信される新しいトランザクションを準備します(暗号化され、署名されています)。これは、ネットワークやインターネットに接続していなくても処理することができます。

ブロック依存（SignerGenericPushRawTransactionOffline）での完全使用例。



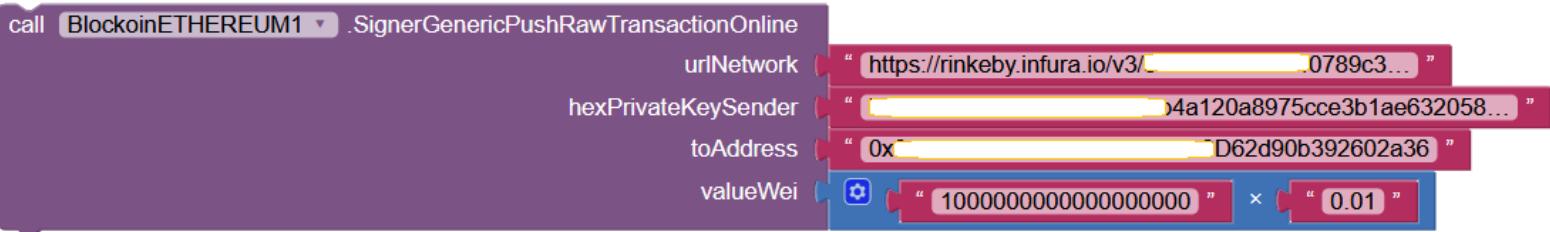
重要な注意事項。

送信するエーテルの量を定義するには、まずWEIをエーテルに変換する必要があります。以下の変換を使用します。

$$1 \text{ エーテル} = 10000000000 \text{ WEI} = 10^{18}$$

この関数"Offline"では、codeSignは200が空でなければなりません。

オンライン」トランザクションに署名するためのブロック



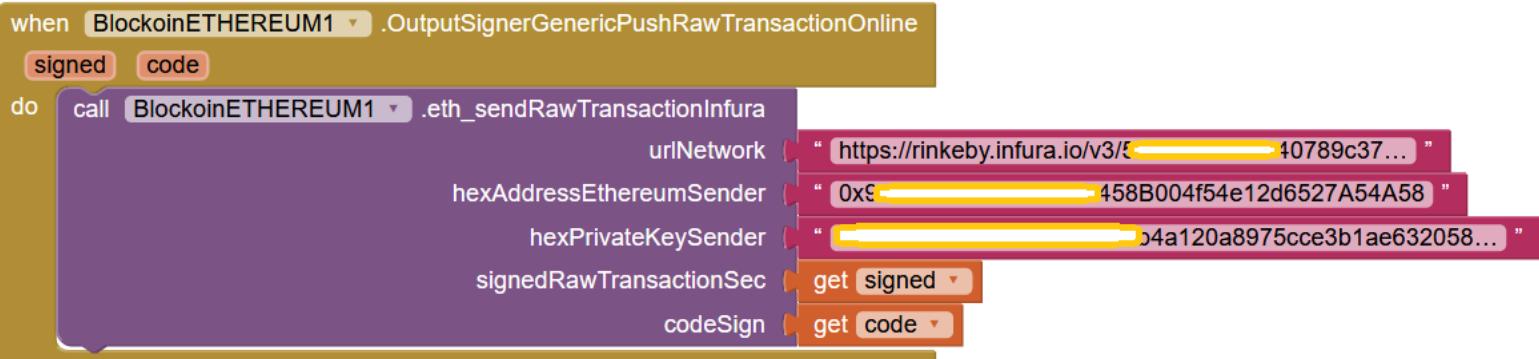
必須単位：ブロック(eth_SendRawTransactionInfura)。

入力パラメータ: urlNetwork <String>、hexPrivateKeySender <String>、toAddress <String>、valueWEI <Integer>。

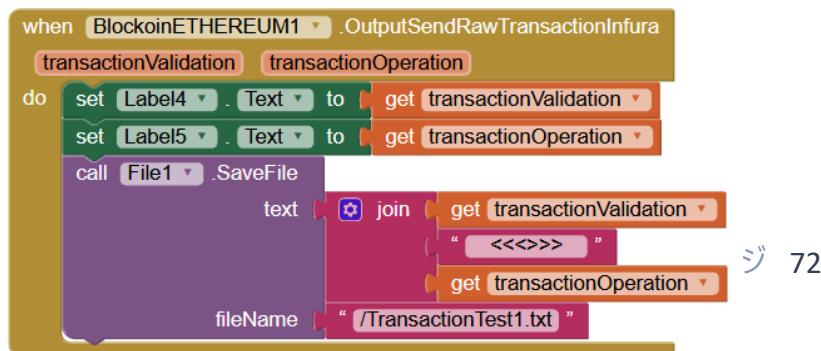
出力パラメータ：以下の順序で使用されるイベント（
OutputSignerGenericPushRawTransactionOnline）と（OutputSendRawTransactionInfura）

◦

出力: signed<String> , code<String>.

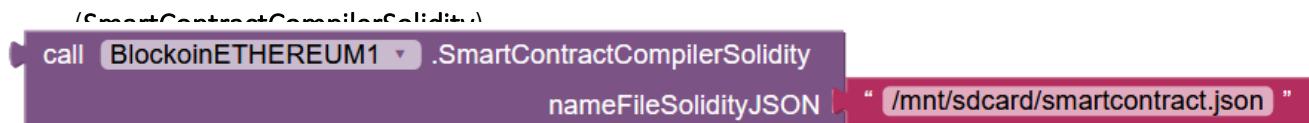


出力ブロックeth_sendRawTransactionInfura: transactionValidation<String> , transactionOperation<String>.



説明: 送信される新しいトランザクションを準備します(暗号化され、署名されています)。ネットワークまたはインターネット接続が必要です。

スマートコントラクトを「オンライン」でコンパイルするためのブロック



入力パラメータ: nameFileSolidityJSON <String>.

出力パラメータ: コンパイルされたスマートコントラクトを表示します。この関数は、Ethereumネットワークでスマートコントラクトを公開する前に、スマートコントラクトがきちんと書かれているかどうかをチェックするのに役立ちます。

出力：コンパイルされたコード。

スマートコントラクトはJSON形式のファイルである必要があります。

SOLIDITY言語でのスマート基本契約の例

ソースコード v0.5.0。

```
宿泊契約
アドレスの所有者です。
関数 mortal() { owner = msg.sender; }
function kill() { if (msg.sender == owner) suicide(owner); } } { if
(msg.sender == owner) suicide(owner)
コントラクトクリエイターは自動的に死んでしまう（△）
文字列の検索
function greeter(string _greeting) public { greeting = _greeting; }
greater(string _greeting)
関数 greet() constant returns (string) {return greeting;}
```

JSON形式での過去のスマートコントラクトの例をコメント付きで掲載しています。

```
# 非表示のテストで空のコンパイルをチェックする
# "greeter"契約通りの動作を使って Ethereum の hello world。
```

ファイル：smartcontract.json

```
{
  "solidity": "contract mortal {\\n /* Define variable owner of the type
address      */\\n address owner;\\n      /* this function is executed at
initialization and sets the owner of the contract */\\n      n function
mortal() { owner = msg.sender; } }      n /* 約束の資金回収に対する処理 */\\n
function kill() { if (msg.sender == owner) suicide(owner); }\\n}\\non 純粋の
greeter is mortal {\\n      n /* define variable greeting of the type string
*/\\n      n string greeting.\\n      * this runs when the contract is executed */\\n
function greeter(string _greeting) public {\\n          greeting =
_greeting;\\n      } \\n      * main function */\\n      function greet() constant
returns (string) {          n return greeting;\\n      } }"
"params": ["Hello Coinsolidation Test"]
}
```

重要な注意: JSONフォーマットでは、必ず各行の最後に改行を入れなければなりません。

コンパイルされたSmartcontractの出力例

```
[
{
  "name": "u003cstdinu003e:greeter".
  "solidity": "contract mortal {\\n /* Define variable owner of the type
address      */\\n address owner;\\n      /* this function is executed at
initialization and sets the owner of the contract */\\n      n function
mortal() { owner = msg.sender; } }      n /* 約束の資金回収に対する処理 */\\n
function kill() { if (msg.sender == owner) suicide(owner); }\\n}\\non 純粋の
greeter is mortal {\\n      n /* define variable greeting of the type string
*/\\n      n string greeting.\\n      * this runs when the contract is executed */\\n
function greeter(string _greeting) public {\\n          greeting =
_greeting;\\n      } \\n      * main function */\\n      function greet()
constant returns (string) {          n return greeting;\\n      } }"
"bin": "606060405260405161023e38038061023e83398101604052805101600080546001
60a060020a031916331790558060016000509080519060200190828054600181600116156
101000203166002900490600052602060002090601f016020900481019282601f10609f57
805160ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565
b50505061016e806100d06000396000f35b828001600101855582156076579182015b8281
1115607657825182600050559160200191906001019060b0565b509056606060405260e06
0020a600035046341c0e1b58114610026578063cfae321714610068575b005b6100246000
543373fffffffffffff908116911614156101375760005
473fffffffffffff16ff5b6100c9600060609081526001
805460a06020601f600260001961010086881615020190941693909304928301819004028
1016040526080828152929190828280156101645780601f10610139576101008083540402
83529160200191610164565b6040518080602001828103825283818151815260200191508
0519060200190808383829060006004602084601f0104600f02600301f150905090810190
601f1680156101295780820380516001836020036101000a031916815260200191505b509
25050506
  "abi": [
}
```

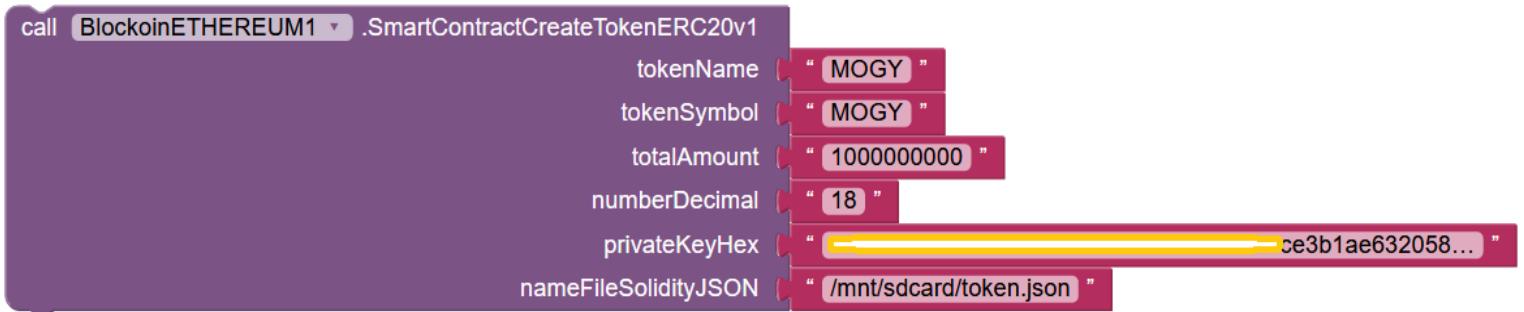


```

        "type": "function"
    },
    {
        "入力": [
            "タイプ": "ビッグ"
        ]
    ],
    "params": [
        "Hello Coinsolidation Test"
    ]
}
]
]

```

ERC20トークンをコンパイル、作成、公開するブロック - (SmartContractCreateTokenERC20v1)

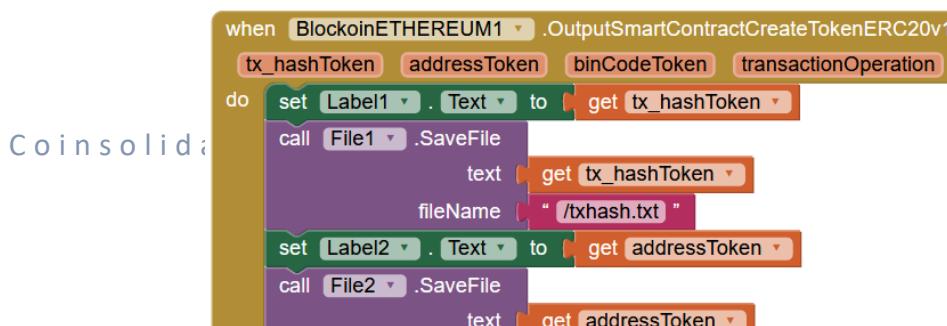


必須単位（複数可）：ブロック（CreateTestingFie）。これは、ブロック（SmartContractCreateTokenERC20v1）で有効なパスを与えなければ、トークンの作成が実行されないからです。

パラメータを入力します。tokenName <String>, tokenSymbol <String>, totalAmount <String>, numberDecimal <String>, privateKeyHex <Integer>, nameFileSolidityJSON <String> このファイルは、一時ファイルを作成するための有効なパスであり、ファイルが作成されたことをテストするためにパスが有効であることを確認する必要があります。

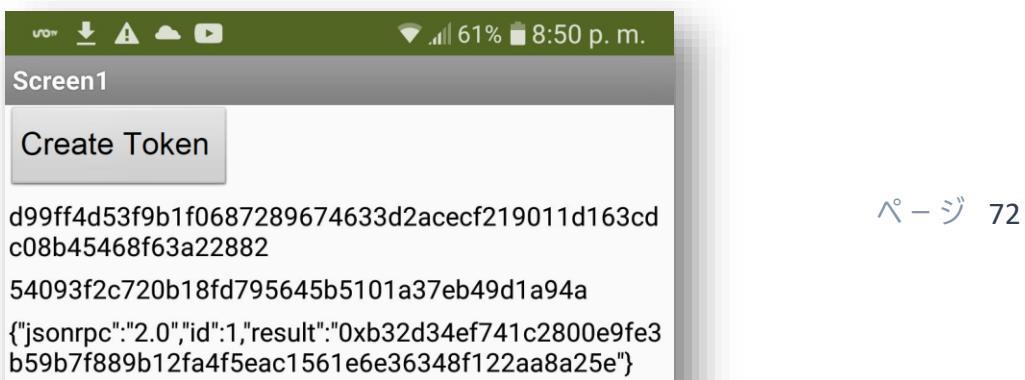
出力パラメータ：イベント（OutputSmartContractTokenERC20v1）。

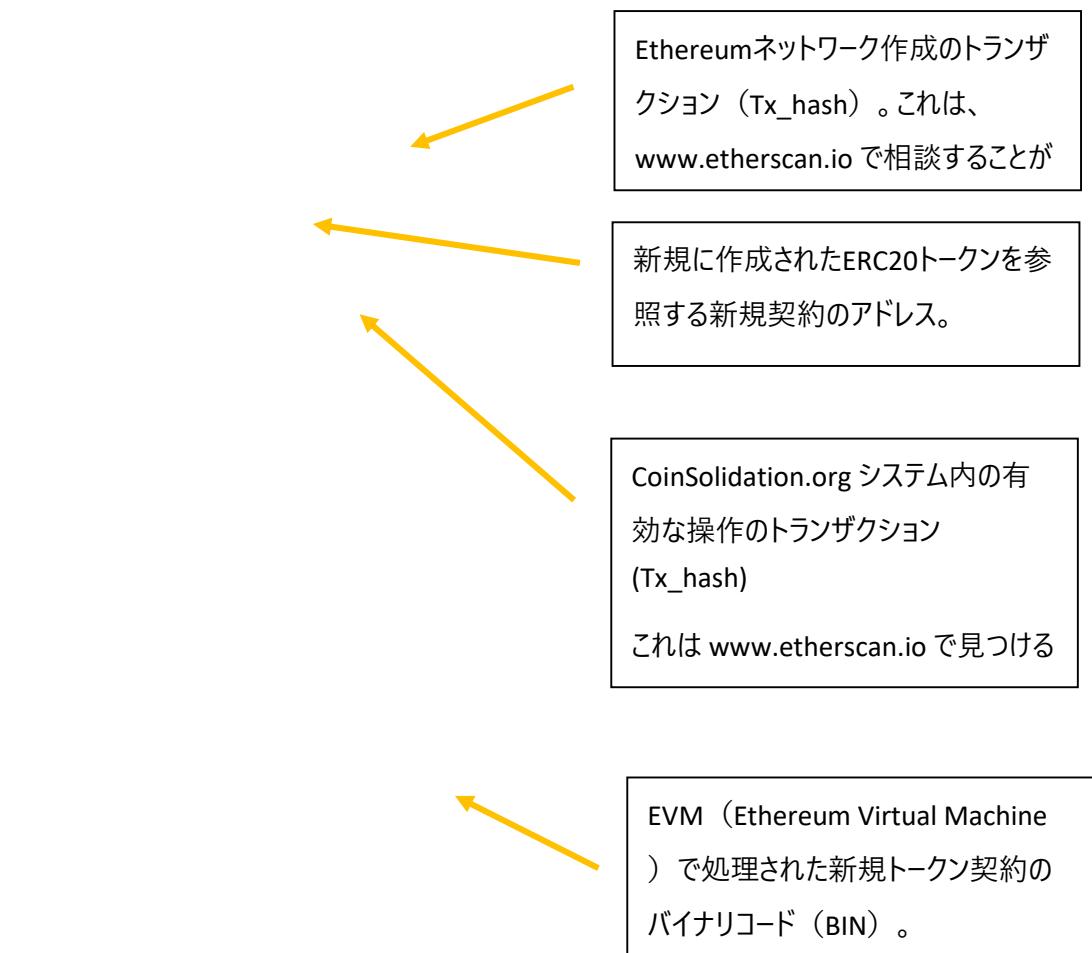
出力: tx_hashToken<String> , addressToken<String> , binCodeToken<String> , transactionOperation<String>.



説明：スマートコントラクト「トークンERC20」-Ethereumネットワーク上でアクティブに公開されています。バージョン1(v1)では、すでにガスリミットパラメータが500,000 WEIに設定されています。

前回の関数SmartContractCreateTokenERC20v1の出力例です。





9. CryptoTokenまたはCryptomoney Tokenを作成する手順です。

ステップ1.

スマート契約を作成したモバイル端末の一時パスが有効で、ファイルが正常に作成できることを確認します。これはブロック(CreateTestingFile)を使って行います。入力"pathTestFile"で与えられたテストファイルが作成されていることを確認してください。

ステップ2（オプション）。

このステップでは、操作の課金元となる口座（アドレス）に、スマート契約の作成と発行のトランザクションを実行するのに十分な残高があるかどうかを確認します。これは、ブロック（**eth_VerifilibalanceForTransaccionSmartContract**）を使用して検証することができます。
SmartContractCreateTokenERC20v1または**SmartContractCreateTokenERC20v2**を生成するブロックは、すでにこの検証を内部的に含んでいるので、このブロックの使用はオプションです。

ステップ3.

ERC20トークンを作成するために使用するブロックを選択するには、2つのオプションがあります。

- a.- ブロック**SmartContractCreateTokenERC20v1**は、すでに500,000 WEiの値で割り当てられたGas Limitの暗黙の値を持っています。
- b.- ブロック**SmartContractCreateTokenERC20v2**には、エンドユーザー や開発者の必要性に応じてガスリミットを構成できるオプションがあります。35万魏以下の非常に低いガスリミットが与えられた場合、スマートコントラクトが非常に可能性があることに注意する必要があります。

第四段階

SmartContractCreateTokenERC20v1または**SmartContractCreateTokenERC20v2**ブロックを使用して、入力変数"nameFileSolidityJSON"を使用する際には、ステップ1で既にチェックしたブロックの"pathTestFile"入力変数(**CreateTestingFile**)と等しいことを確認してください。

ステップ5

SmartContractCreateTokenERC20v1ブロックまたは**SmartContractCreateTokenERC20v2**ブロックのいずれかでERC20トークンの作成を実行する前に、イベント値（結果）を適宜保存しておくことを推奨します（tx_hashToken、addressToken、binCodeToken、transactionOperation）。先ほどの関数**OutPutSmartContractCreateTokenERC20v1**の出力例を参照してください。

ステップ6

ERC20トークンの作成を実行し、販売用に公開します。10項を参照してください。

10.新しい資産や自分のクリプトトークンを売りに出す方法(トークンERC20)

ERC20 - クリプトモネイ・トークン（SmartContractCreateTokenERC20v1ブロックまたはSmartContractCreateTokenERC20v2ブロックを参照）を作成したので、世界中の誰もがそれを購入できるように、どこかの取引所にアップロードしなければなりません。取引所とは、新しいトークンが公開されているインターネット上のサイトです。

取引所は、中央集権型と分散型の2種類に分類されます。主な違いは、ある種の国際機関（中央集権型）によって統治され、監査されていることと、地方集権型のものは監査人との間に誰もいないことです。この方が自信を持てるかもしれません、最近では分散型の方が力を持ち、大きな問題なく使われているのが実情です。

あらゆる種類の資産を取り扱う際のベストプラクティスの一つは、秘密鍵と公開鍵の両方のセキュリティを確保するために、1つのアカウントにすべての資産を持たせるのではなく、複数のアカウントに分散されることです。

私たちのケースでは、我々は分散型Exchangeを使用しますが、すでに悪くない歴史を持つ、我々はwww.forkdelta.appを使用します。

この時点では我々はすでにブラウザ（MozillaまたはChorme）METAMASK www.metamask.io これはあなたのページを訪問するためのExchangeがあるため、アプリケーションをインストールしている必要があります www.forkdelta.app 我々はEthereumを持っているアカウントに接続する必要があります。

重要なのは、METAMASKで既に保有しているEthereumアカウントの残高が10米ドル以上あることです。これは、SmartContractCreateTokenERC20v1ブロックまたはSmartContractCreateTokenERC20v2ブロックで作成した新しいERC20トークンを公開する際に、取引所に公開するための取引を支払う必要があるからです。

取引所 www.forkdelta.app を利用するには、取引所で販売するために公開したい以下のトークンデータが必要です。

先ほど作成した新しいERC20トークンのアドレスです。

0x54093F2C720b18Fd795645b5101A37EB49d1A94a

タッチが使用する小数の数。

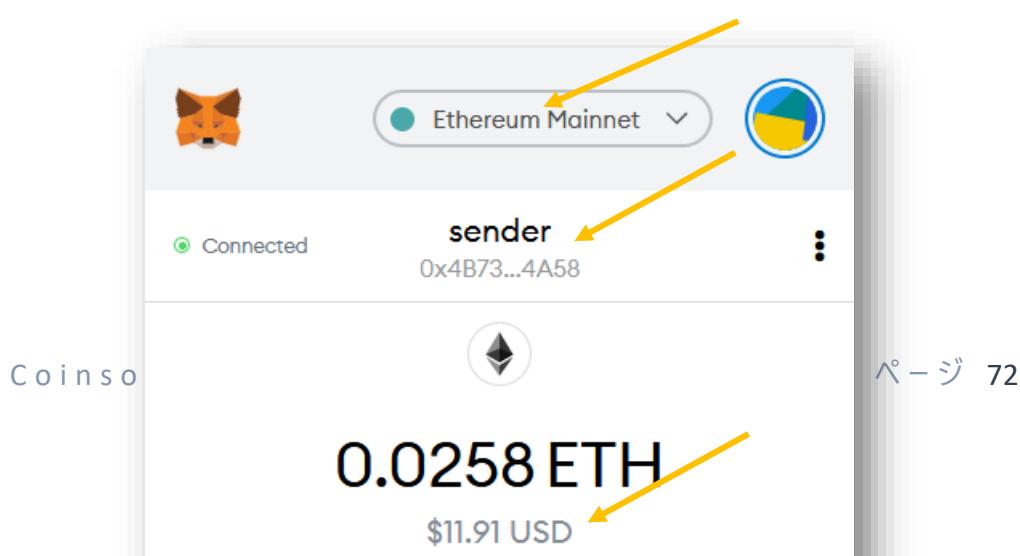
18

トークンを識別する名前。

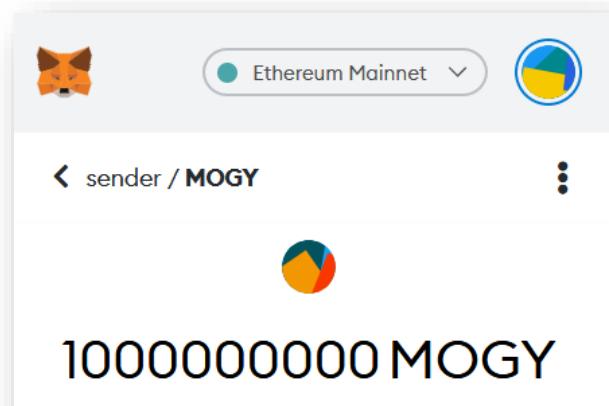
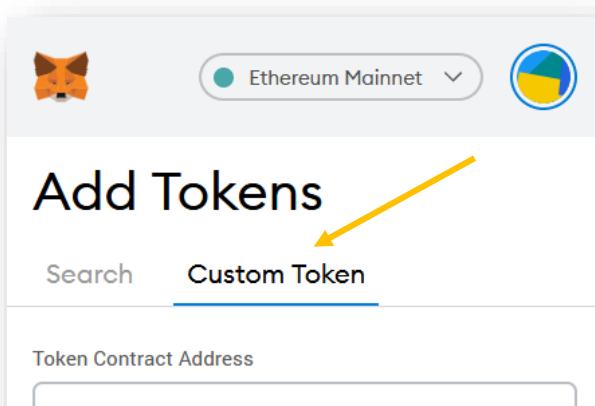
モギー

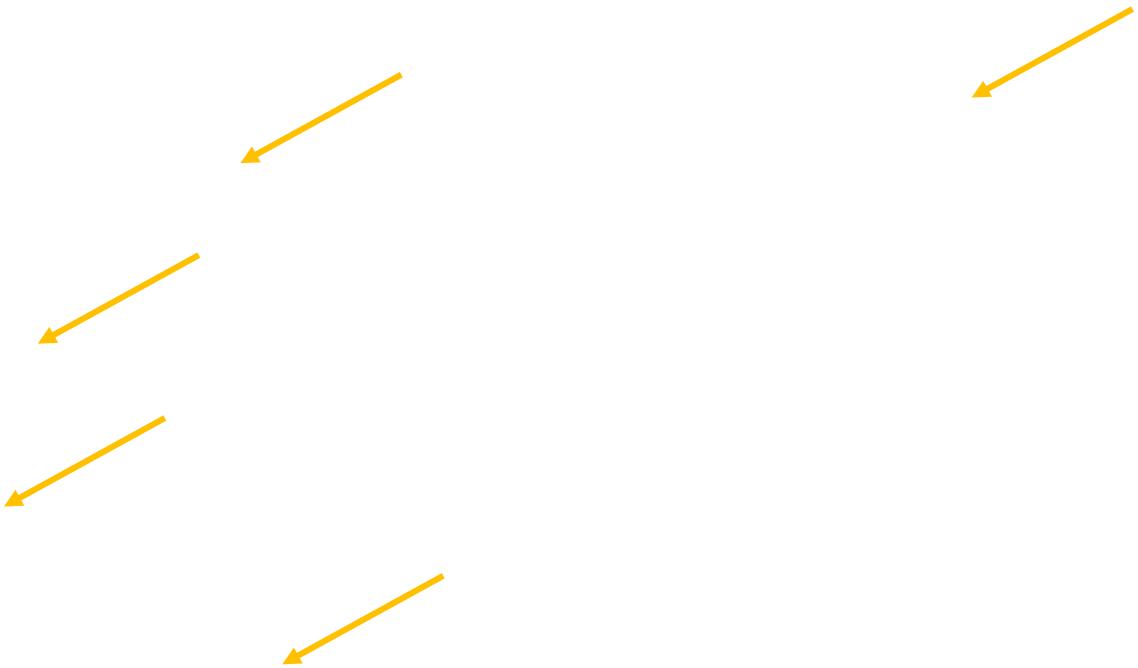
まずは、作成したトークンが上記のパラメータを持っているか、またそのパラメータが最初に作成したものであるかどうかを確認してみましょう。

METAMASKに行って、まずトークンを作成したアカウントであることを確認しましょう。その後、下の方に移動して「トークンの追加」ボタンをクリックします。



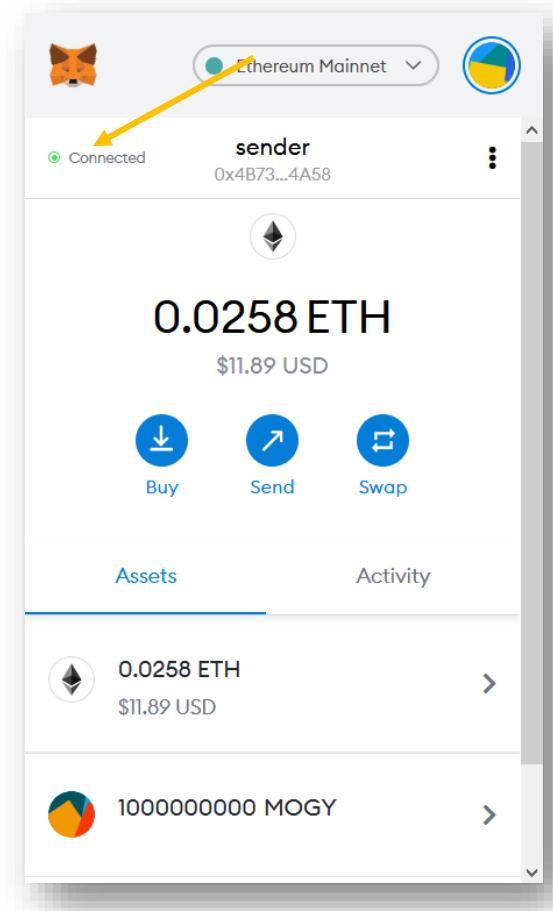
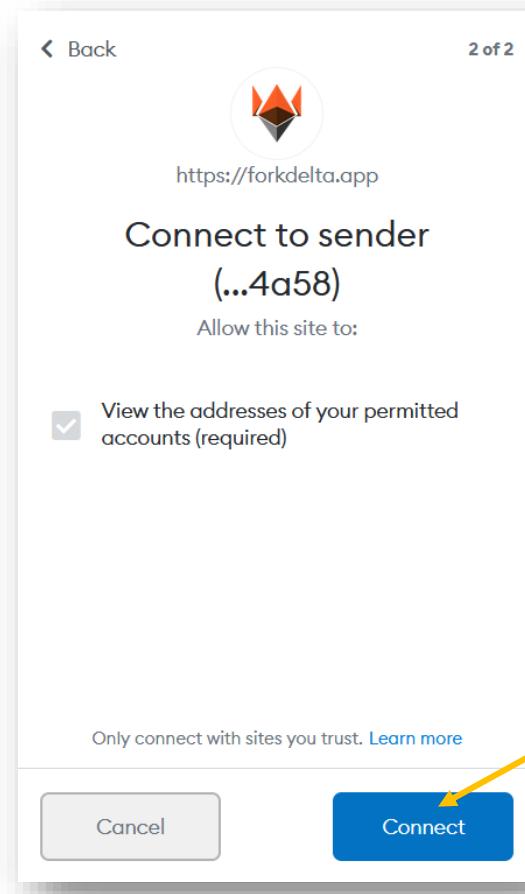
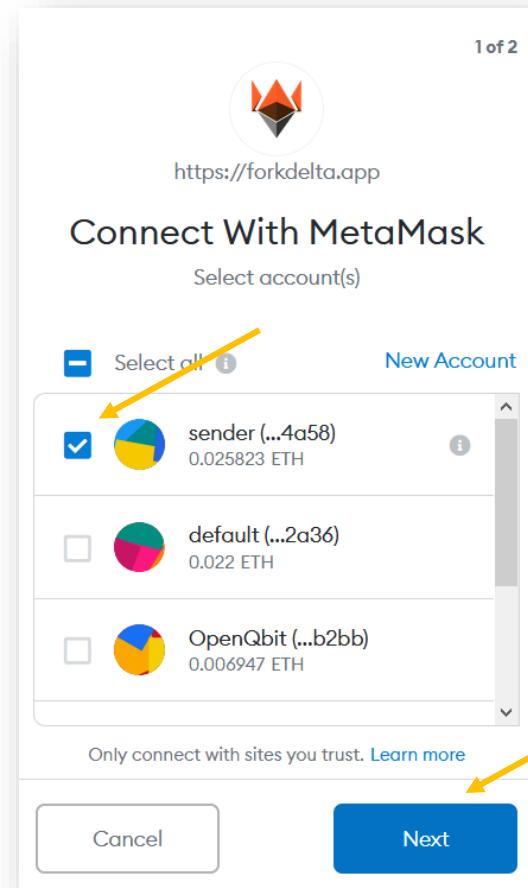
これで新しいトークンを追加して、現在の残高を確認することができます。ページ上部の「カスタムトークン」ボタンをクリックした後、以下の項目に必要な情報を入力し、「次へ」ボタンをクリックしてください。その後、新しいトークンは、あなたが持っている残高と一緒に表示されます。残高がない場合は、トークンを作成したアカウントに入っているかどうかを確認し、トークンを作成したアカウントに入っていない場合は、まだトークンを購入していないので残高はゼロになります。





取引所で販売するために新しいトークンをアップロードしたらすぐに [www.forkdelta.app](https://forkdelta.app)

Exchangeのサイトに入っていると、<https://forkdelta.app>、下の「次へ」ボタンをクリックして、「接続」をクリックして、最後にforkdelta.appのサイトに接続されていることを確認することができます。



新しいトークンを取引所で公開する時が来た <https://forkdelta.app>

上部メニューの"DAI"をクリックして、スクロールの最後に移動し、オプション"その他"を選択します。

The screenshot shows the ForkDelta interface. At the top left, there's a dropdown menu labeled "DAI". A yellow arrow points from the text above to this menu. Below it, a sub-menu is open with several options: "ZAP TOKEN", "ZCG ZCashGOLD", "ZDR Zloadr Token", "ZIL Zilliqa", "ZIP ZipCoin", "ZRX 0x Protocol Token", "ZSC Zeus Shield Coin", "ZXBT ZeroXBToken Project 0xbt", "cV cVToken", "eGO GoGuides", and "ePRX eProxy". Another yellow arrow points from the text above to the "Other" option at the bottom of this list.

The main trading area includes an "Order Book", a "Price Chart" showing price vs depth, and a "Trades" table listing recent trades for DAI/ETH, DAI, and ETH. The "My Transactions" section shows a history of activity. On the right, there's a "Tweets" feed by @ForkDelta and a "URL & Status" section with a bookmark link.

そして、すでに知っているデータで新しいトークンを登録します。

The screenshot shows the ForkDelta application interface. A modal dialog box titled "Other token" is open in the center. It contains three input fields: "Address" with the value "0x54093F2C720b18Fd795645b5101A37EB49d1A94a", "Name" with the value "MOGY", and "Decimals" with the value "18". Below these fields are two buttons: "Cancel" and "Go".

On the left side of the screen, there is a "Balance" section showing DAI and ETH amounts, and a "Volume" section showing various tokens like ASTRO, REQ, SXDT, and SNOV with their current prices. On the right side, there is a "Trades" section listing recent trade history and a "My Transactions" section.

At the bottom of the page, there is a "Tweets" section by @ForkDelta, which includes a link to the project's URL and some promotional text about DeFi and the project's goals.

この時点では、新しいトークンは取引所の左上に表示されますが、取引所にアップロードしただけで、みんなが買って見れるように公開する必要があります。今、我々はいくつかのエーテル（0.015）を入金する必要がありますが、取引所に私たちのアカウントから十分です。

The image consists of two side-by-side screenshots of the ForkDelta website. Both screenshots show the 'Balance' section where tokens can be deposited.

Left Screenshot:

- The token listed is MOGY.
- The 'Wallet' column shows 1000000000.000.
- The 'ForkDelta' column shows 0.000.
- A yellow arrow points from the 'Token' label to the MOGY entry.
- A yellow arrow points from the 'Wallet' label to the 1000000000.000 value.
- A blue 'Deposit' button is visible.

Right Screenshot:

- The token listed is MOGY.
- The 'Wallet' column shows 1000000000.000.
- The 'ForkDelta' column shows 0.000.
- A yellow arrow points from the 'Token' label to the MOGY entry.
- A yellow arrow points from the 'Wallet' label to the 1000000000.000 value.
- A blue 'Deposit' button is visible.
- A tooltip appears over the 'Deposit' button: "Use this to deposit from your personal Ethereum wallet ("Wallet" column) to the current smart contract ("ForkDelta" column)."
- The 'Amount' input field for ETH has '0.015' typed into it.

入金をしたので、取引所には何個でもトークンを入金することができます www.forkdelta.app

決められた金額のトークンを入金するには、購入注文を作成する必要があります。次に、世界のあらゆる買い手に利用可能にしたいトークンの量を入力し、それを販売したいイーサの価格（単価）を入力し、"Expires"パラメータには、これらのトークンを販売したい時間の量を入力します。

Expires Time = 14秒 × 入力された量。

例：14秒×10000 = 140,000秒 = 1.62日

The screenshot shows the ForkDelta app interface for creating a new order. On the left, there's a 'Volume' section listing various tokens with their current prices. On the right, the 'New Order' form is displayed with the following details:

- Sell:** MOGY (amount: 10000)
- MOGY/ETH:** 0.05
- ETH:** 500.000
- Expires:** 10000

Arrows from the text above point to the 'Sell' button, the MOGY amount input, the MOGY/ETH price input, and the Expires time input respectively.

そこに、あなたの新しいトークンが販売されているので、誰でも買いに来てください。新しいトークンを認識しないことがあるので、メタマスクアプリケーションから販売する必要があります。

新規作成したトークンのサイトwww.etherscan.ioへの相談例

The screenshot shows the Etherscan Transaction Details page for a specific Ethereum transaction. The transaction hash is 0xd99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882. The transaction was successful and included in block 11240201, which has 224 block confirmations. It occurred 47 minutes ago on Nov-12-2020 at 02:49:56 AM UTC. The transaction was confirmed within 30 seconds. The 'To' field indicates the creation of a new contract at address 0x54093f2c720b18fd795645b5101a37eb49d1a94a. The transaction fee was 0.011014691 Ether (\$5.06). The gas price was 0.000000041 Ether (41 Gwei) and the gas limit was 500,000. The gas used by the transaction was 268,651 (53.73%).

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

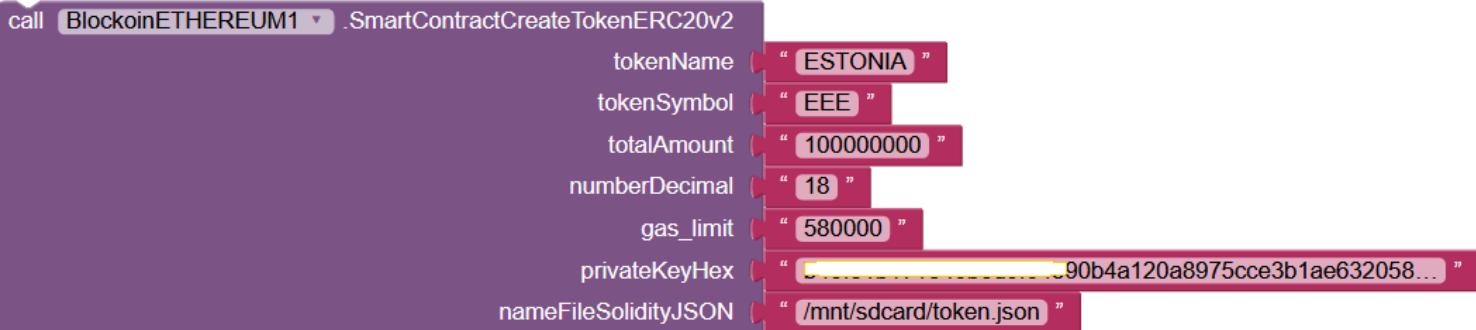
② Gas Used by Transaction: 268,651 (53.73%)

Ethereumネットワーク作成のトランザクション (トランザクション)

契約アドレス 新たに作成されたトークン

Ethereumのネットワークコストのみ。
運営費+15ドルは含まれていません。

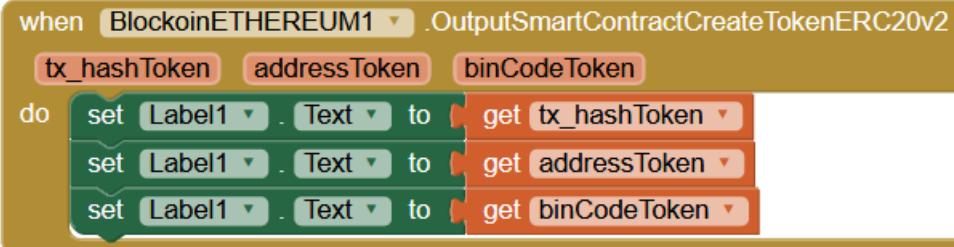
ERC20トークンをコンパイル、作成、公開するブロック - (SmartContractCreateTokenERC20v2)



エントリ項目: `tokenName <String>`, `tokenSymbol <String>`, `totalAmount <String>`, `numberDecimal <String>`, `gas_limit <Integer>`, `privateKeyHex <Integer>`, `nameFileSolidityJSON <String>`.

出力パラメータ：イベント（OutputSmartContractTokenERC20v2）。

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



説明：スマートコントラクト「トークンERC20」-Ethereumネットワーク上で公開されている資産。バージョン2（v2）は、ガスリミットの値を最適化することができますので、スマートに応じて契約どのように速く、あなたはイーサリアムネットワーク上の公開を行いたい。失敗や遅延のない出版のためには、最低でも35万WEIの値とすることを推奨する。

TokenERC20v1の作成機能とTokenERC20v2の作成機能の違いは、バージョン1の場合はGasLimitのパラメータがあらかじめ設定されており、バージョン2ではユーザー・開発者のニーズに合わせて設定できる点です。

ERC20トークンの呼び出しまだは実行をブロック - (SmartContractExecution)



入力パラメータ: addressSmartContract<String>、nameFileSolidityJSON<String>、functionExecutionSmartContract<String>。

出力パラメータ：参照されたスマートコントラクトで指定された関数の実行。

出力：スマートコントラクトの実行

注：実行するには、スマート契約を実行したいアドレスの主キーのパラメータを含むJSON形式でファイルを作成する必要があります、あなたはガスの制限（WEI）を入力する必要があります。

上記のコンパイラ機能例におけるコンパイルされたスマートコントラクトの機能を実行するためのJSONファイルの例ファイル名は任意のものを指定することができます。

ファイル: call.json

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

スマートコントラクトの「挨拶」機能の出力例です。

```
{  
  "gas_limit": 20,000,  
  "address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",  
  "結果": [...  
    "Hello Coinsolidation Test"  
  ]  
}
```

ERC20トークンプロパティ表示ブロック - (SmartContractGetCreationTx)

call BlockoinETHEREUM1 .SmartContractGetCreationTx
txSmartContract " d99ff4d53f9b1f0687289674633d2acecf219011d163cdc0..."

入力パラメータ: txSmartContract<String>

出力パラメータ : (Tx_hash)で参照しているスマートコントラクトのプロパティを表示します。

説明: 参照するスマートコントラクトの主な機能とABIコードを表示します。

ERC20トークンのプロパティ例、以前に関数(martContractCreateTokenERC20v1)を使用して作成されたtokenName "MOGY"のサンプルです。

```
{  
  "block_hash":  
    "cadb8914c43ed28fb370c9e1b6f5d8818ba31a1e944d1f7049441b982902dea8",  
  "block_height": 11240201,  
  "block_index": 170,  
  "hash":  
    "d99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882",  
  "アドレス": [...  
    "4b7355fd05be6dac458b004f54e12d6527a54a58"  
  ],  
  "合計": 0  
  "米金": 110146910000000.  
  "サイズ": 958  
  "gas_limit": 500,000.  
  "gas_used": 268651.  
  "gas_price": 41000000000.  
  "contract_creation": true.  
  "reableed_by": "200.77.24.87".  
  "confirmed": "2020-11-12T02:49:56Z",  
  "received": "2020-11-12T02:50:13.185Z",
```


ERC20トークンのコンパイルコードを表示するブロック - (SmartContractGetcodeABI)

call BlockoinETHEREUM1 .SmartContractGetcodeABI
 addressSmartContract " 0eb688e79698d645df015cf2e9db5a6fe16357f1 "

入力パラメータ : addressSmartContract<String>

出力パラメータ : 参照されているスマートコントラクトコードを表示します。

説明: 参照されているスマートコントラクトのABIコードを表示します。

一般的なスマートコントラクトのABIコードの例。

```
{
  "solidity": "0.4.21",
  "contract": "mortal",
  "inputs": [],
  "outputs": [],
  "functions": [
    {
      "name": "kill",
      "type": "function",
      "constant": false,
      "inputs": [],
      "outputs": [],
      "body": "address owner;\n        /* this function is executed at\n         initialization and sets the owner of the contract */\n        n\n        mortal() { owner = msg.sender; } }\n        n /* 繰り返し処理に対するループ */\n        function kill() { if (msg.sender == owner) suicide(owner); }\\n}\n        non 繰り\n        greeter is mortal {\n        n /* define variable greeting of the type string */\n        n string greeting.\n        /* this runs when the contract is executed */\n        function greeter(string _greeting) public {\n        greeting = _greeting;\n        } }\n        main function *\n        function greet()\n        constant returns (string) {\n        n return greeting;\n        } }\n        \"bin\": \"0x606060405260405161023e38038061023e83398101604052805101600080546001\n        60a060020a031916331790558060016000509080519060200190828054600181600116156\n        101000203166002900490600052602060002090601f016020900481019282601f10609f57\n        805160f19168380011785555b50608e9291505b8082111560cc57600081558301607d565\n        b50505061016e806100d06000396000f35b828001600101855582156076579182015b8281\n        1115607657825182600050559160200191906001019060b0565b509056606060405260e06\n        0020a600035046341c0e1b58114610026578063cfae321714610068575b005b6100246000\n        543373ffffffffffffffffff908116911614156101375760005\n        473fffffffffffff16ff5b6100c9600060609081526001\n        805460a06020601f600260001961010086881615020190941693909304928301819004028\n        1016040526080828152929190828280156101645780601f10610139576101008083540402\n        83529160200191610164565b6040518080602001828103825283818151815260200191508\n        0519060200190808383829060006004602084601f0104600f02600301f150905090810190\n        601f1680156101295780820380516001836020036101000a031916815260200191505b509\n        25050506\n        \"abi\": [ {\n        \"constant\": false,\n        \"inputs\": [],\n        \"name\": \"kill\",\n        \"outputs\": [],\n        \"type\": \"function\"\n        }, {\n        \"constant\": true,\n        \"inputs\": [],\n        \"name\": \"greet\",\n        \"outputs\": [{\n        \"name\": \"\", \"type\": \"string\"\n        }],\n        \"type\": \"function\"\n        }, {\n        \"inputs\": [{\n        \"name\": \"_greeting\", \"type\": \"string\"\n        }],\n        \"type\": \"function\"\n        }, {\n        \"inputs\": [\n        {\n        \"name\": \"constructor\"\n        }\n        ],\n        \"creation_tx_hash\": \"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9\",\n        \"created\": \"2016-07-20T01:54:50Z\",\n        \"address\": \"0eb688e79698d645df015cf2e9db5a6fe16357f1\"}\n      ]\n    }
```

*以下のブロック（SmartContractPublish）を使用する前に、ブロック（SmartContractCompilerSolidity）を使用して、スマートコントラクトがきちんと書かれているかどうかを確認する必要があります。

ERC20トークンをEthereumネットワーク上で公開するためのブロック - (SmartContractPublish)

```
call BlockchainETHEREUM1 .SmartContractPublish  
    nameFileSolidityJSON " /mnt/sdcard/PublishSmartContract.json "
```

入力パラメータ : nameFileSolidityJSON<String

出力パラメータ：参照しているスマートコントラクトのプロパティを表示します。イベント（OutputSmartContractPublish）で。

The screenshot shows a visual programming interface for blockchain development. At the top, there's a header bar with the text "when BlockchainETHEREUM1 .OutputSmartContractPublish". Below this, there are two main sections: "tx_hash", "addressContract", and "binCode" (each in a pink rounded rectangle), and a "do" block (in a green rounded rectangle) containing three "set" statements. Each "set" statement has a dropdown menu next to the label ("Label1", "Label2", "Label3") and a "Text" dropdown next to "to". To the right of each "Text" dropdown is a "get" button followed by another rounded rectangle containing either "tx_hash", "addressContract", or "binCode".

説明: JSONファイルで参照されているスマートコントラクトをethereumネットワークで公開します。

ファイル例：PublishSmartContract.json

```
{  
"solidity": "contract mortal {\n/* Define variable owner of the type  
address */\naddress owner;\n/* this function is executed at  
initialization and sets the owner of the contract */\nn function  
mortal() { owner = msg.sender; } }\n/* 約束の資金回収に対する数 */\nfunction kill() { if (msg.sender == owner) suicide(owner); }\n\nnon 約束的  
greeter is mortal {\n/* define variable greeting of the type string */\nn string greeting.\n/* this runs when the contract is executed */\nfunction greeter(string _greeting) public {\n            greeting =  
_greeting;\n        } }\n/* main function */\nfunction greet()  
constant returns (string) {\n            return greeting;\n        } }"  
"params": "ハロースト".  
"/ソルシユ": ["挨拶"]。  
"private": "3ca40...".  
"gas_limit": 500000  
}
```

上記のコードに示すように、それは同じJSONファイルの最後にパラメータが追加されている同じコードにコンパイル関数の例で使用されるのと同じJSONコードです。

Params: Smart contrの暗黙のパラメータ。

発行：スマート契約をどのように発行するかの名称。

プライベート：スマートコントラクトを実行するアカウントの主キーが残高を持っている必要があります。

Gas_limit：スマートコントラクトの発行に使いたいWEIの残高です。

スマートコントラクトを実行（スマートコントラクトを公開）した場合の出力例 スマートコントラクトをイーサリアムネットワークに入力します。実行されたトランザクション"creation_tx_hash"と、作成されたスマートコントラクトの割り当てアドレス"address"を表示します。

```
[
{
  "名前": "グリーター",
  "solidity": ".contract mortal { \n    /* Define variable owner of the type\n     address */\n    address owner;\n    \n    /* this function is executed at\n     initialization and sets the owner of the contract */\n    function mortal() { owner = msg.sender; } }\n    \n    /* 業務上の資本回収に対する制限 */\n    function kill() { if (msg.sender == owner) suicide(owner); } } \n\n\n  著の\n  greeter is mortal { \n    \n    /* define variable greeting of the type string */\n    string greeting;\n    \n    /* this runs when the contract is executed */\n    function greeter(string _greeting) public { \n        \n        greeting = _greeting;\n    }\n    \n    /* main function */\n    function greet()\n        constant returns (string) { \n            \n            return greeting;\n        }\n    }\n  }\n\n\n  \"bin\": \"606060405260405161023e38038061023e83398101604052805101600080546001\n60a060020a031916331790558060016000509080519060200190828054600181600116156\n101000203166002900490600052602060002090601f016020900481019282601f10609f57\n805160ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565\nb50505061016e806100d06000396000f35b828001600101855582156076579182015b8281\n1115607657825182600050559160200191906001019060b0565b509056606060405260e06\n0020a600035046341c0e1b58114610026578063cf3ae321714610068575b005b6100246000\n543373ffffffffffffffffff908116911614156101375760005\n473fffffffffffff16ff5b6100c9600060609081526001\n805460a06020601f600260001961010086881615020190941693909304928301819004028\n1016040526080828152929190828280156101645780601f10610139576101008083540402\n83529160200191610164565b6040518080602001828103825283818151815260200191508\n0519060200190808383829060006004602084601f0104600f02600301f150905090810190\n601f1680156101295780820380516001836020036101000a031916815260200191505b509\n25050506\n  \"アビ\" : [\n    {\n      \"constant\": false,\n      \"入力\" : [],\n      \"名前\": \"殺す\",\n      \"出力\" : [],\n      \"type\": \"function\"\n    },\n    {\n      \"constant\": true,\n      \"入力\" : []\n    }\n  ]
}
```

```

    "名前": "摸索",
    "出力": [...,
    {
        "名前": "",
        "type": "string"
    },
    ],
    "type": "function"
},
{
    "input": [[ 入力
    {
        "名前": "_greeting".
        "type": "string"
    }
],
    "タイプ": "ビレダ"
}
],
"gas_limit": 500,000.
"creation_tx_hash":
"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
"address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",
"params": [...,
    "ハローテスト"
],
},
{
    "名前": "死すべき者",
    "solidity": ".contract mortal {\n/* Define variable owner of the type\naddress */\naddress owner;\n/* this function is executed at\ninitialization and sets the owner of the contract */\nn function\nmortal() { owner = msg.sender; } }\nn /* 約束の資金回収に対する数 */\nfunction kill() { if (msg.sender == owner) suicide(owner); }\n}\non 約束\ngreeter is mortal {\n    n /* define variable greeting of the type string */\n    n string greeting.\n        * this runs when the contract is executed *\nfunction greeter(string _greeting) public {\n    greeting = _greeting;\n    } *\n        main function */\n        function greet()\nconstant returns (string) { \n            n return greeting;\n    } }"
},
"bin": ".606060405260008054600160a060020a03191633179055605c8060226000396000\nf3606060405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffff\nffffffffffffffffffffffffff90811691161415605a5760005473ffffffff\nffffffffffffffffffff16ff5b56",
"アビ": [
{
    "constant": false,
    "入力": [],
    "名前": "摸索",
    "出力": [],
    "type": "function"
},
{
    "入力": [],
    "タイプ": "ビレダ"
}
]

```

```
],
  "gas_limit": 500,000.
"params":["ハローテスト"]
}
]
```

ファイルを作成するテストブロック - (createTestingFile)

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

入力パラメータ : pathTestFile<String>

出力パラメータ : 参照されたパスにテストファイルが作成されます。

説明します。これは、ブロック(SmartContractCreateTokenERC20v1)やブロック(SmartContractCreateTokenERC20v2)を使用する際に、有効な一時ファイル作成パスが正しいことを確認するのに役立ちます。

ガス料金の料金を取得するためのブロック - (eth_RatesGasStationInfo)。

```
call BlockoinETHEREUM1 .eth_RatesGasStationInfo
```

入力パラメータ : nameFileSolidityJSON<String>

出力パラメータ : 参照しているスマートコントラクトのプロパティを表示します。イベント (OutputEth_GasStationInfo) では、配信された値はGWEIで与えられます。

ガス価格は、Ethereumのネットワーク内で取引を実行するシステムに支払われる値で、これらのシステムは一般的に「採掘者」と呼ばれており、ガス価格の値は、Ethereumのネットワーク内で取引がどれだけ速く（時間と優先度）実行されるかの関数となっています。

納品された値は、以下の実行時間に基づいています。これらの時間はおおよそのものであり、Ethereumネットワーク上での需要（取引）をどのように行っているかによって異なる場合があります。

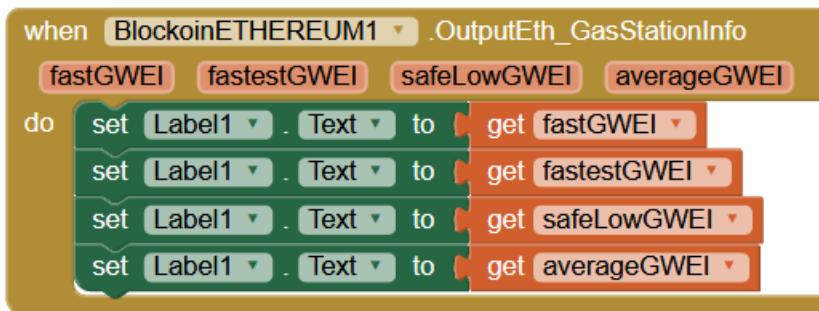
速い < 2分。

最速 < 30秒。

セーフロー < 30分。

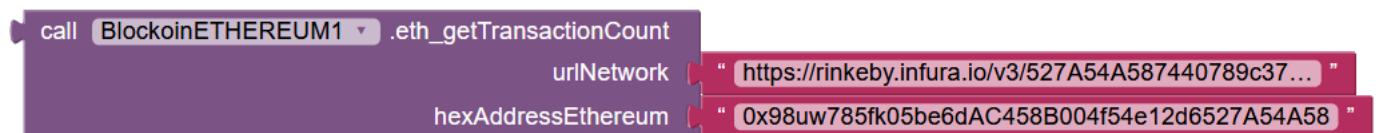
平均 15分未満。

取引所 Ethereum Extension (EEE) で行われる取引は、常に「ガス価格 = 平均」を使用しています。



説明: 新しいトランザクションを作成するためのクエリの時点で更新されたガス価格を取得します。

番号「nonce」を取得するためのブロック - (eth_getTransactionCount)。



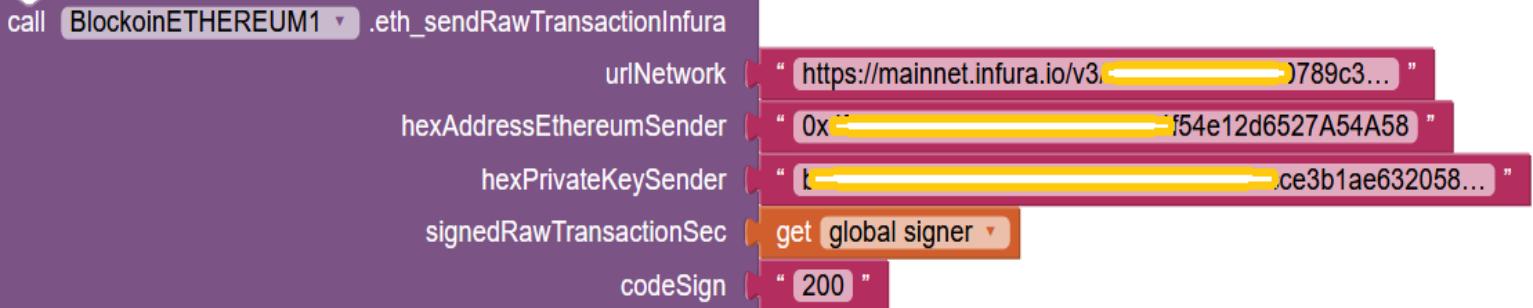
入力パラメータ : urlNetwork<String>, hexAddressEthereum<String>。

出力パラメータ : 参照されたアドレスの連続番号「nonce」を16進数で表示します。

「nonce」番号は、特定のアドレスから行われたトランザクションの数を記録する増分番号です。

説明: 参照されているアドレスの nonce 番号を取得します。

署名付きトランザクションを送信するブロック - (eth_SendRawTransactionInfura)。

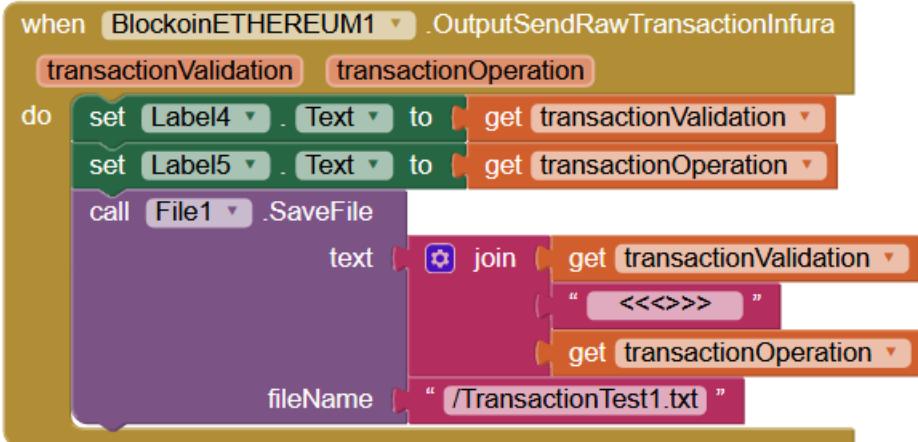


必要な依存関係: ブロック(eth_getTransactionCount)、ブロック
(SignerGenericPushRawTransactionOffline)。ブロック
(SignerGenericPushRawTransactionOffline)の例を確認します。

入力パラメータ: urlNetwork <String>、hexAddressEthereumSender <String>、
hexPrivateKeySender <String>、SignedRawTrasactionSec <String>、codeSign <String>。

出力パラメータ：イベント(OutputSendRawTransactionInfura)

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



説明します。トランザクションの結果、16進数の2つの値を配信します。transactionValidation値は、Ethereumネットワーク上で行われたトランザクションで、Ethereumの暗黙のコストが含まれています。transactionOperationの値は、Coinsolidation.orgネットワークで行われたトランザクションで、トランザクション時のエーテルの値に基づいて、トランザクションごとに0.5セントUSDのコストが

かかります。このコストは、Coinsolidation.orgネットワークのサービスの支払いのためのものであり、世界中の暗号と資産分野のための拡張機能のメンテナンス、サポート、作成に投資されています。

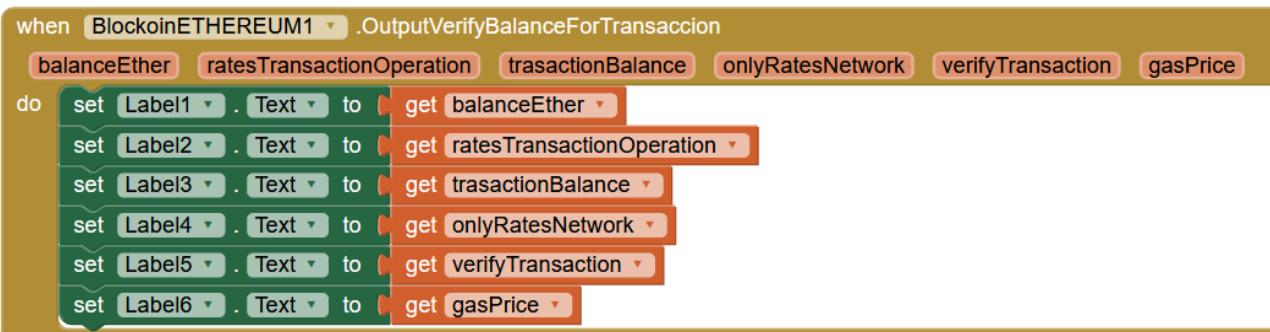
標準トランザクションのコストを計算するブロック - (eth_VerifiBalanceForTransaction)



入力パラメータ : addressEthereumSender<String>、valueEthertoSend<String>。

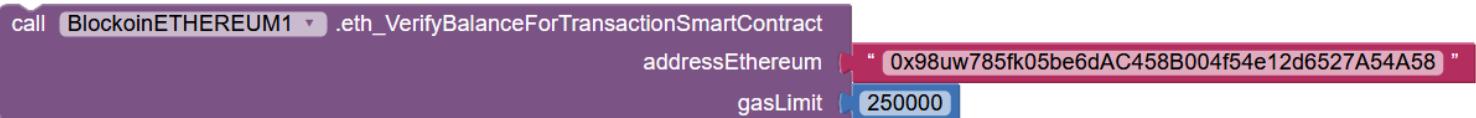
出力パラメータ : Event (OutputVerifyBalanceForTransaction)。

出力: balanceEther<String>, ratesTransactionOperation<String>, transactionBalance<String>, OnlyRatesNetwork<String>, verifyTransaction<String>, gasPrice<String>。



説明します。標準的な取引の費用がどのようなものになるのか、エントリーの住所を参照して詳細を提供します。出力パラメータ"verifyTransaction"は、トランザクションが"True"にすることができるか、または参照されるアドレスが十分なバランスを持っていない場合は、私たちに"False"を与えるかどうかを教えてくれます。

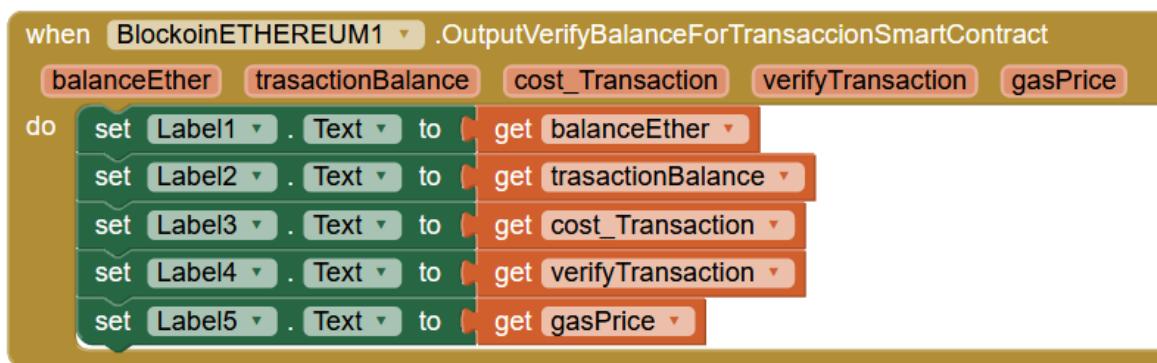
標準トランザクションのコストを計算するブロック
(eth_VerifiBalanceForTransaccionSmartContract)



入力パラメータ：addressEthereum<String>、gasLimit<String>。

出力パラメータ：イベント(OutputVerifyBalanceForTransaccionSmartContract)

出力：balanceEther<String>，trasactionBalance<String>，cost_Transaction<String>，verifyTransaction<String>，gasPrice<String>。



説明します。スマート契約の郵送にかかる標準的な取引のおおよその費用がどのようになるか、エントリーの住所を参照して詳細を提供しています。出力パラメータ"verifyTransaction"は、トランザクションが"True"にすることができるか、または参照されるアドレスが十分なバランスを持っていない場合は、私たちに"False"を与えるかどうかを教えてくれます。

balanceEther: 参照されるアドレスのバランスをエーテルで配信します。

trasactionBalance : 取引が行われた後の残高。

cost_Transaction: スマートコントラクトを公開するためのトランザクションのコストです。

verifyTransaction: (balanceEther から cost_Transaction を差し引いたもの)を使用しています。

gasPrice: "Miners"で使用されるGasPriceの現在の値、これは毎分変化することができます。

指定した国の通貨のイーサ価格を取得するブロック - (eth_getExchangeRates)

```
call BlockoinETHEREUM1 .eth_getExchangeRates  
countryRates "USD"
```

入力パラメータ: countryRates<String>.出力パラメータ「国」をチェックすると、すべての国の通貨タイプが含まれているので、希望する通貨を選択することができます。

出力パラメータ：イベント（OutputEth_getExchangeRates）。

出力: rates<String>, countries<String> 世界の国のすべてのレートをJSON形式で出力します。

```
when BlockoinETHEREUM1 .OutputEth_getExchangeRates  
rates country  
do set Label1 . Text to get rates  
set Label2 . Text to get country
```

説明します。イーサの現在の価格を、参照国の通貨の為替レートで表示します。

最小限の入力パラメータで標準トランザクションを実行するためのブロック
(eth_sendTransactionEasy)。

```
call BlockoinETHEREUM1 .eth_sendTransactionEasy  
hexPrivateKeySender "71a8D62d90b9f04590b4a120a8975cce3b1ae632058b0301..."  
toAddress "0xe3b1ae632058b0306071a8D62d90b392602a36"  
valueEther "0.05"
```

入力パラメータ: hexPrivateKeySender<String>, toAddress<String>, valueEther<String>。

出力パラメータ：イベント（OutputSendTransactionEasy

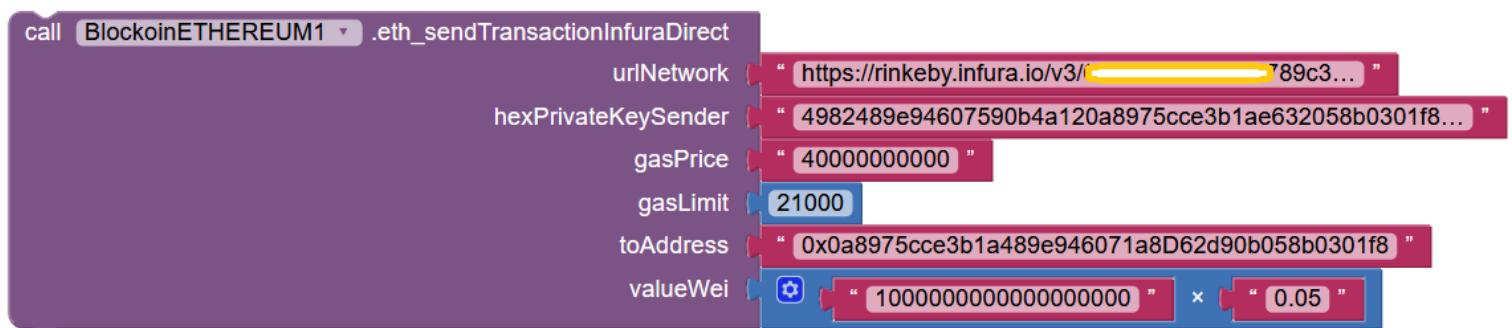
出力: transactionValidation<String>, transactionOperation<String>.

```
when BlockoinETHEREUM1 .OutputSendTransactionEasy  
transactionValidationE transactionOperationE  
do set Label1 . Text to get transactionValidationE  
set Label2 . Text to get transactionOperationE
```

説明: Ethereumのネットワークで標準的な取引を行うための機能で、この機能はINFURAのアカウントを持っている必要はなく、3つのパラメータを入力するだけで、目的の取引を行うのに十分なバランスを持っている必要があります。

取引は、公式のEthereum Web3jライブラリと当社のネットワークCoinsolidation.orgを使用して直接Ethereumネットワーク上に配置されます。

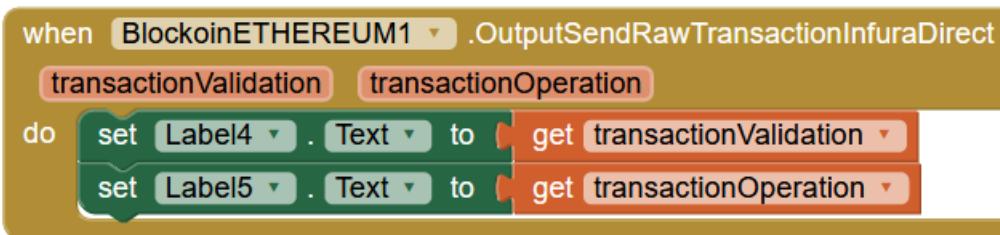
最小限の入力パラメータで標準トランザクションを実行するためのブロック
(eth_sendTransactionInfuraDirect)。



入力パラメータ: urlNetwork<String>、hexPrivateKeySender<String>、gasPrice<String>、
gasLimit<String>、toAddress<String>、valueWEI<String>。

出力パラメータ：イベント(OutputSendTransactionInfuraDirect)

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



説明: 既に暗黙の電子署名を含む標準トランザクションを送信する機能は、これはすでにトランザクションのコンポーネントの前の知識を持っており、彼らのニーズに応じて、これらのパラメータを最適化したい人のために有用です。

11. 標準取引コストの計算とスマートコントラクト取引

標準的なトランザクションの計算には、Ethereumネットワークでは3つのパラメータが必要です。

1. - ガス料金
2. - ガスの限界。
3. イーサ（Ethereumのデジタル通貨）の現在値。

GasPrice : これは通常GWEI (GigaWEI) の単位で与えられます。これは、1エーテルが1,000,000,000,000,000,000weiであれば、1GWEIは1,000,000,000,000,000に相当します。このユニットは、Ethereum ネットワークのすべてのトランザクションを実行するシステムの支払いとして機能し、「採掘者」と呼ばれる、それは世界中に分散されています。GasPriceは固定値ではなく、変動値であり、1分毎、1秒毎に変化します。GasPriceの価値を定義する人は「採掘者」であり、Ethereumネットワークがどれだけ飽和しているかにかかっています。

GasLimit: この値は通常、WEIの単位で与えられ、標準的なトランザクションでは平均的なデフォルト値は21,000 WEIですが、それはあなたがしたいトランザクションの種類に応じて高くても低くてもよいのですが、標準的なトランザクションでは、私たちは40,000 WEIの値を使用して、Gas Limit の下で持っていることの拒否を持っていないことを保証します。

イーサの値 : この値はまた、変数であり、グローバルな金融市場の異なるパラメータに起因するものであり、この値は、常に世界中のイーサの値を更新しているエンティティから取得することができます中央および分散型取引所と呼ばれる。

重要な注意事項。取引所のEthereum拡張（EEE）では、我々はより高いガスの制限（40,000）を使用しています。これは、"鉱夫"によって確立された最小のクオータに達していない場合

には、トランザクションが影響を受けていないという事実に起因していますが、Ethereumネットワークは、それがそれを作ることなく、トランザクションの計算で行われたことを費用を請求される場合。このため、ユーザーの中には自分のトランザクションが実行されない理由を説明しない人もいますが、彼らはトランザクションのリクエストが開始されたときに提供された金額またはすべてのGasLimitを請求されることになります。

GasPriceはeth_GetRatesGasStation関数を用いて参照することができ、標準取引のGasLimitは21,000WEI以上、スマートコントラクトを発行・実行する取引は最低50万WEI以上でなければなりません。

標準的な取引コスト。

次の式で定義されています。

コスト = (GasLimit × (GasPrice ÷ (1,000,000,000,000,000)) × エーテル値)。

例。

以下の値を想定してください。

ガスリミット = 40,000、ガス価格 = 45 GWEI、エーテル値 = 406 ドル。

コスト = (40,000 × (45,000,000,000 ÷ (1,000,000,000,000)) × 406)

標準取引コスト = 0.0018エーテル × 406 米ドル = 0.73 米ドル

スマートコントラクト取引の場合、コストは「鉱夫」がスマートコントラクトを処理するために使う作業量に正比例するため、つまり「鉱夫」が扱うコンピュータシステムでの処理量が単純になるため、どのようなスマートコントラクトが発行されるのかを把握しておく必要がある。

スマート契約の場合、デフォルト値は50万WEIの値でGasLimitを開始することになります。

留意すべき重要な点は、ある人がGasLimitを提案するとき、「鉱夫」は必ずしも提案された金額のすべてを取るとは限らないということである。つまり、ある人が取引を送信するとき、「鉱夫」は計算の労力を計算し、GasLimitから何が取り出されるかを計算し、それは場合によっては提供されるデフォルトのGasLimitである21,000 WEIより小さいか、または等しいかもしれない。

すべての取引は、各取引の詳細を相談できるサイト www.etherscan.io で相談できるようになります。

標準的なトランザクションの例では、トランザクションが40,000 WEIsのガスリミットで送信されたが、「鉱夫」は、処理コストがどれくらいになるかを計算する際に、52.5%、すなわち21,000 WEIsであるデフォルト値だけを取った。

The screenshot shows a transaction details page on Etherscan. The transaction hash is 0x7dae251f21c616fb04a94112633c97e04d11c91f4d06dd9b85ed11112f02703a. It was successful and has 2 block confirmations. The value sent was 0.001084598698481562 Ether (\$0.50). The transaction fee was 0.000672 Ether (\$0.31). The gas price was 0.000000032 Ether (32 Gwei) and the gas limit was 40,000. The gas used was 21,000 (52.5%).

**トランザクションオペレーション
キナイトラハナゲクシコノバリテ**

イーサでの取引コスト。

提率されたガスの限界：40,000

取引で使用した実際のガス限度額：21,000WEI

Overview	Status	Comments
② Transaction Hash:	0x7dae251f21c616fb04a94112633c97e04d11c91f4d06dd9b85ed11112f02703a	🔗
② Status:	Success	
② Block:	11234630	2 Block Confirmations
② Timestamp:	52 secs ago (Nov-11-2020 06:17:24 AM +UTC)	Confirmed within 38 secs
② From:	0x4b7355fd05be6dac458b004f54e12d6527a54a58	🔗
② To:	0x5d2acdb34c279aa6d1e94a77f7b18ab938bf2bb	🔗
② Value:	0.001084598698481562 Ether (\$0.50)	
② Transaction Fee:	0.000672 Ether (\$0.31)	
② Gas Price:	0.000000032 Ether (32 Gwei)	
② Gas Limit:	40,000	
② Gas Used by Transaction:	21,000 (52.5%)	
② Nonce	Position	

Eth: \$460.89 (+2.91%) | 34 Gwei

All Filters | Search by Address / Txn Hash / Block / Token / Ens | Home | Blockchain | Tokens | Resources | More | Sign In | Buy | Exchange | Earn | Crypto Credit

Transaction Details | Buy | Exchange | Earn | Crypto Credit

Featured: BeaconScan Guild Warz Set up ETH 2.0 Medalla Testnet validators and win more than \$1,000 in rewards! [Create Guild](#).

スマート契約の取引に戻り、50万のWEI GasLimitを取ることで、全部使った場合のコストは以下のようになります。

スマートな契約取引コスト。

次の式で定義されています。

コスト = (GasLimit × (GasPrice ÷ (1,000,000,000,000,000)) × エーテル値。

例。

以下の値を想定してください。

ガスリミット = 50万、ガス価格 = 45GWEI、エーテル値 = 406ドル。

コスト = (50万 × (45,000,000,000 ÷ (1,000,000,000,000)) × 406

取引コスト公開 スマート契約 = 0.0225エーテル × 406ドル = 9,135ドル

すべての取引は、サイト www.etherscan.io でご相談いただけます。

注：トランザクションの合計コストを取得するには、それらを合計する必要があります。

総取引コスト = Ethereumネットワークコスト + Coinsolidation.orgの手数料

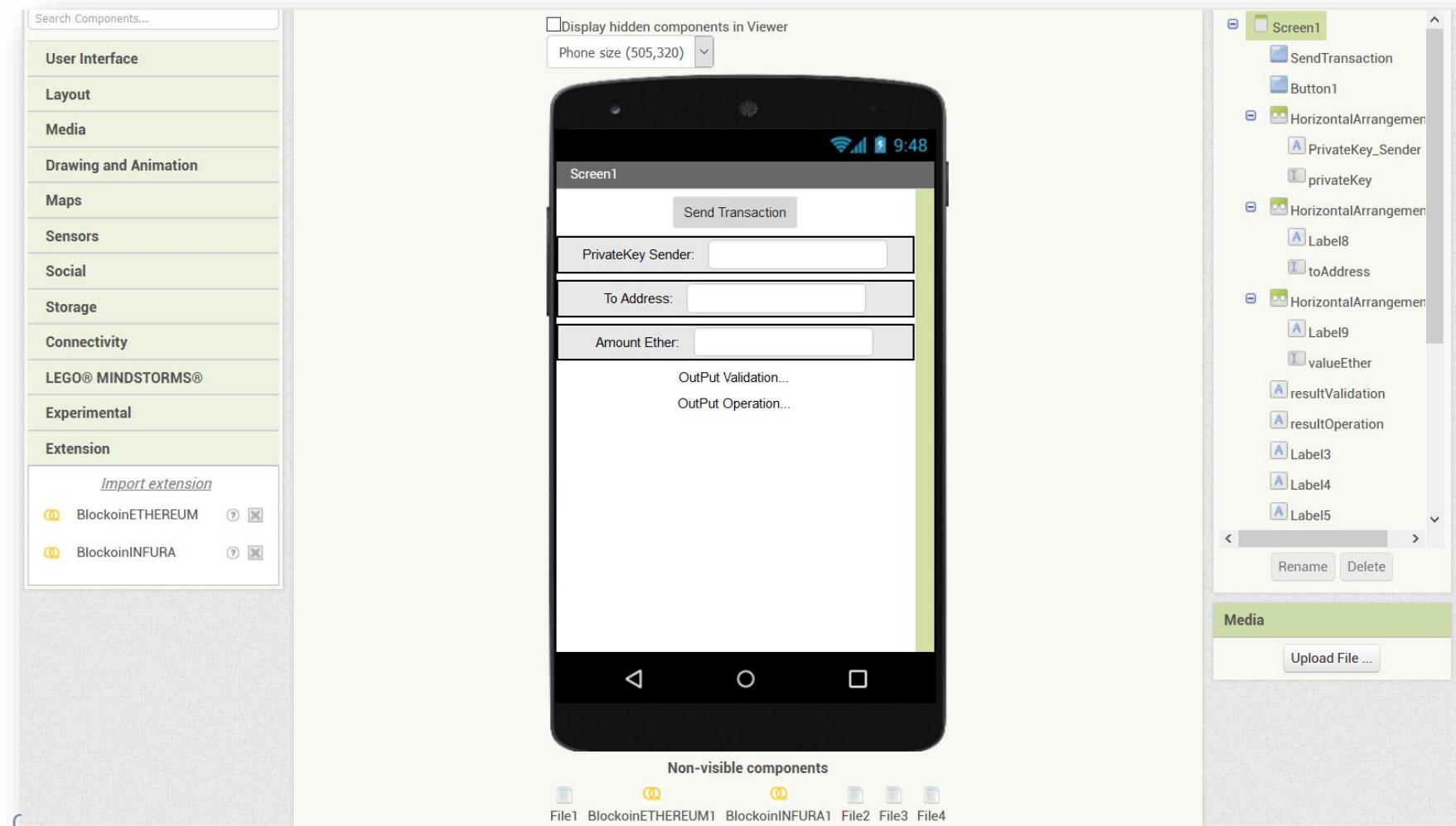
12. Coinsolidation.org 料金

標準的なトランザクション：0.5セント米ドル+Ethereumネットワークのコスト。

トランザクションは、公開および/またはスマート契約を実行する：15ドル+Ethereumネットワークのコスト。

13.15分でAndroidアプリ（Exchange）を作成します。

App Inventorでのデザイン（画面）。- 5分です。



関数ブロック(eth_SendTransactionEasy)とイベント(OutPutSendTransactionEasy)- 5分

The image shows a Scratch script consisting of two main parts:

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

```
when [SendTransaction v].Click
do
  call [BlockchainETHEREUM1 v].eth_sendTransactionEasy
    hexPrivateKeySender [privateKey v].Text
    toAddress [toAddress v].Text
    valueEther [valueEther v].Text
```

Bottom Part (when [BlockchainETHEREUM1 v].OutputSendTransactionEasy):

```
when [BlockchainETHEREUM1 v].OutputSendTransactionEasy
  transactionValidationE transactionOperationE
do
  set [resultValidation v].Text to (get [transactionValidationE v])
  call [File1 v].SaveFile
    text (get [transactionValidationE v])
    fileName "/trasactionValidation.txt"
  set [resultOperation v].Text to (get [transactionOperationE v])
  call [File1 v].SaveFile
    text (get [transactionOperationE v])
    fileName "/trasactionOperation.txt"
```

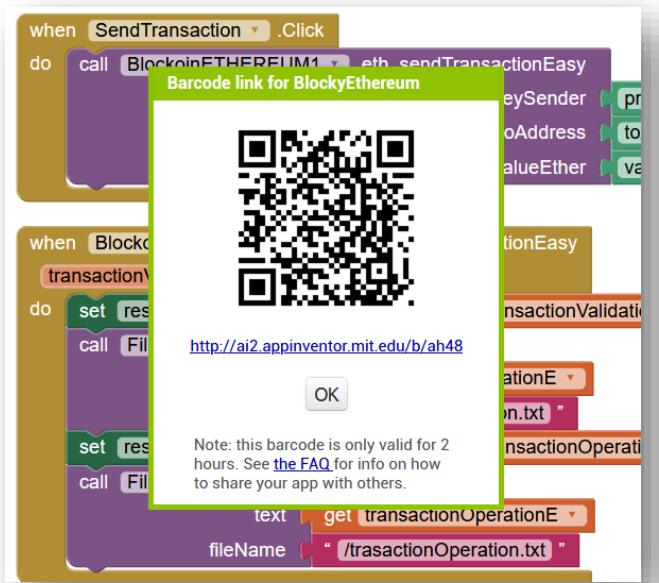
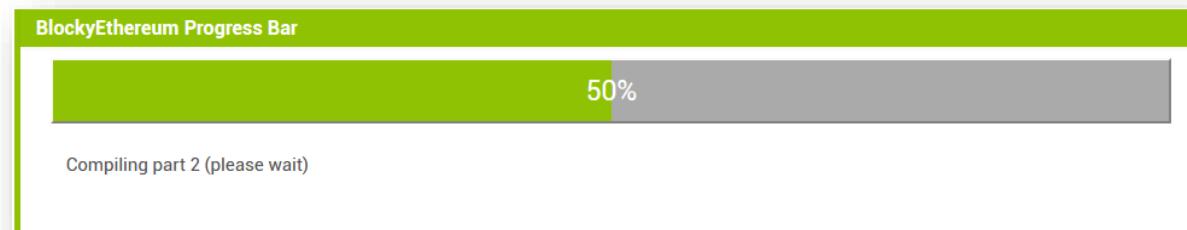
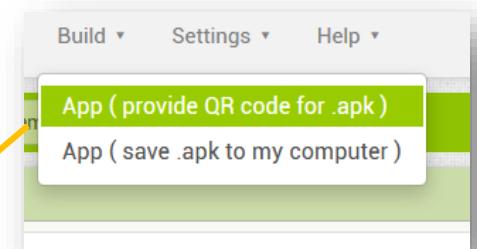
Annotations for the top part:

- 私有鍵を入力します。
- PrivateKey: 送信者のアドレスの主キー。
- toAddress: 受信機の 16 進数アドレス。
- valueEther: 送信するエーテルの量を指定します。

Annotations for the bottom part:

- 結果をテキストファイルに保存します。
関数 File1: ファイル **trasactionValidation.txt**
- 結果をテキストファイルに保存します。
File2 関数です。ファイル **trasactionValidation.txt**

コンパイルしてAPKファイルを生成し、Android端末にインストールします。- 5分



注: トランザクションが実行されると、"トランザクションを送信"ボタンが解放されるまでに約6~8秒かかります。Ethereumネットワークとの接続時間の関係で

14. トークンコインソリダクション。

トークン・コインソリデーションは、大きく分けて3つの指針を持ったプロジェクトです。

あなたのビジネスを成長させるためにあなた自身のトークン暗号通貨を持っていると想像してください。 Coinsolidationを使用すると、簡単かつシンプルにできます。

最初は、その簡単で直感的な使用によってプログラミングの前知識がなくても誰でも使用することができます視覚的なプログラミング方法論Blocklyに基づいて拡張機能の最初のネットワークを作成することができますが、私たちのロードマップ（白書）に相談することができます拡張機能は、世界経済の2つの基本的なセクターに向かっている、暗号通貨と/またはトークンと通貨のセクター（不換紙）または一般的な使用の世界的なドルドルUSD、ユーロEU、ポンドまたは使用中の他の通貨などのセクターです。

CoinSolidationプロジェクトの第2のガイドラインは、現在と将来の暗号通貨のアドレスを統合するための新しいアルゴリズムを作成することです。 私たちは、異なるブロックチェーンの異なるアドレスをユニバーサルアドレスに統合するアドレスを作成するための新しいモデルアルゴリズムを開発しています。

第三のガイドラインは、コインソリデーション環境のセキュリティに量子コンピューティング技術を適用することであり、これはすでに開発されているQRNG（量子乱数発生器）とPQC（ポスト量子コンピューティング）のセキュリティアルゴリズムの拡張機能と一緒に適用されます。 これらはGithubサイトの公式拡張機能リポジトリ（<https://github.com/coinsolidation>）にあります。

CryptoToken COINsolidationの一般的な機能

名称: CoinSolidation

シンボル : CUAG

タイプ : NFT

発売国 : エストニア

公式サイト : www.CoinSolidation.org

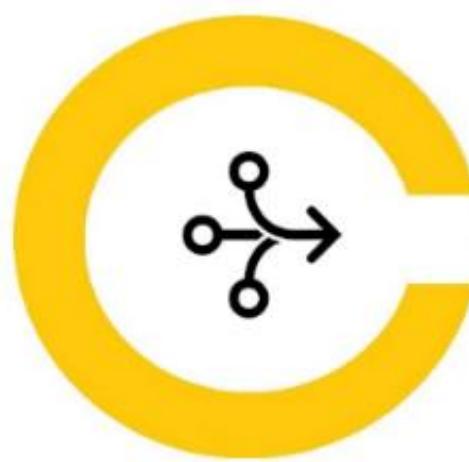
会社名 : コインズインターナショナル

発売日 : 2020年12月30日

作成者 : ルグサマヤ

コンセンサスアルゴリズム : 量子の証明

アドレスアルゴリズム。連結ユニバーサルアドレス。



15.ソフトウェアのライセンスと使用。

ライセンス、使用条件は www.coinsolidation.org を参照するか、info@coinsolidation.org までご連絡ください。