



EX^{change}_{tensions}

Configuration et administration.

Extension de l'échange d'éthereum.

Guide de l'utilisateur

version 1.0.0 Beta

Novembre 2020.

Blockoin.org est une marque déposée de Bankoin Inc, sous licence d'utilisation libre et commerciale. Conditions d'utilisation à l'adresse suivante : www.CoinSolidation.org

Contenu

1.	Introduction	3
2.	Qu'est-ce que la programmation en Blockly ?	5
3.	Qu'est-ce qu'une extension ?.....	5
4.	Qu'est-ce que le BlockoinEthereum et le BlockoinINFURA ?.....	5
5.	Concepts de base appliqués dans la plate-forme Ethereum.	6
6.	Installation et configuration de l'extension et de l'environnement de test Ethereum.	13
7.	Définition et utilisation des blocs (fonction générique).....	20
8.	Caractéristiques et événements d'Exchange Ethereum Extension (EEE).....	21
9.	Étapes pour créer un CryptoToken ou un Cryptomoney Token.	32
10.	Comment mettre en vente un nouvel actif ou votre jeton de crypte (Token ERC20).	33
11.	Calcul du coût de transaction standard et de la transaction contractuelle intelligente.....	56
12.	Frais de Coinsolidation.org.....	59
13.	Créez votre application Android (Exchange) en 15 minutes.....	60
14.	Token CoinSolidation.	63
15.	Licences et utilisation des logiciels.....	64

1. Introduction.

Ethereum est une plateforme open source, décentralisée contrairement aux autres chaînes de blocs, Ethereum peut faire beaucoup plus. Il est programmable, ce qui signifie que les développeurs peuvent l'utiliser pour créer de nouveaux types d'applications. Sa monnaie numérique est appelée "Ether".

Ces applications décentralisées (ou "dapps") bénéficient des avantages du montage cryptographique et de la technologie des chaînes de blocs. Ils sont fiables et prévisibles, ce qui signifie qu'une fois "chargés" dans l'Ethereum, ils fonctionneront toujours dans les délais prévus. Ils peuvent contrôler les biens numériques pour créer de nouveaux types d'applications financières. Ils peuvent être décentralisés, ce qui signifie qu'aucune entité ou personne ne les contrôle.

En ce moment, des milliers de développeurs dans le monde entier créent des applications sur Ethereum et inventent de nouveaux types d'applications, dont beaucoup peuvent être utilisées aujourd'hui :

- Portefeuilles en crypto-monnaie qui vous permettent d'effectuer des paiements instantanés et peu coûteux avec l'ETH ou d'autres actifs
- Les applications financières qui vous permettent d'emprunter, de prêter ou d'investir vos actifs numériques
- Des marchés décentralisés, permettant d'échanger des biens numériques, ou même d'échanger des "prédictions" sur des événements du monde réel.
- Des jeux où vous avez des atouts dans le jeu et pouvez même gagner de l'argent réel.

Comment l'éther fonctionne-t-il ?

L'éther, comme les autres crypto-monnaies, utilise un livre numérique partagé où toutes les transactions sont enregistrées. Il est accessible au public, totalement transparent et très difficile à modifier par la suite.

C'est ce qu'on appelle une **chaîne de blocs, qui se** construit au cours du processus **d'extraction**.

Les mineurs sont chargés de vérifier des groupes de transactions d'éther pour former des "blocs" et de les coder en résolvant des algorithmes complexes. Ces algorithmes peuvent être plus ou moins difficiles à mettre en œuvre afin de maintenir une certaine cohérence dans le temps de traitement des blocs (environ un toutes les 14 secondes).

Les nouveaux blocs sont ensuite reliés à la chaîne de blocs précédente et le mineur en question reçoit une **récompense, c'est-à-dire** un nombre fixe de *jetons d'éther*. Normalement, il s'agit de 5 unités d'éther, bien que ce nombre puisse être considéré comme variable si la monnaie de la crypte continue à augmenter.

Comment fonctionne Ethereum ?

La *chaîne de blocs* Ethereum est très similaire à celle de bitcoin, mais son langage de programmation permet aux développeurs de créer des logiciels permettant de gérer les transactions et d'automatiser certains résultats. Ce logiciel est connu sous le nom de **contrat intelligent**.

Si un contrat traditionnel décrit les termes d'une relation, un contrat intelligent s'assure que ces termes sont respectés en les écrivant en code. Il s'agit de programmes qui exécutent automatiquement le contrat une fois que les conditions prédéfinies sont remplies, ce qui élimine le retard et le coût qui existent lorsque l'on exécute un accord manuellement.

Pour donner un exemple simple, un utilisateur d'Ethereum pourrait créer un contrat intelligent pour envoyer une quantité déterminée d'éther à un ami à une certaine date. Ils inscrivent ce code dans la chaîne de caractères du bloc et lorsque le contrat est terminé (c'est-à-dire lorsque la date convenue est atteinte), l'éther est envoyé automatiquement.

Cette idée de base peut être appliquée à des configurations plus complexes, et son potentiel est probablement illimité, avec des projets qui ont déjà fait des progrès remarquables dans des secteurs tels que l'assurance, l'immobilier, les services financiers, les services juridiques et la micro-finance.

Les contrats intelligents présentent également plusieurs avantages supplémentaires :

1. Ils éliminent la figure de l'intermédiaire, offrant à l'utilisateur un contrôle total et minimisant les coûts supplémentaires
2. Ils sont enregistrés, cryptés et dupliqués dans la chaîne de blocs publics, où tous les utilisateurs peuvent voir l'activité du marché
3. Éliminer le temps et les efforts nécessaires dans les processus manuels

Bien sûr, les contrats intelligents sont encore un système très récent avec de nombreux détails à peaufiner. Le code est traduit littéralement, de sorte que toute erreur lors de la création du contrat pourrait entraîner des résultats non désirés qui ne peuvent être modifiés.

DApps vs. contrats intelligents

Les contrats intelligents présentent des similitudes avec les **DApps** (applications décentralisées), mais les séparent aussi de certaines différences importantes.

Comme les contrats intelligents, un DApp est une interface qui relie un utilisateur à un service d'un fournisseur par le biais d'un réseau de pairs décentralisé. Mais, alors que les contrats intelligents nécessitent la création d'un nombre fixe de participants, les DApps n'ont pas de limite au nombre d'utilisateurs. En outre, elles ne se limitent pas aux applications financières telles que les contrats intelligents : un DApp peut servir à toutes les fins auxquelles vous pensez.

2. Qu'est-ce que la programmation en blockly ?

Blockly est une méthodologie de **programmation visuelle** basée sur le langage de programmation JavaScript, qui consiste en un simple ensemble de commandes que nous pouvons combiner comme s'il s'agissait des pièces d'un puzzle. C'est un outil très utile pour ceux qui veulent **apprendre à programmer de** manière intuitive et simple ou pour ceux qui savent déjà programmer et qui veulent voir le potentiel de ce type de programmation.

Blockly est une forme de programmation où il n'est pas nécessaire d'avoir des connaissances dans un quelconque langage informatique, c'est parce qu'il s'agit simplement de joindre des blocs graphiques comme si nous jouions au lego ou à un puzzle, il suffit d'avoir un peu de logique et c'est tout !

Tout le monde peut créer des programmes pour les téléphones mobiles (smartphones) sans se frotter aux langages de programmation difficiles à comprendre, il suffit d'assembler des blocs de manière graphique, de façon simple, facile et rapide à créer.

3. Qu'est-ce qu'une extension ?

Une extension est un module réalisé dans un langage de programmation Java donné dans un fichier avec l'extension .AIX

Dans le projet Blockoin.org, nous nous sommes basés sur la création d'extensions conviviales pour le domaine financier avec lesquelles ils peuvent arriver à faire des applications mobiles en quelques minutes sans avoir une grande connaissance de la programmation.

4. Qu'est-ce que le BlockoinEthereum et le BlockoinINFURA ?

BlockoinEthereum et BlockoinINFURA sont des logiciels à usage libre (extensions) qui comprennent les solutions technologiques (algorithmes) suivantes pour pouvoir créer des être utilisés dans le réseau Ethereum :

- CRÉATION DE NOUVEAUX COMPTES SUR LA PLATEFORME ETHEREUM.
- CONSULTATION DU BILAN, TRANSFERTS D'ACTIFS (CRYPTOMONEDA-ETHER ET TOKENS)
- CRÉATION DE NOUVEAUX JETONS ERC20 ET CRYPTOMONEDA-TOKEN.
- LA COMPILEMENT, LA CRÉATION, LA CONSULTATION ET LA PUBLICATION DU CONTRAT INTELLIGENT
- LA CRÉATION DE TRANSACTIONS EN LIGNE ET HORS LIGNE.
- LE CHARGEMENT DES CLÉS PRIMAIRES ET PUBLIQUES.
- TAUX DE CHANGE ET CONSULTATION DES STATIONS D'ESSENCE.
- DÉVELOPPEMENT, TEST ET ETHEREUM DES ENVIRONNEMENTS DE RÉSEAUX CENTRAUX (RÉSEAUX)
- COMPREND LES 39 CARACTÉRISTIQUES D'INFURA.

5. Concepts de base appliqués dans la plate-forme Ethereum.

Qu'est-ce qu'une chaîne de blocage ?

La chaîne de blocs est généralement associée aux bitcoins et autres devises cryptées, mais ce ne sont que la partie visible de l'iceberg, car elle n'est pas seulement utilisée pour la monnaie numérique, mais peut être utilisée pour toute information pouvant avoir une valeur pour les utilisateurs et/ou les entreprises. Cette technologie, dont les origines remontent à 1991, lorsque Stuart Haber et W. Scott Stornetta ont décrit les premiers travaux sur une chaîne de blocs cryptographiquement sécurisés, n'a pas été remarquée avant 2008, date à laquelle elle est devenue populaire avec l'arrivée des bitcoins. Mais actuellement, son utilisation est demandée dans d'autres applications commerciales et devrait se développer à moyen terme sur plusieurs marchés, comme les institutions financières ou l'Internet des objets, entre autres secteurs.

La blockchain, mieux connue sous le terme de blockchain, est un enregistrement unique, convenu d'un commun accord, réparti sur plusieurs nœuds (appareils électroniques tels que les PC, les smartphones, les tablettes, etc. Dans le cas des crypto-monnaies, on peut considérer qu'il s'agit du livre comptable où chacune des transactions est enregistrée.

Son fonctionnement peut être complexe à comprendre si nous entrons dans les détails internes de sa mise en œuvre, mais l'idée de base est simple à suivre.

Il est stocké dans chaque bloc :

1.- un certain nombre d'enregistrements ou de transactions valables,

2.- des informations concernant ce bloc,

3.- son lien avec le bloc précédent et le bloc suivant grâce au hachage de chaque bloc –un code unique qui serait comme l'empreinte digitale du bloc.

Par conséquent, **chaque bloc a une place spécifique et immuable dans la chaîne**, car chaque bloc contient des informations provenant du hachage du bloc précédent. La chaîne entière est stockée sur chaque nœud du réseau qui constitue la chaîne de blocs, de sorte qu'**une copie exacte de la chaîne est stockée sur tous les participants du réseau**.

Qu'est-ce qu'une adresse ou un compte au sein de la plateforme Ethereum ?

Il s'agit d'une chaîne de 42 caractères dans la plateforme Ethereum qui représente un nombre en base hexadécimale, où les avoirs définis dans l'Ethereum seront déposés ou envoyés. Dans d'autres plateformes de chaînes de blocs, le nombre de caractères du compte ou de l'adresse peut être différent, par exemple :

0x5d2Acdb34c279Aa6d1e94a77F7b18aB938BFb2bB

Qu'est-ce qu'une kryptomonie ?

Il s'agit d'une monnaie numérique ou virtuelle conçue pour fonctionner comme un moyen d'échange. Elle utilise la cryptographie (sécurité numérique) pour sécuriser et vérifier les transactions, ainsi que pour contrôler la création de nouvelles unités d'une cryptomoney particulière.

Qu'est-ce que l'éther ?

L'éther est la monnaie numérique native de la plateforme Ethereum, une plateforme décentralisée développée en 2013. Un éther est une unité qui peut être divisée en unités plus petites appelées WEI et qui a l'équivalence suivante.

1 Ether = 1.000.000.000.000.000 Wei. (En termes mathématiques, il s'agit de 10^{18}).

Quelles sont les unités manipulées lors de l'utilisation d'un éther ?

1	ether
10^3	finney
10^6	szabo
10^9	Shannon ou GWEI : il est utilisé pour le prix de l'essence.
10^{12}	babbage
10^{15}	lovelace
10^{18}	wei

Qu'est-ce qu'un jeton ?

Les jetons sont des actifs numériques qui peuvent être utilisés dans le cadre d'un écosystème de projet donné.

La principale distinction entre les jetons et les crypto-monnaies est que les premiers nécessitent une autre plate-forme de chaîne de blocs (pas la leur) pour fonctionner. Ethereum est la plateforme la plus courante pour la création de jetons, principalement en raison de sa fonction de contrat intelligent. Les jetons créés sur la chaîne de blocs Ethereum sont généralement connus sous le nom de jetons ERC-20, bien qu'il existe d'autres types de

jetons plus spécialisés comme le jeton ERC-721 utilisé principalement pour les biens de collection (cartes, utilisation dans les jeux vidéo, œuvres d'art, etc.).

Qu'est-ce que le gaz ?

Le gaz est le coût de l'exécution d'une opération ou d'un ensemble d'opérations sur le réseau Ethereum. Ces opérations peuvent être multiples : de la réalisation d'une transaction à l'exécution d'un contrat intelligent ou à la création d'une application décentralisée.

En d'autres termes, plus simplement, le gaz est l'unité de mesure du travail effectué dans l'Ethereum.

Comme dans le monde physique, il y a aussi dans Ethereum des emplois qui coûtent plus cher que d'autres : si l'opération que nous voulons réaliser nécessite une plus grande utilisation des ressources par les nœuds qui forment la plate-forme, cela fera augmenter le gaz aussi et vice versa.

Le gaz sert principalement à remplir trois fonctions sur la plate-forme Ethereum :

1.- Attribue un coût à l'exécution des tâches ; Dans Ethereum, l'exécution des tâches a également un coût. **Selon la difficulté de la tâche ou la vitesse à laquelle nous voulons que cette tâche soit traitée, le coût de calcul de cette opération sera plus ou moins élevé en conséquence et le nombre de gaz augmentera ou diminuera en proportion.**

2.- Sécurise le système ; Le système Ethereum est un système sûr et cela est largement possible grâce au gaz.

En exigeant le paiement d'une commission pour chaque transaction effectuée, la plate-forme garantit qu'aucune transaction inutilisable n'est traitée sur le réseau. Cela permet d'alléger la chaîne de blocage, car cela n'ajoutera pas beaucoup de mégaoctets d'informations inutiles à la chaîne de blocage.

En outre, avec le gaz, le système est également protégé contre le "spam" et l'utilisation infinie de boucles : des instructions pour effectuer des tâches répétitives par code.

Par exemple, si le gaz n'existe pas, rien n'empêcherait de répéter une tâche à l'infini, ce qui ferait s'effondrer le système et le rendrait inutilisable.

3.- Récompenser les mineurs ; Les mineurs sont des systèmes externes indépendants, répartis dans le monde entier, qui sont chargés d'exécuter les transactions au sein de la plate-forme Ethereum. Lorsque nous effectuons une transaction ou exécutons un contrat intelligent, nous "payons" une certaine quantité de gaz.

Ce gaz sert à "payer" les mineurs pour les ressources qu'ils ont utilisées (matériel, électricité et temps) et **ajoute également une récompense** pour leur travail. On peut donc dire que le gaz contribue également à maintenir l'équilibre de la plate-forme.

Nous parlons de "payer" le gaz - entre guillemets - parce que c'est ce que coûtent les opérations. En d'autres termes, vous "payez" les frais qu'Ethereum doit engager pour traiter les transactions.

Qu'est-ce que le prix de l'essence ?

Une unité de gaz correspond à l'exécution d'une instruction, telle qu'un pas d'ordinateur.

Alors comment les mineurs "encaissent-ils" le gaz qu'ils acquièrent comme récompense ?

Le gaz lui-même ne vaut rien, et ne peut donc pas être facturé. Pour "faire payer" ce gaz usagé, il est nécessaire de donner une valeur à ces ressources consommées, c'est-à-dire de donner une valeur monétaire au travail effectué par les mineurs. La quantité de gaz utilisée dans une transaction ou un contrat intelligent a un prix équivalent en éther. Ce prix est appelé "prix du gaz", il est normalement attribué par les mineurs et varie en fonction de l'activité de la chaîne de l'Ethereum. Il est normalement traité dans les unités GWEI.

Qu'est-ce que la limite de gaz ?

Ces données indiquent la valeur maximale de gaz qu'une transaction peut consommer pour être valable.

Normalement, le logiciel utilisé pour effectuer les transactions sur le réseau Ethereum calcule automatiquement une estimation de la quantité de gaz nécessaire pour effectuer la transaction et nous la montre immédiatement.

Qu'est-ce qu'une station-service ?

C'est l'endroit où les valeurs traitées par les mineurs sont référencées afin de décider par consensus quel est le prix du gaz nécessaire pour effectuer les différents types de transactions dans la chaîne du bloc Ethereum.

En fonction de la quantité de GasPrice sélectionnée, le temps de priorité accordé à l'exécution de la transaction sera pondéré. Normalement, trois types de GasPrice sont traités : TRADER : ASAP, FAST < 2 minutes, STANDARD < 5 minutes. Il s'agit de valeurs moyennes qui peuvent varier en fonction de la charge de travail (tractions) du réseau principal de l'Ethereum.

<https://ethgasstation.info/>

Qu'est-ce qu'un échange ?

Un échange de kryptomanes est le point de rencontre où les échanges de kryptomanes ont lieu en échange d'argent fictif ou d'autres kryptomanes. Dans ces maisons d'échange en ligne, le prix du marché est généré, ce qui marque la valeur des cryptomonies en fonction de l'offre et de la demande.

Qu'est-ce que le taux de change ?

Il s'agit des taux pour la valeur d'un éther dans les devises de chaque pays. Par exemple, le jour de la création de ce manuel, un éther a une valeur en dollars américains de 430,94

Qu'est-ce qu'un contrat intelligent ?

Dans ethereum, un contrat intelligent est un programme en langage de programmation appelé Solidity qui se trouve dans la chaîne de blocs Ethereum, qui a des instructions à exécuter automatiquement sur la base de règles préétablies. Cette propriété fait une évolution dans toutes les chaînes de blocs existantes car vous pouvez créer de nombreux contrats intelligents adaptés aux différents secteurs privés et publics, ce qui rend les opérations des entreprises plus efficaces ou, le cas échéant, les adapter à des systèmes ou des programmes de toutes sortes.

Qu'est-ce que le "nonce" ?

Il s'agit d'un compteur "hexadécimal" incrémentiel qui permet de suivre les transactions dans chaque sens ou compte créé dans Ethereum.

Qu'est-ce qu'une transaction ?

Il s'agit de l'exécution ou du transfert d'un certain type de bien non corporel auquel on peut attribuer une valeur préétablie dans le cadre du système Ethereum et qui peut ensuite être transformé en une valeur corporelle pour une entreprise ou une personne.

Qu'est-ce que txHash ?

Il s'agit d'un nombre hexadécimal qui permet de suivre le résultat en détail de chaque transaction.

Quels sont les types de transactions ?

Vous en avez deux types, l'un est la transaction "hors ligne" que cela crée sans qu'il soit nécessaire d'avoir une connexion au réseau principal d'Ethereum ; elle peut être stockée jusqu'à ce que vous choisissez de vous connecter au réseau d'Ethereum et de libérer la transaction, ce qui présente l'avantage de la sécurité car toute la transaction est traitée hors ligne ce qui évite toute anomalie qui pourrait se trouver dans la connexion au réseau. L'autre transaction est celle "en ligne" qui doit toujours être connectée à l'internet avec les avantages et les inconvénients que la connexion apporte en termes de sécurité.

Qu'est-ce que INFURA.io ?

Infura.io est une plateforme qui fournit un ensemble d'outils et d'infrastructures permettant aux développeurs de faire passer facilement leur chaîne de blocage d'applications du test à la mise à l'échelle, avec un accès simple et fiable à Ethereum et IPFS.

Qu'est-ce qu'une adresse sur le réseau Ethereum ?

Une adresse ou un compte est composé de trois parties, l'adresse, la clé publique et la clé privée, ces deux clés sont une chaîne de chiffres et de caractères au format hexadécimal qui sont utilisés pour envoyer et recevoir (actif) ou éther (monnaie numérique).

La clé principale ne doit jamais être partagée avec quiconque car c'est elle qui autorise la libération du solde (signe les transactions) détenu sur le compte.

La clé publique est connue de tout le monde et est partagée avec tout le monde car elle est la référence pour confirmer que la transaction est correcte à la fois en termes de valeur et de destinataire.

Exemples :

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

Qu'est-ce que Coinsolidation.org ?

C'est une plateforme de consolidation des crypto-monnaies, nous avons créé les premières adresses hybrides dans le monde financier centralisé et décentralisé, en utilisant la technologie Blockly qui est une forme de programmation visuelle sans avoir besoin de connaissances avancées en programmation basée sur des extensions fonctionnelles. Voir notre livre blanc à l'adresse www.coinsolidation.org

Quels types de réseaux de test et de réseau principal dans la chaîne d'approvisionnement Ethereum ?

<https://mainnet.infura.io>

Réseau principal, réseau de production où toutes les transactions sont effectuées en temps réel.

<https://ropsten.infura.io>

Le réseau de test Ropsten permet aux développements en chaîne de blocs de tester leur travail dans un environnement réel, mais sans avoir besoin de véritables jetons ETH, avec un algorithme appelé (*Proof-of-Work*).

<https://kovan.infura.io>

Le réseau de test Kovan, qui contrairement à son prédecesseur ropsten utilise un algorithme appelé Preuve d'autorité.

<https://rinkeby.infura.io>

Test Network, utilise un algorithme appelé Preuve d'autorité. L'un des plus utilisés pour les tests.

<https://goerli.infura.io>

Goerli est un réseau public de test pour Ethereum que le moteur de consensus POA (Proof of Authority) soutient auprès de divers clients. Antispam.

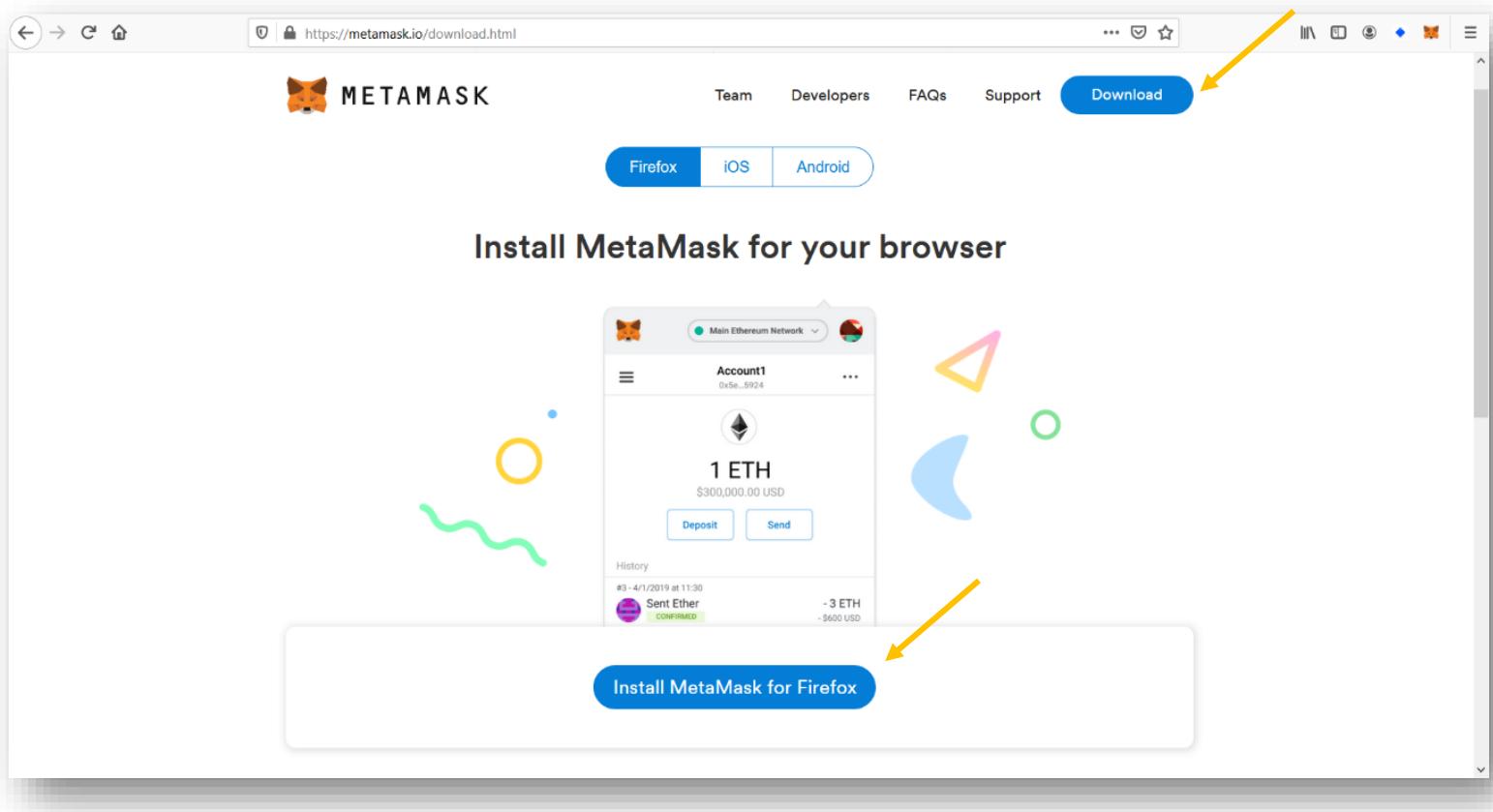
Au total, il existe cinq réseaux basés sur la chaîne de blocs Ethereum, un pour la production ou principal et quatre pour les tests, puis nous utiliserons le réseau de Rinkeby pour installer notre environnement de test.

6. Installation et configuration de l'extension et de l'environnement de test Ethereum.

Nous avons d'abord besoin d'un environnement de test. Pour apprendre à utiliser les différentes fonctionnalités des deux extensions qui seront utilisées pour créer, envoyer, publier, réviser et obtenir les données de toutes les transactions que nous effectuons sur la plateforme Ethereum.

La première chose que nous devons faire est de mettre en place notre environnement de test. Nous devrons installer l'application **METAMASK**, un portefeuille permettant de créer, d'importer des comptes Ethereum et de gérer les transactions à partir du navigateur de notre navigateur Internet, qui est Mozilla et peut également être utilisé dans Chrome.

Rendez-vous sur le site officiel [www.metamask.io](https://metamask.io) et cliquez sur le bouton "Télécharger".

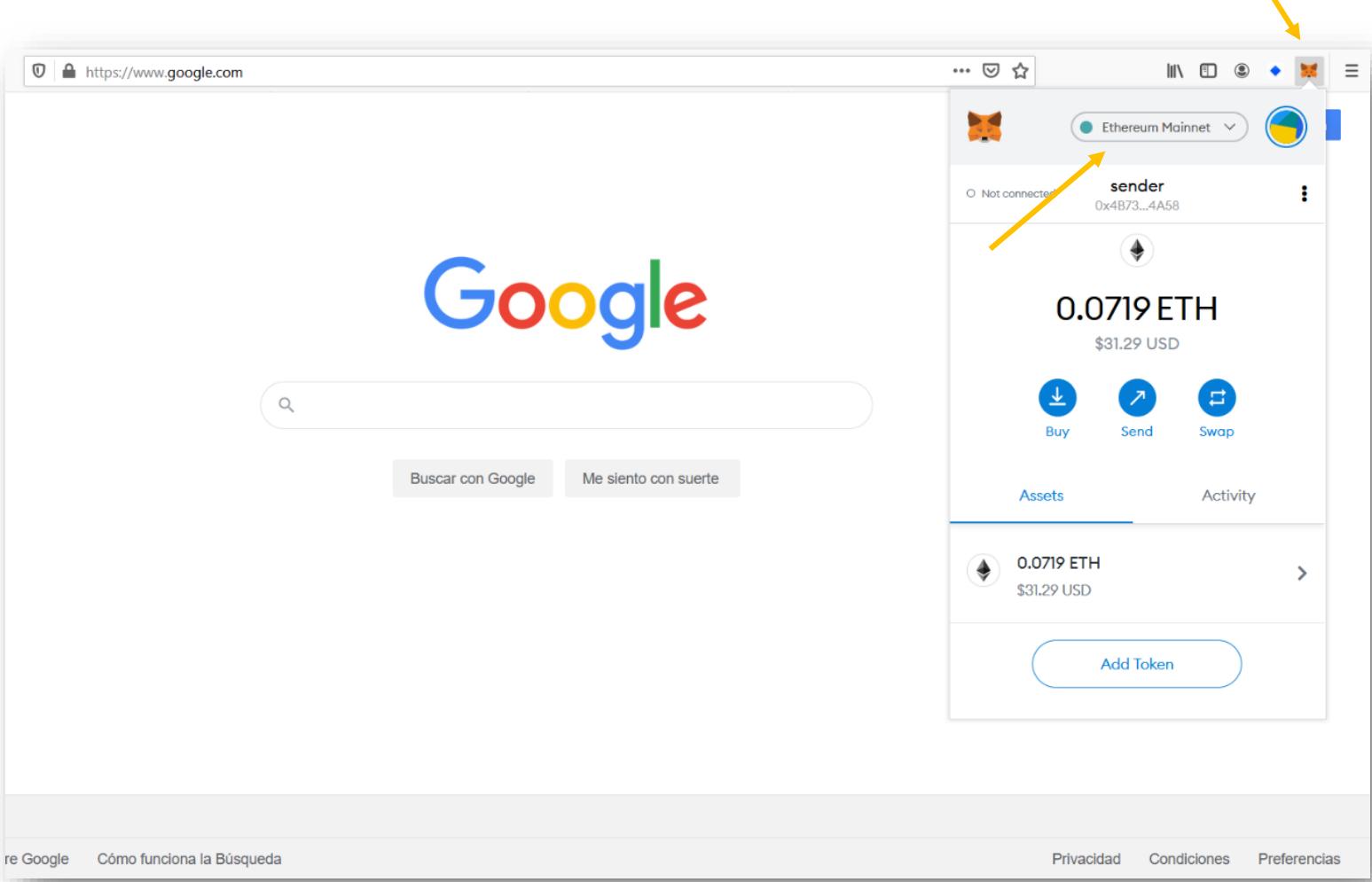


Cliquez ensuite sur le bouton "Installer MetaMask pour Firefox" et acceptez les options par défaut. Après l'installation, nous verrons une icône dans le coin supérieur droit où nous verrons le logiciel Metamask déjà installé.

Cliquez sur l'icône Metamask et l'option permettant d'importer un compte existant ou d'en créer un nouveau apparaîtra. Dans notre cas, nous importerons un compte que nous avons

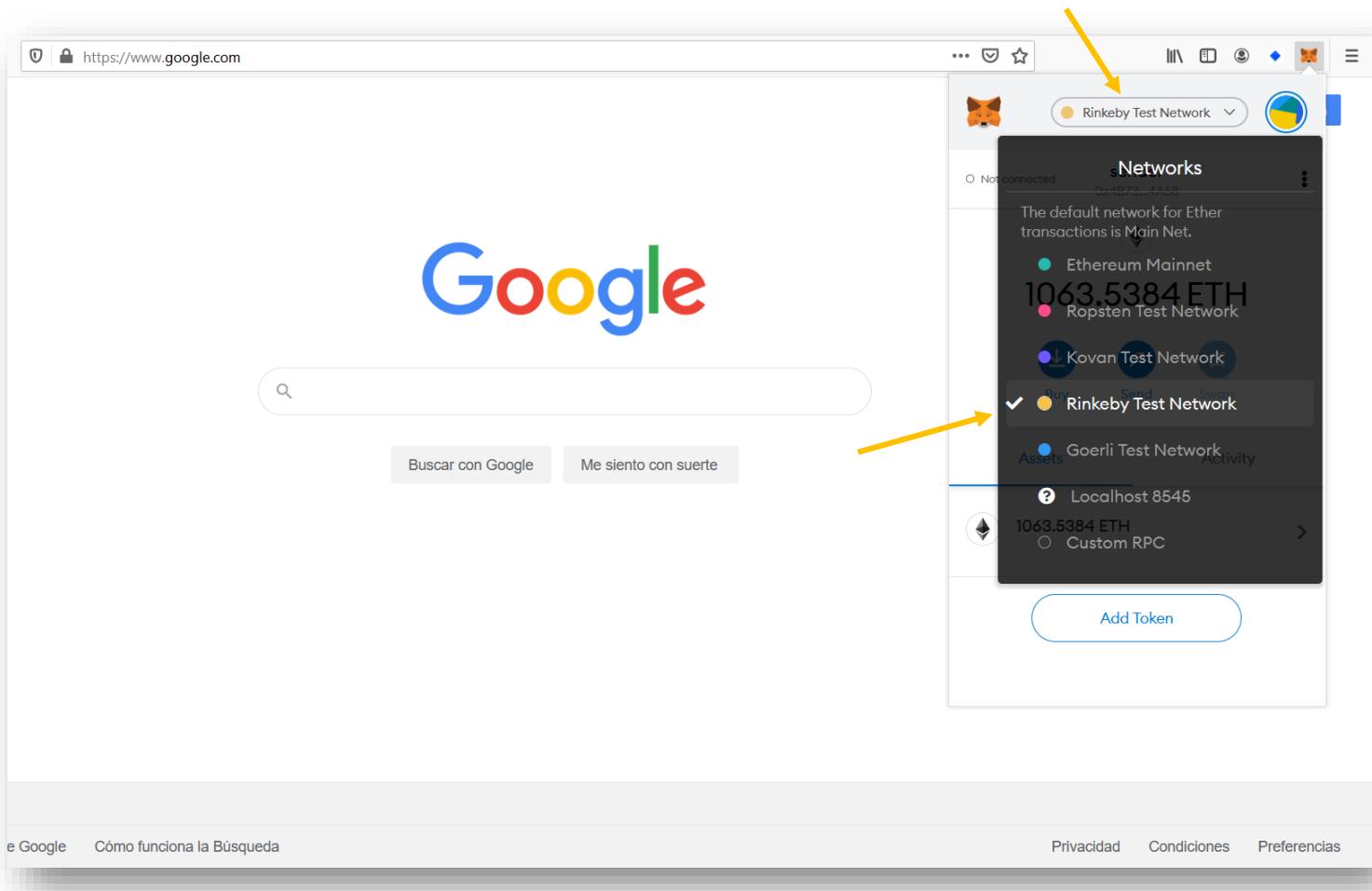
déjà en cours d'exécution en utilisant la méthode de "restauration par les semences". Si vous n'en avez pas, il vous suffit de créer un nouveau compte. Dans ce cas, on nous demandera de nous attribuer un mot de passe dans les deux cas.

Plus tard, on entre avec le "mot de passe" et on peut vérifier quel est le solde dont on dispose, logiquement s'il est nouveau, le solde sera nul.



Nous allons maintenant examiner comment nous allons déposer quelques éthers (pièces numériques) sur notre compte test Rinkeby.

En haut, nous avons la possibilité de choisir le type de réseau que nous utiliserons, par défaut lorsque vous entrez, cela vous place dans le "réseau principal d'Ethereum", cependant, lorsque vous cliquez, nous pouvons passer en revue tous les réseaux optionnels que nous pouvons choisir, dans notre cas nous avons choisi le réseau Rinkeby.



La prochaine étape consiste à déposer des éthers sur notre compte de test référencé par le réseau Rinkery.

NOTE IMPORTANTE : Les éthers (monnaie numérique) que nous déposons sur notre compte référencé au réseau de test Rinkery n'ont aucune valeur sur le marché de la cryptographie, ils ne seront utilisés par nous que pour le test de logiciels. Vérifiez toujours sur quel réseau nous travaillons pour éviter les erreurs dans les transactions.

Afin d'obtenir de l'éther pour les tests, nous devons effectuer la procédure suivante.

Dans tout compte twitter que vous avez, nous devrons entrer dans twitter et créer un commentaire qui n'inclut que le compte et/ou l'adresse que nous avons dans Metamask ; ensuite, nous devrons copier le lien du commentaire puisque nous avons celui-ci ; nous irons au lien suivant pour ancrer notre compte.

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

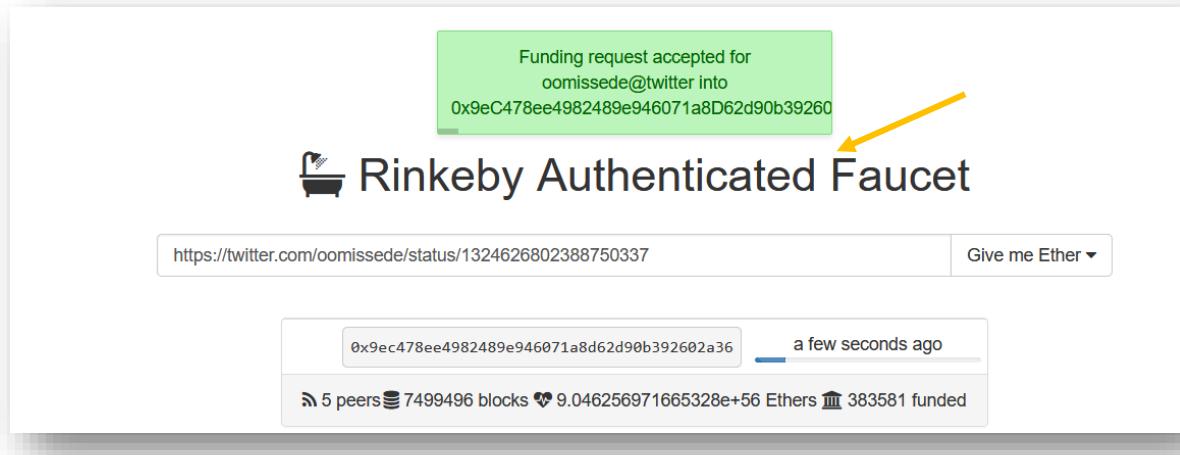
Nous avons créé un commentaire sur twitter et copié le lien du commentaire.

A screenshot of a Twitter tweet page. The URL in the address bar is https://twitter.com/oomissede/status/1324628185468854272. A yellow arrow points to the URL. The tweet content is: "0x88ac6dcecc3603c7042f4334fc06db8e8d7062d5". Below the tweet are options to "Traducir Tweet" and "2:22 a. m. · 6 nov. 2020 · Twitter Web App". Below the tweet is a link "Ver actividad del Tweet". At the bottom are icons for reply, retweet, like, and share.

Sur le site <https://faucet.rinkeby.io/> nous copions le lien twitter et sur le bouton "Give me Ether" nous choisissons la quantité désirée.

A screenshot of the Rinkeby Authenticated Faucet website at https://faucet.rinkeby.io. A yellow arrow points to the input field containing the Twitter link. Another yellow arrow points to the "Give me Ether" dropdown menu, which shows options: "3 Ethers / 8 hours", "7.5 Ethers / 1 day", and "18.75 Ethers / 3 days". Below the input field, there is a section titled "How does this work?" with instructions for requesting funds via Twitter or Facebook. At the bottom, there is a note about reCaptcha protection and a footer with the logo of the organization.

Enfin, si tout est correct, nous annoncerons que le dépôt a été accepté et, en fonction de la charge de travail du système du réseau Rinkeby, le dépôt sera effectué dans quelques minutes ou pourra prendre plus de temps.



Maintenant que nous avons des Ethers sur notre compte, nous pouvons commencer les tests sur la plateforme Ethereum.

Nous disposons de deux extensions pour interagir avec la chaîne de blocage Ethereum.

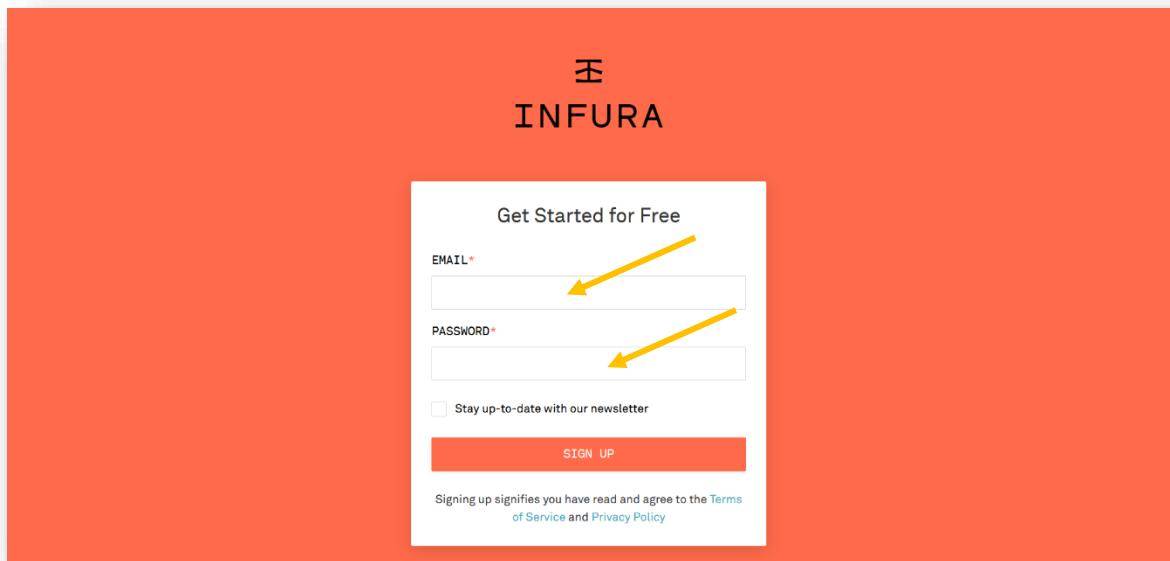
L'extension de la **co-solidation**. **BlockoinETHEREUM.aix** contient les fonctions permettant d'effectuer les opérations suivantes :

- Création de nouveaux comptes (adresses) dans la chaîne Ethereum (privateKey, publicKey, Address)
- Stockage des données de compte (adresses) dans des fichiers binaires.
- Importation de comptes de fichiers binaires (adresses)
- Obtention d'un compte (adresse) via privateKey.
- Création et envoi de transactions entre comptes (adresses) "en ligne".
- Création, signature et envoi de transactions PushRaw hors ligne.
- Consultation des détails de la transaction Tx.
- Vérification du solde des adresses et des contrats intelligents.
- Compilateur de contrats intelligents.
- Création, publication et exécution de contrats intelligents.
- Création, publication et exécution de Token ERC20 (cryptomoney token).
- Obtention du code ABI à partir d'un contrat intelligent.
- Vérification de la connexion au réseau.
- Interrogation de la valeur de l'éther sur le marché du cryptage dans n'importe quel pays du monde (monnaie nationale du pays) - Taux de change.
- Consultation sur le prix du gaz.
- Consultation du "nonce" d'un compte spécifique.

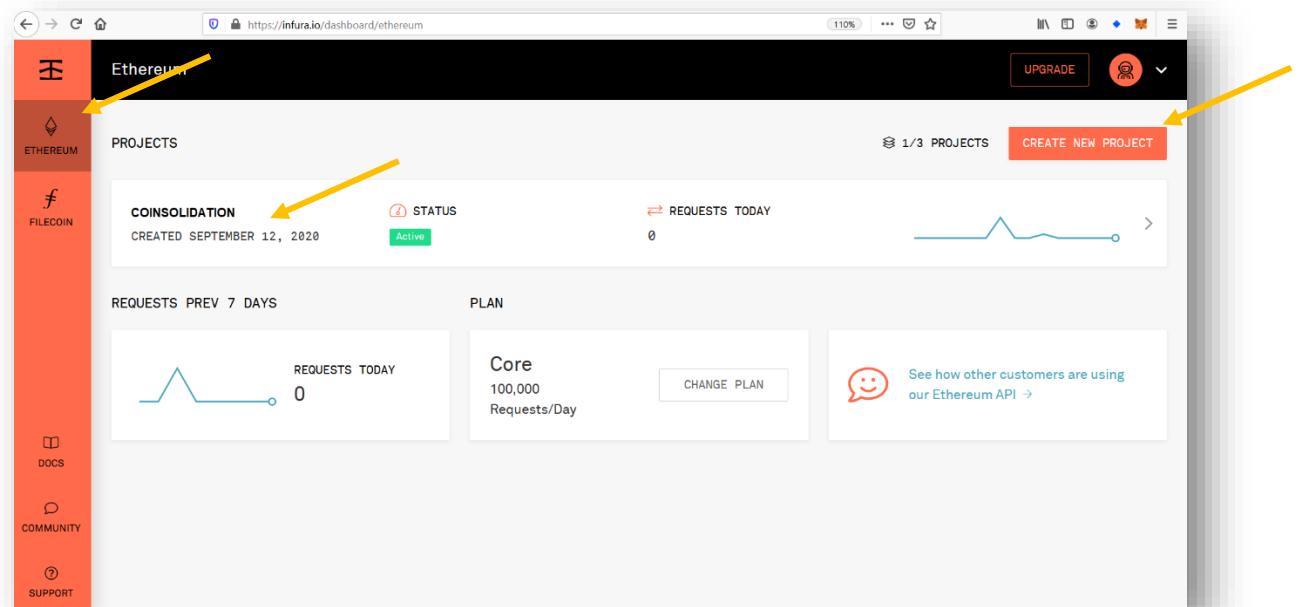
L'extension de la **BlockoinINFURA.aix** nous fournit les 40 fonctionnalités de la plateforme Infura.io. Pour plus de détails, veuillez consulter la documentation json-rpc directement sur <https://infura.io/docs>

Pour pouvoir utiliser l'extension INFURA, vous devez créer un compte sur le site infura.io car nous avons besoin d'une KEY API pour pouvoir envoyer des requêtes au réseau Ethereum, ainsi que pour pouvoir utiliser les réseaux de test Ethereum.

L'ouverture d'un compte est simple, comme le montre le schéma ci-dessous.



Une fois le compte créé, on entre et on peut avoir l'API clé des différents réseaux. On va en haut à gauche et on clique sur Ethereum, puis on voit les projets, on crée un nouveau projet et on se présente à ce projet.

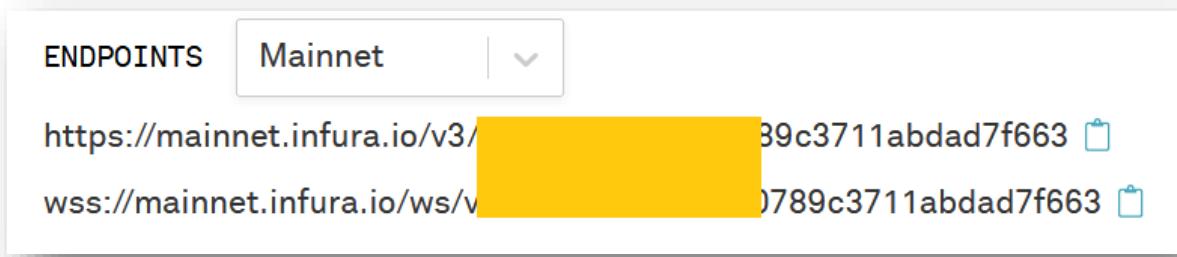


Dans le cadre du projet, nous disposerons des données de l'API Clé et des différents réseaux que nous pouvons utiliser.

Dans le cadre du projet, nous pouvons examiner la Clé API (ID de projet) et sélectionner le réseau sur lequel nous allons travailler ou les liens complets des ENDPOINTS à choisir.

The screenshot shows the Infura.io dashboard interface. On the left sidebar, there are links for COINSOLIDATION, ETHEREUM (selected), FILECOIN, DOCS, COMMUNITY, and SUPPORT. The main content area has tabs for REQUESTS and SETTINGS, with SETTINGS selected. A 'SAVE CHANGES' button is at the top right. The 'KEYS' section contains fields for PROJECT ID (39c3711abdad7f663) and PROJECT SECRET (1ac27e75434133134), both with copy icons. Below these are dropdown menus for ENDPOINTS (set to Mainnet) and SECURITY settings (checkboxes for project secret and JWT). Arrows from the text above point to the PROJECT ID, PROJECT SECRET, and the ENDPOINTS dropdown.

Par exemple, pour utiliser le réseau principal de l'Ethereum, nous prendrions la ligne suivante :

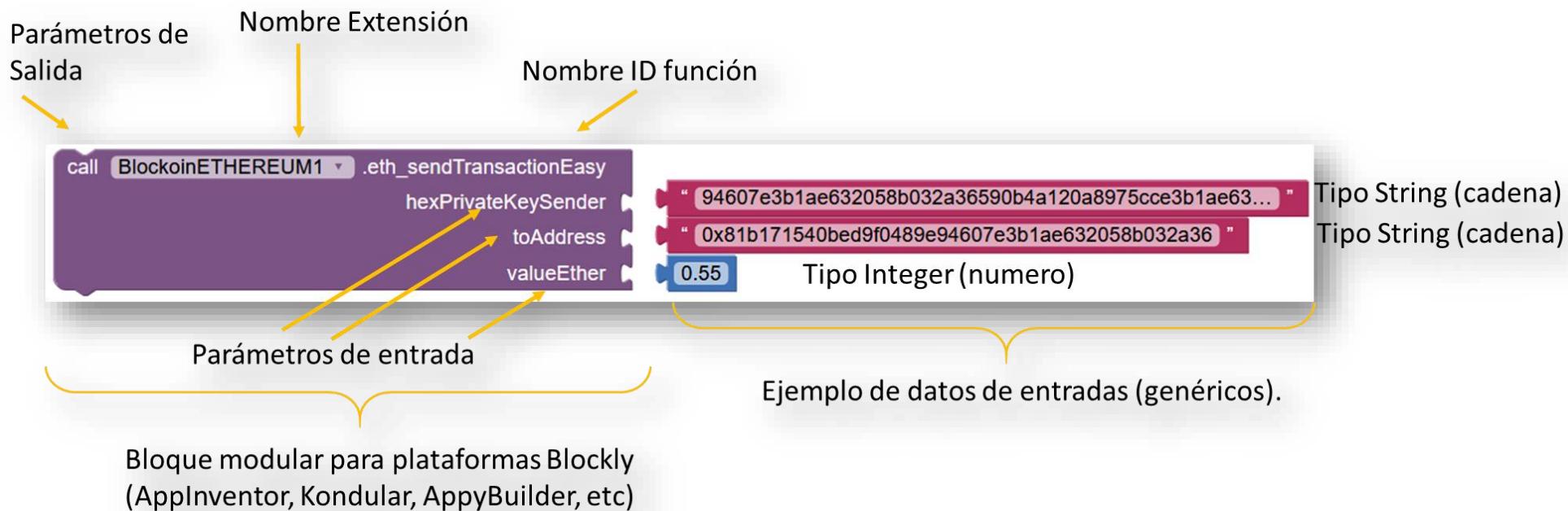


NOTE : Les extensions ont été testées sur les systèmes AppInventor, Kondular, Thunkable et AppyBuilder.

7. Définition et utilisation des blocs (fonction générique)

Nous commencerons par expliquer la répartition des données que tous les blocs auront, leur syntaxe d'utilisation et leur configuration.

Dans l'exemple suivant, nous pouvons voir un bloc modulaire et ses paramètres d'entrée et de sortie, ainsi que les types de données d'entrée, ces données peuvent être de type String (chaîne de caractères) ou Integer (entier ou décimal). Nous montrons comment il est utilisé et le configurons pour son bon fonctionnement.



Chaque bloc de module aura sa description et sera nommé au cas où il aurait une ou plusieurs dépendances obligatoires ou facultatives d'autres blocs utilisés comme paramètres d'entrée, le processus d'intégration sera annoncé.

8. Caractéristiques et événements d'Exchange Ethereum Extension (EEE).

***Réseau d'essai que nous utiliserons **Rinkeby**, comme urlNetwork, lorsque vous voulez faire des transactions réelles, il vous suffit de changer le réseau urlNetwork pour **mainnet**.

Blocage pour vérifier la connexion internet - (**CheckInternetConnection**).

call **BlockoinETHEREUM1** .**CheckInternetConnection**

Paramètres d'entrée : Non applicable.

Paramètres de sortie : Retourne "Vrai" si vous avez une connexion ou retourne "Faux" s'il n'y a pas de connexion.

Description : Bloc permettant de vérifier la connexion internet et d'envoyer des données (transactions).

Bloc pour générer une **nouvelle** adresse "Offline" - (**GenerateNewAddressEthereum**)

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

Paramètres d'entrée : **phrasaHex <String>**.

Paramètres de sortie : Événement (**OutputGenerateNewAddressEthereum**)

Les résultats : **PrivateKey<String>**, **PublicKey<String>** , **addressEtehreum<String>**.

when **BlockoinETHEREUM1** .**OutputGenerateNewAddressEthereum**
privKeyEther pubKeyEther addressEthereum
do

Description : créer une nouvelle adresse (compte) basée sur une phrase ou une séquence de chiffres. La nouvelle adresse peut être créée sans réseau ni connexion internet - "Offline".

Bloc pour générer une nouvelle adresse "en ligne" - (**GenerateNewAddressEthereum**)

call **BlockoinETHEREUM1** .**GenerateNewAddressEthereumAPI**

Paramètres d'entrée : Non applicable.

Paramètres de sortie : retour au format de données JSON ; privateKey, publicKey, adresse.

Exemple de sortie :

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

Description : créer une nouvelle adresse d'ethereum (compte). Il est nécessaire d'avoir un accès ou une connexion à l'internet puisque la génération se fait par le service REST API - "Online".

Bloc pour générer une nouvelle adresse "Offline" et sauvegarder les clés publiques et privées dans des fichiers binaires - (**GenerateNewAddressEthereumStoreKeys**)



Paramètres d'entrée : **pathFilePrivateKey<String>** , **pathFilePublicKey<String>**.

Paramètres de sortie : Événement (**OutputGenerateNewAddressEthereumStoreKeys**)

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



Description : crée une nouvelle adresse éthérée aléatoire (compte) et stocke les clés publiques et privées dans des fichiers binaires qui seront utilisés pour importer et exporter les données du compte. La nouvelle adresse peut être créée sans réseau ni connexion internet - "Offline".

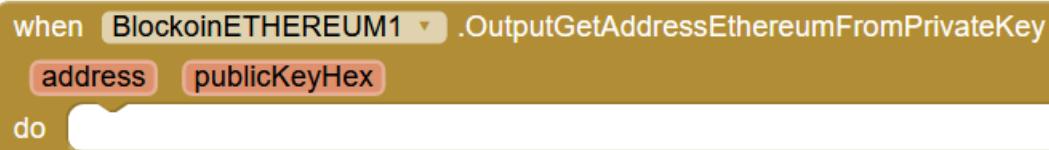
Bloc pour générer une clé publique - (**GeneratePublicKeyHexFromPrivateKeyHex**).



Paramètres d'entrée : **hexPrivateKey <String>**.

Paramètres de sortie : Événement (**OutputGeneratePublicKeyHexFromPrivateKeyHex**)

Sorties : **address<String>** , **publicKeyHex<String>**.



Description : Crée la clé publique basée sur l'entrée d'une clé privée.

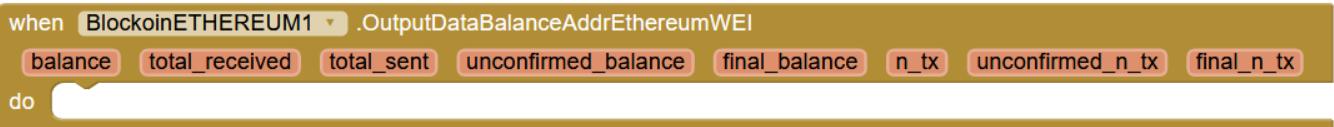
Bloc pour obtenir le solde d'une adresse - (**GetBalanceAddrEthereum**).



Paramètres d'entrée : **hexPrivateKey <String>**.

Paramètres de sortie : Événement (**OutputGeneratePublicKeyHexFromPrivateKeyHex**)

Sorties : **balance<Corde>** , **total_received<Corde>** , **total_sent<Corde>** ,
unconfirmed_balance <Corde> , **final_balance<Corde>** , **n_tx<Corde>** ,
unconfirmed_n_tx<Corde> , **final_n_tx<Corde>**.



Description : affiche le bilan et les données détaillées du compte (adresse).

Blocage pour vérifier si l'interface réseau de votre mobile est activée - (GetDataNetworkConnection).

call BlockoinETHEREUM1 .GetDataNetworkConnection

Paramètres d'entrée : Non applicable.

Paramètres de sortie : Événement (OutputGeneratePublicKeyHexFromPrivateKeyHex)

Sorties : nom d'interface<Corde>, est connecté<Corde>.

when BlockoinETHEREUM1 .OutputGetDataNetworkConnection
interfacename isconnected
do

Description : affiche le nom de l'interface mobile et indique si l'interface est activée ou non.

Bloc pour signer la transaction "Offline" - (SignerGenericPushRawTransactionOffline)

call BlockoinETHEREUM1 .SignerGenericPushRawTransactionOffline

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

Dépendance(s) requise(s) : Bloc (eth_getTransactionCount), Bloc (eth_SendRawTransactionInfura)

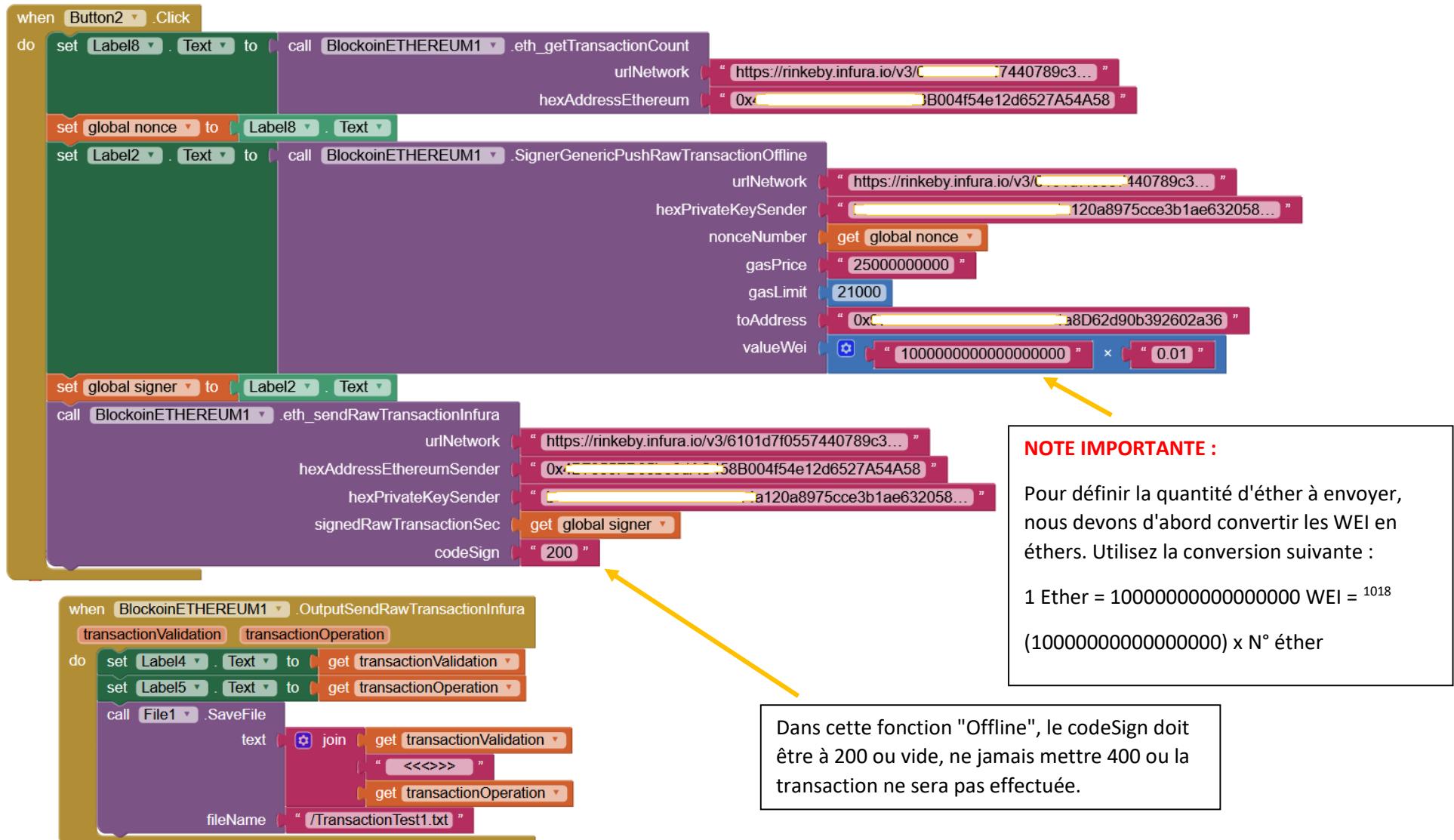
Parámetros de entrada : urlNetwork <String>, hexPrivateKeySender <String>, nonceNumber <String>, gasPrice <String>, gasLimit <Integer>, toAddress <String>, valueWEI <Integer>.

Paramètres de sortie :

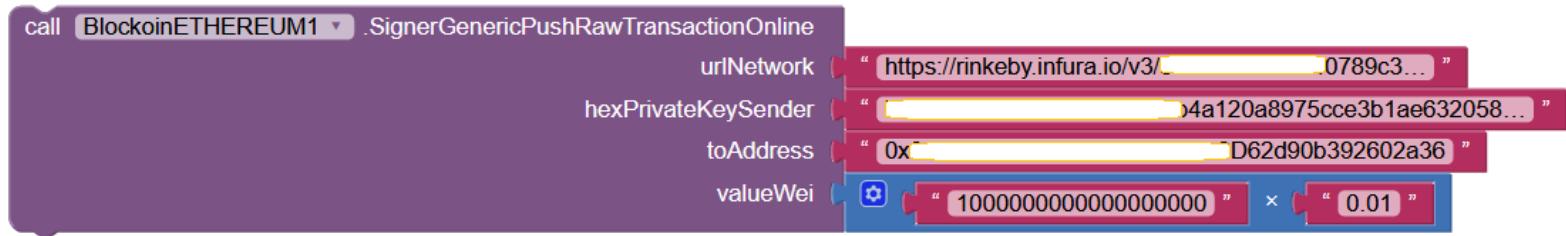
Résultats : Transaction signée à envoyer. <Corde>.

Description : Prépare une nouvelle transaction à envoyer (cryptée et signée). Il peut être traité sans réseau ni connexion Internet - "Offline".

Exemple d'utilisation complète avec des dépendances de blocs (`SignerGenericPushRawTransactionOffline`).



Bloc pour signer une transaction "en ligne" - (`SignerGenericPushRawTransactionOnline`).

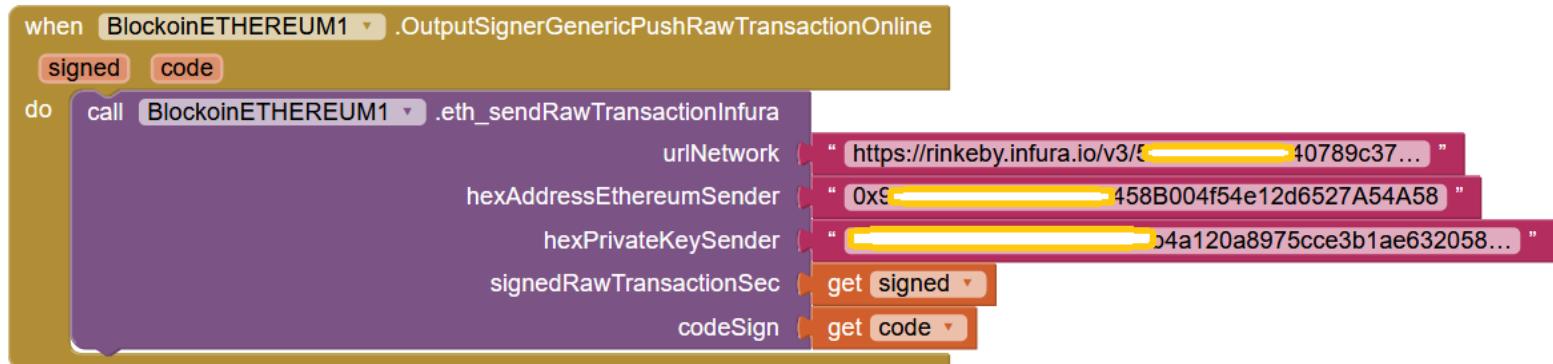


Unité(s) obligatoire(s) : Block (`eth_SendRawTransactionInfura`).

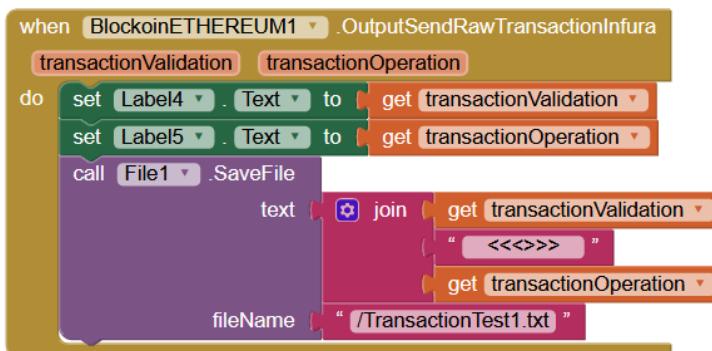
Paramètres d'entrée : urlNetwork <String>, hexPrivateKeySender <String>, toAddress <String>, valueWEI <Integer>.

Paramètres de sortie : événements utilisés dans l'ordre suivant
(OutputSignerGenericPushRawTransactionOnline) et **(OutputSendRawTransactionInfura)**.

Sorties : signé<Chaîne>, code<Chaîne>.

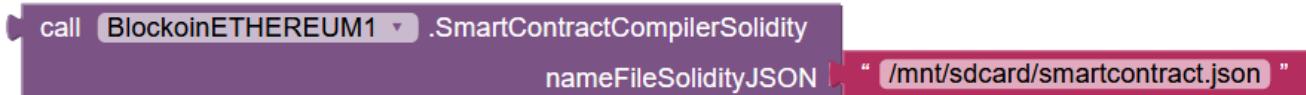


Les sorties bloquent eth_sendRawTransactionInfura : transactionValidation<String> , transactionOperation<String>.



Description : Prépare une nouvelle transaction à envoyer (cryptée et signée). Une connexion réseau ou internet est nécessaire - "en ligne".

Bloc pour la compilation du contrat Smart "Online" - (SmartContractCompilerSolidity).



Paramètres d'entrée : nameFileSolidityJSON <String>.

Paramètres de sortie : affiche le contrat intelligent compilé, cette fonction nous aide à vérifier s'il est bien écrit avant de publier un contrat intelligent dans le réseau Ethereum.

Résultats : Code compilé.

Le contrat Smart doit être dans un fichier au format JSON.

Exemple d'un contrat de base Smart en langage Solidity.

```
pragma solidity ^0.5.0 ;  
  
contrat fatal {  
propriétaire de l'adresse ;  
function mortal() { owner = msg.sender ; }  
function kill() { if (msg.sender == owner) suicide(owner) ; } }  
l'accueil des contrats est mortel {  
salutation par chaîne de caractères ;  
fonction greeter(string _greeting) public { greeting = _greeting ; }  
fonction greet() constant returns (string) {return greeting;}
```

Exemple de contrat Smart précédent au format JSON avec commentaires.

```
# Vérifier la compilation de la solidité par un test non publié
# En utilisant l'exemple de la solidité du contrat "greeter", le "bonjour
monde" d'Ethereum.
```

Dossier : smartcontract.json

```
{
  "solidité" : "contract mortal {\n    /* Définir une variable propriétaire\n    du type address*/\n    address owner;\n    /* cette fonction est\n    exécutée à l'initialisation et fixe le propriétaire du contrat */\n    function mortal() { owner = msg.sender ; }\n    /* Fonction pour\n    récupérer les fonds sur le contrat */\n    function kill() { if\n        (msg.sender == owner) suicide(owner) ; }\n}\n\n\ncontract greeter is\nmortal {\n    /* définition de la variable greeting du type string */\n    string greeting ;\n    /* ceci s'exécute lorsque le contrat est exécuté */\n    fonction greeter(string greeting) public {\n        greeting =\n    }\n}
```

```
_greeting;\n    }\\n    /* fonction principale */\\n    fonction greet()\nconstant returns (string) {\\n        return greeting;\\n    }\\n}",\n"params". ["Hello BlockCypher Test"]\n}
```

NOTE IMPORTANTE : Le format JSON doit toujours comporter un retour à la ligne à la fin de chaque ligne.

Exemple de production compilée de Smartcontract.

```

[{"nom" : "u003cstdin\u003e:greeter",
 "solidit\u00e9" : "contract mortal {\n/* D\u00e9finir une variable\npropri\u00e9taire du type address*/\naddress owner;\n/* cette\nfonction est ex\u00e9cut\u00e9e \u00e0 l'initialisation et d\u00e9finit le propri\u00e9taire du\ncontrat */\nfunction mortal() { owner = msg.sender ; }\n/*\nFonction pour r\u00e9cup\u00e9rer les fonds sur le contrat */\nfunction kill()\n{ if (msg.sender == owner) suicide(owner) ; }\n}\n\ncontract greeter is mortal {\n/* d\u00e9finition de la variable greeting du type\nstring */\nstring greeting;\n/* ceci s'ex\u00e9cute lorsque le\ncontrat est ex\u00e9cut\u00e9 */\nfunction greeter(string _greeting) public\n{ greeting = _greeting; }\n/* fonction principale */\nfunction greet() constant returns (string) {\nreturn greeting;\n}\n}",
 "bin" : "606060405260405161023e38038061023e8339810160405280510160008054600160a060020a031916331790558060016000509080519060200190828054600181600116156101000203166002900490600052602060002090601f016020900481019282601f10609f57805160ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565b50505061016e806100d06000396000f35b828001600101855582156076579182015b82811115607657825182600050559160200191906001019060b0565b509056606060405260e060020a600035046341c0e1b58114610026578063cfcae321714610068575b005b6100246000543373ffffffffff08116911614156101375760005473fff0908116911614156101375760005473fff16ff5b6100c9600060609081526001805460a06020601f6002600019610100868816150201909416939093049283018190040281016040526080828152929190828280156101645780601f1061013957610100808354040283529160200191610164565b60405180806020018281038252838181518152602001915080519060200190808383829060006004602084601f0104600f02600301f150905090810190601f1680156101295780820380516001836020036101000a031916815260200191505b50925050506
 "abi" : [
 {
 "constant" : faux,
 "intrants" : [],
 "nommer" : "tuer",
 "outputs" : [],
 "type" : "fonction"
 },
 {
 "constant" : vrai,
 "intrants" : [],
 "nom" : "saluer",
 "r\u00e9sultats" : [

```

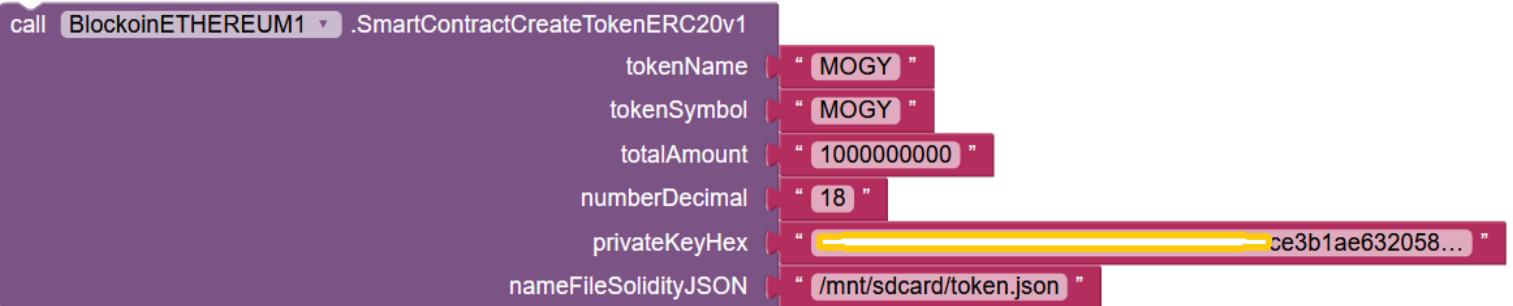
```

        {
            "nom" : "",
            "type" : "chaîne de caractères
        }
    ],
    "type" : "fonction
},
{
    "intrants" : [
        {
            "nom" : "salut",
            "type" : "chaîne de caractères
        }
    ],
    "type" : "constructeur
}
],
"params" : [
    "Test Hello BlockCypher"
]
},
{
    "nom" : "mortel",
    "solidité" : "contract mortal {\n/* Définir une variable\npropriétaire du type address*/\naddress owner;\n/* cette\nfonction est exécutée à l'initialisation et définit le propriétaire du\ncontrat */\nfunction mortal() { owner = msg.sender ; }\n/*\nFonction pour récupérer les fonds sur le contrat */\nfunction kill()\n{ if (msg.sender == owner) suicide(owner) ; }\n}\n\ncontract greeter is mortal {\n/* définition de la variable greeting du type\nstring */\nstring greeting;\n/* ceci s'exécute lorsque le\ncontrat est exécuté */\nfunction greeter(string _greeting) public\n{ greeting = _greeting; }\n/* fonction principale */\nfunction greet() constant returns (string) {\nreturn greeting;\n}\n}",
    "bin" :
"606060405260008054600160a060020a03191633179055605c8060226000396000f36060
60405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffffffffffff
ffffffffffffffffffff90811691161415605a5760005473fffffffffffff16ff5b56",
    "abi" : [
        {
            "constant" : faux,
            "intrants" : [],
            "nommer" : "tuer",
            "outputs" : [],
            "type" : "fonction
        },
        {
            "intrants" : [],
            "type" : "constructeur
        }
    ],
    "params" : [
        "Test Hello BlockCypher"
]
}

```

]

Bloc pour compiler, créer et publier le jeton ERC20 - (SmartContractCreateTokenERC20v1)

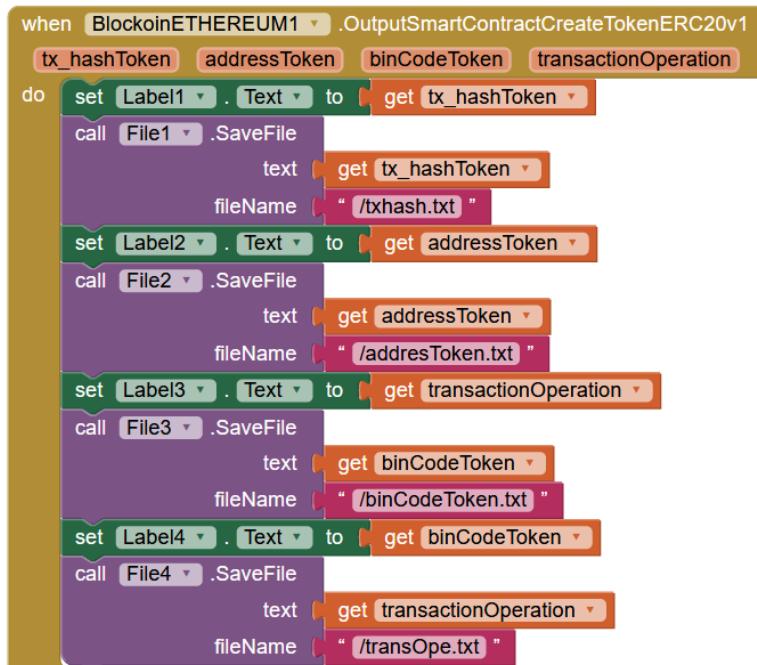


Unité(s) obligatoire(s) : Block (**CreateTestingFie**). **IMPORTANT** : Vous devez d'abord utiliser ce bloc pour vous assurer que le chemin est correct, car si vous ne donnez pas un chemin valide dans le bloc (**SmartContractCreateTokenERC20v1**) la création du jeton ne sera pas exécutée car le paramètre d'entrée "nameFileSolidityJSON" est utilisé pour créer un fichier temporaire.

Paramètres d'entrée : **tokenName <String>**, **tokenSymbol <String>**, **totalAmount <String>**, **numberDecimal <String>**, **privateKeyHex <Integer>**, **nameFileSolidityJSON <String>** Ce fichier est le chemin valide pour **créer un fichier temporaire**, vous devez vous assurer que le chemin est valide pour tester que le fichier est créé ; vous pouvez utiliser le Block (**CreateTestingFile**) après l'avoir utilisé ; vérifiez qu'il a été créé avec succès.

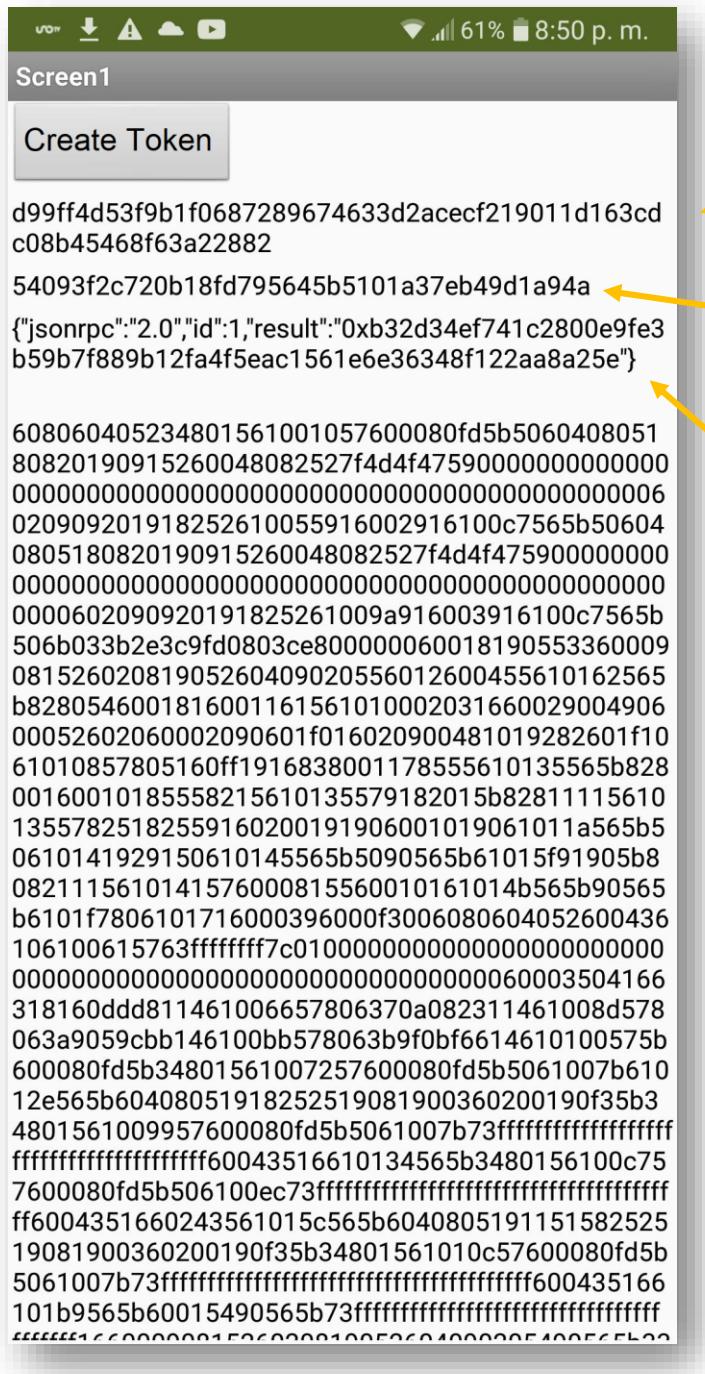
Paramètres de sortie : Événement (**OutputSmartContractTokenERC20v1**).

Sorties : **tx_hashToken<String>** , **addressToken<String>** , **binCodeToken<String>** , **trasactionOperation<String>**.



Description : contrat intelligent "Token ERC20" - actif publié sur le réseau Ethereum. La version 1 (v1) a déjà le paramètre Limite de gaz fixé à 500 000 WEI.

Exemple de sortie de la fonction précédente **SmartContractCreateTokenERC20v1**.



Transaction (Tx_hash) de la création du réseau Ethereum. Ce document peut être consulté à l'adresse suivante :

Adresse du nouveau contrat qui fait référence au nouveau jeton ERC20 créé.

Transaction (Tx_hash) de
l'opération validée dans le système
CoinSolidation.org

Vous pouvez consulter le site suivant : www.etherscan.io

Code binaire (BIN) du nouveau contrat de jeton qui a été traité dans l'EVM (Ethereum Virtual Machine).

9. Étapes pour créer un CryptoToken ou un Cryptomoney Token.

Étape 1.

Vérifiez que le chemin temporaire sur l'appareil mobile où le contrat Smart a été créé est valide et qu'un fichier peut être créé avec succès. Cela se fait à l'aide du bloc (**CreateTestingFile**). Vérifiez que le fichier test indiqué dans l'entrée "pathTestFile" a été créé, le chemin d'accès est en général indiqué par : /mnt/sdcard/nom_fichier.txt

Étape 2 (facultative).

Dans cette étape, nous vérifierons si le compte (adresse) à partir duquel l'opération sera facturée a un solde suffisant pour effectuer la transaction de création et de publication d'un contrat Smart. Cela peut être vérifié en utilisant le bloc (**eth_VerifiBalanceForTransaccionSmartContract**). Cette utilisation de ce bloc est facultative puisque les blocs qui génèrent le **SmartContractCreateTokenERC20v1** ou **SmartContractCreateTokenERC20v2** contiennent déjà cette vérification en interne.

Étape 3.

Sélectionnez le bloc à utiliser pour créer le jeton ERC20, vous avez deux options :

- a.- Le bloc **SmartContractCreateTokenERC20v1** a déjà la valeur implicite de la limite de gaz attribuée avec une valeur de 500.000 WEi.
- b.- Block **SmartContractCreateTokenERC20v2** a la possibilité de pouvoir configurer la limite de gaz en fonction des besoins de l'utilisateur final ou du développeur. Il convient de noter que si une limite de gaz très basse, inférieure à 350 000 Wei, est donnée, il est très possible que la Smart contr.

Quatrième étape.

Utilisez les blocs **SmartContractCreateTokenERC20v1** ou **SmartContractCreateTokenERC20v2** en **vous** assurant que lorsque vous utilisez la variable d'entrée "nameFileSolidityJSON", elle est égale à la variable d'entrée "pathTestFile" du bloc (**CreateTestingFile**) déjà cochée à l'étape 1.

Cinquième étape.

Avant d'exécuter la création d'un jeton ERC20 avec l'un des blocs **SmartContractCreateTokenERC20v1** ou **SmartContractCreateTokenERC20v2**, il est recommandé de sauvegarder les valeurs des événements (résultats) comme il convient pour sauvegarder les résultats (tx_hashToken, addressToken, binCodeToken, transactionOperation). Voir l'exemple de sortie de la fonction précédente **OutPutSmartContractCreateTokenERC20v1**.

Sixième étape.

Exécuter la création du jeton ERC20 puis le publier pour la vente. Voir section 10.

10. Comment mettre en vente un nouvel actif ou votre jeton de crypte (Token ERC20).

Depuis que nous avons créé un ERC20 - Cryptomoney Token (voir le bloc **SmartContractCreateTokenERC20v1** ou avec le bloc **SmartContractCreateTokenERC20v2**), nous devons le télécharger sur un échange afin que n'importe qui dans le monde puisse l'acheter. Un échange est un site sur Internet où de nouveaux jetons sont publiés.

Les échanges sont classés en deux types, centralisés et décentralisés. La principale différence est que l'on est régi et contrôlé par une sorte d'organisme international (centralisé) et que les décentralisés n'ont personne avec les auditeurs. Bien que cela puisse donner plus de confiance, la réalité est que récemment les décentralisés ont pris plus de force et sont utilisés la plupart du temps sans problèmes majeurs.

L'une des meilleures pratiques en matière de gestion de biens de toute nature consiste à ne pas les regrouper sur un seul compte, mais à les répartir sur plusieurs comptes pour assurer la sécurité des clés privées et publiques.

Dans notre cas, nous utiliserons un échange décentralisé mais déjà avec une histoire pas mal, nous utiliserons www.forkdelta.app

A ce moment nous devons déjà avoir installé l'application pour le navigateur (Mozilla ou Chorme) **METAMASK** www.metamask.io c'est parce que la Bourse pour visiter votre page www.forkdelta.app doit se connecter au compte que nous avons Ethereum.

Un point important est que notre compte Ethereum que nous avons déjà dans METAMASK devrait avoir un solde de plus ou moins 10 USD ; en effet, lorsque nous publierons notre nouveau jeton ERC20 que nous avons créé avec le bloc **SmartContractCreateTokenERC20v1** ou avec le bloc **SmartContractCreateTokenERC20v2**, nous devrons payer la transaction pour le publier sur la Bourse.

Pour pouvoir utiliser la Bourse www.forkdelta.app, nous devons disposer des données de jeton suivantes que nous voulons publier pour la vente sur la Bourse.

Adresse du nouveau jeton ERC20 que nous avons créé précédemment.

0x54093F2C720b18Fd795645b5101A37EB49d1A94a

Nombre de décimales utilisées par notre toucher.

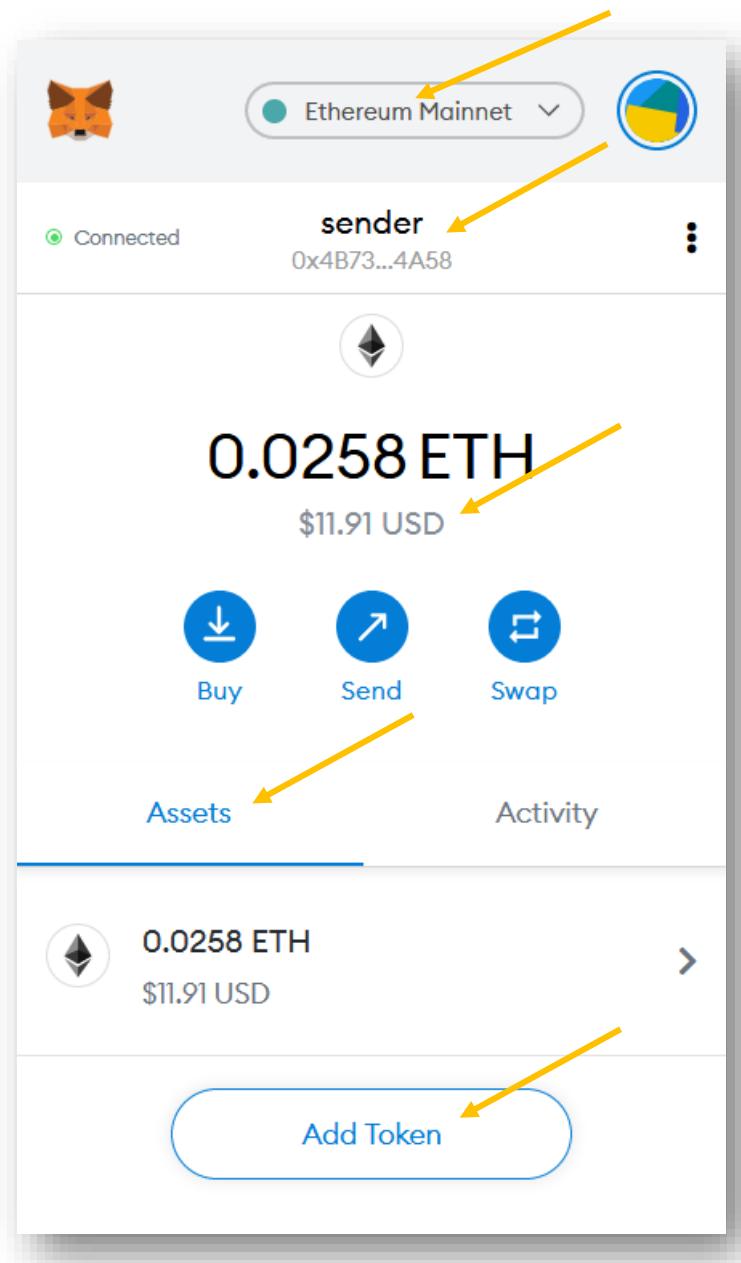
18

Le nom qui identifie le jeton.

MOGY

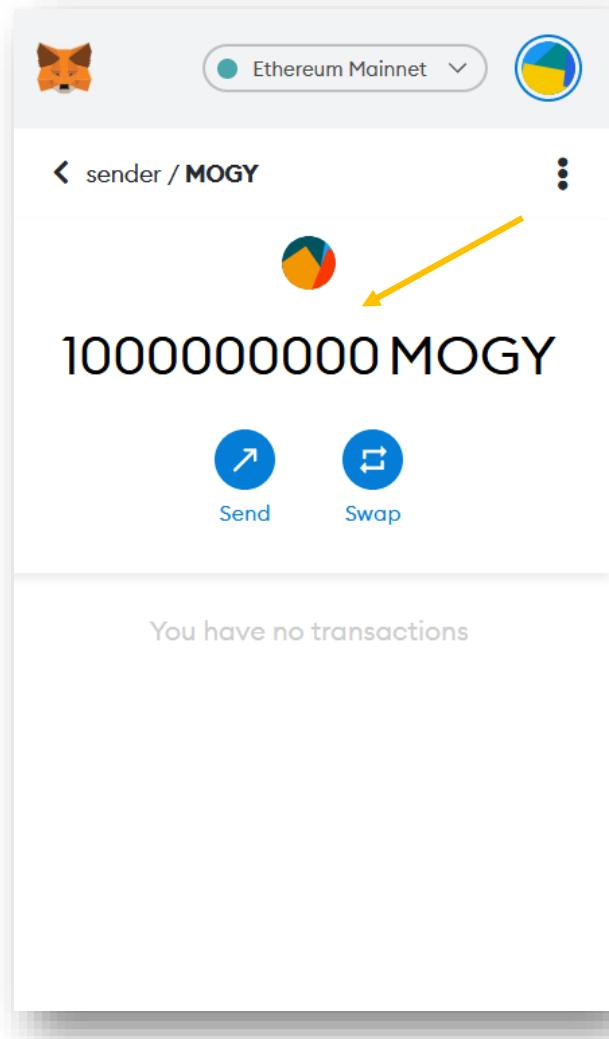
Commençons par vérifier que le jeton que nous avons créé possède les paramètres mentionnés ci-dessus et que ce sont ceux avec lesquels il a été initialement créé.

Allons à **METAMASK** et assurons-nous d'abord que nous sommes sur le compte avec lequel nous avons créé le jeton. Ensuite, allez en bas et cliquez sur le bouton "Ajouter un jeton".



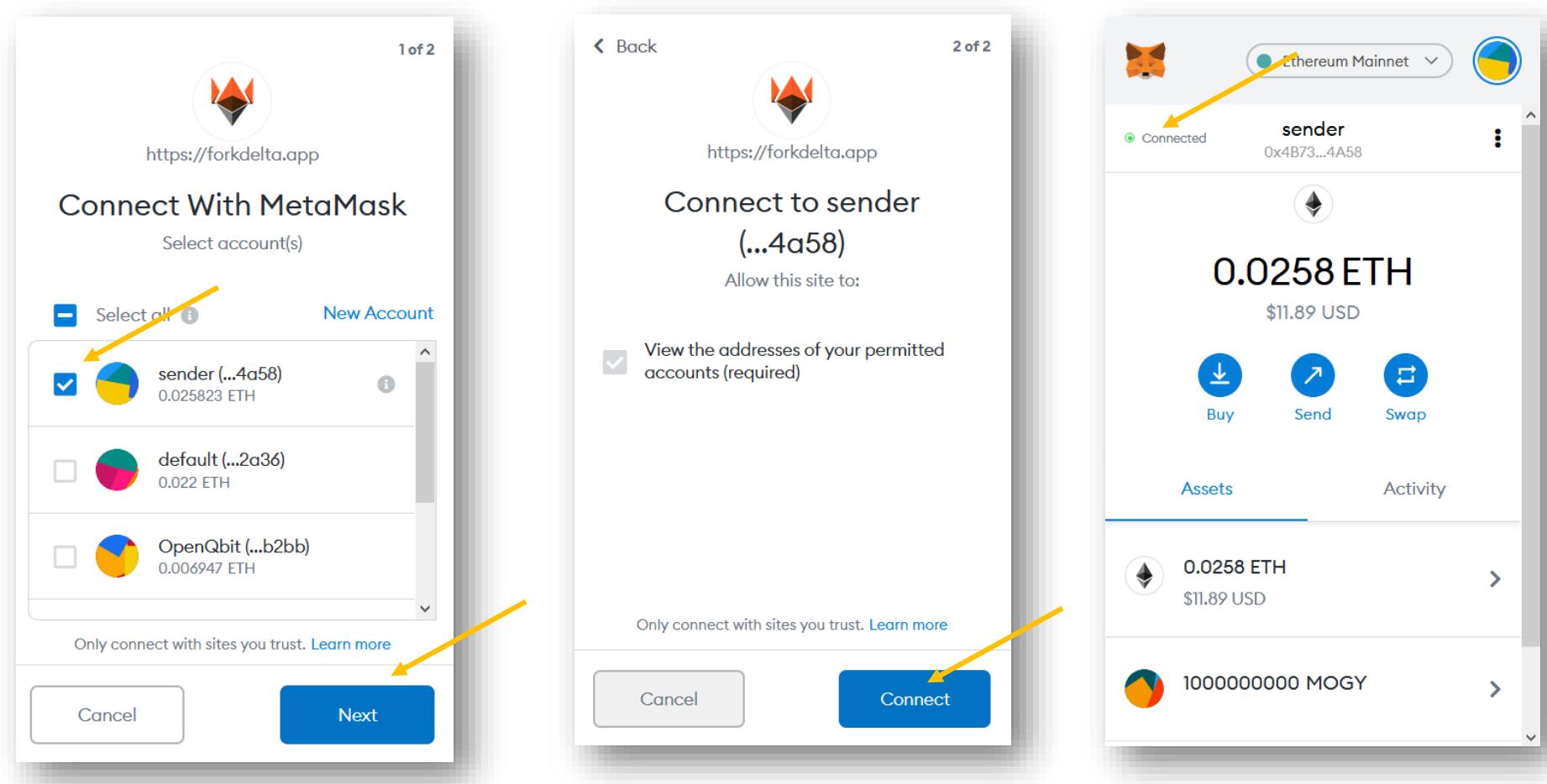
Nous allons maintenant ajouter notre nouveau jeton pour voir votre solde actuel. Après avoir cliqué sur le bouton "Jetton personnalisé" en haut de la page, entrez les informations demandées dans les champs suivants et cliquez ensuite sur le bouton "Suivant". Ensuite, notre nouveau jeton apparaîtra avec le solde dont vous disposez. Si vous n'avez pas de solde, vérifiez si vous êtes sur le compte avec lequel vous avez créé le jeton. Si vous n'êtes pas sur un compte avec lequel vous avez créé le jeton, vous aurez un solde nul car vous n'avez pas encore acheté de jeton.

The screenshot shows the 'Add Tokens' interface. At the top, there is a search bar and a 'Custom Token' button, which is highlighted with a yellow arrow. Below it is a field for 'Token Contract Address' containing the value '0x54093F2C720b18Fd795645b5101A3'. Another yellow arrow points to this field. Underneath is a 'Token Symbol' field containing 'MOGY', also with a yellow arrow pointing to it. A 'Decimals of Precision' field contains the value '18', with a yellow arrow pointing to it. At the bottom are two buttons: 'Cancel' and 'Next', with a yellow arrow pointing to the 'Next' button.



Dès que nous aurons mis notre nouveau jeton en vente sur la Bourse [www.forkdelta.app](https://forkdelta.app)

Lorsque vous êtes sur le site d'échange, il vous sera demandé de vous connecter au site <https://forkdelta.app>. Cliquez sur le bouton "Next" ci-dessous, puis sur "Connect" et enfin vous pourrez vérifier que vous êtes bien connecté au site de forkdelta.app



Il est temps de lancer le nouveau jeton sur la Bourse <https://forkdelta.app>

Cliquez sur le menu supérieur "DAI" et allez à la fin du défilement et choisissez l'option "Autre".

The screenshot shows the ForkDelta app interface. At the top, there is a navigation bar with links for FAQs, Help, Tokens, Smart Contract, English, and MetaMask. The URL in the address bar is https://forkdelta.app/#/trade/DAI-ETH. On the left, there is a sidebar with sections for Balance, Volume, and a search bar. The Balance section shows DAI and ETH amounts. The Volume section shows various tokens like ASTRO, REQ, SXDT, and SNOV with their current values. A yellow arrow points from the text above to the 'DAI' dropdown menu in the top right corner of the sidebar. Another yellow arrow points to the 'Other' option in the dropdown menu. The main trading area has tabs for Order Book, Price Chart, and Trades. The Order Book tab shows a list of orders for DAI/ETH, DAI, and ETH. The Price Chart shows price levels from 0.00 to 1.00 over a 24-hour period. The Trades tab shows a list of recent trades for DAI/ETH, DAI, and ETH. Below the trading area, there is a 'My Transactions' section with tabs for Important, Trades, Orders, and Funds. A red button at the bottom of this section says 'chat on Telegram reddit ForkDelta'. There is also a 'URL & Status' section with a link to the project's URL and a note about the interface being a fork of EtherDelta. A 'How to Start Trading' section provides instructions. On the right side, there is a 'Tweets' section by @ForkDelta with a recent tweet from MARKETHA (@markorjic) about DeFi. The bottom right corner shows the MetaMask extension icon with the balance 0.026 ETH.

Ensuite, nous enregistrons le nouveau jeton avec les données que nous connaissons déjà.

The screenshot shows the ForkDelta app interface. A modal window titled "Other token" is open, prompting for token details. The "Address" field contains the value `0x54093F2C720b18Fd795645b5101A37EB49d1A94a`. The "Name" field contains `MOGY`. The "Decimals" field contains the value `18`. The background shows a "Balance" section with DAI and ETH amounts, and a "Trades" section listing recent transactions.

Pour l'instant, le nouveau jeton apparaîtra dans la partie supérieure gauche de la Bourse, nous l'avons seulement mis en ligne sur la Bourse, nous devons le publier pour que tout le monde puisse l'acheter et le voir. Maintenant, nous devons déposer un peu d'Ether (0,015) est suffisant de notre compte à la Bourse.

The image consists of two side-by-side screenshots of the ForkDelta website, both titled "ForkDelta MOGY".

Left Screenshot (MOGY Deposit):

- Balance:** Shows a "Deposit" tab selected. Under "Token", "MOGY" is listed with an amount of "1000000000.000" in the "Wallet" column and "0.000" in the "ForkDelta" column. Below this, there is an "Amount" input field for MOGY and a "Deposit" button.
- Volume:** Shows a search bar and a list of tokens with their bid and ask prices. The tokens listed are ASTRO, REQ, SXDT, and SNOV.
- Search Bar:** At the bottom, it says "Escribe aquí para buscar" (Type here to search).

Right Screenshot (ETH Deposit):

- Balance:** Shows a "Deposit" tab selected. Under "Token", "ETH" is listed with an amount of "0.026" in the "Wallet" column and "0.000" in the "ForkDelta" column. Below this, there is an "Amount" input field containing "0.015" and a "Deposit" button.
- Volume:** Shows a search bar and a list of tokens with their bid and ask prices. The tokens listed are ASTRO, REQ, SXDT, and SNOV.
- Text Overlay:** A tooltip appears over the "Deposit" button for ETH, stating: "Use this to deposit from your personal Ethereum wallet ("Wallet" column) to the current smart contract ("ForkDelta" column)."
- Search Bar:** At the bottom, it says "Escribe aquí para buscar" (Type here to search).

Depuis que nous avons effectué le dépôt, nous pouvons déposer autant de jetons que nous le souhaitons sur la Bourse www.forkdelta.app

Pour déposer une quantité définie de jetons, nous devons créer un ordre d'achat, ce qui se fait en bas où il est indiqué "Nouvelle commande" et nous cliquons sur l'option "Vendre". Ensuite, nous entrons la quantité de jetons que nous voulons mettre à la disposition de tout acheteur dans le monde, le prix en éther que nous voulons vendre pour chacun d'entre eux (prix unitaire) et le paramètre "Expires" est la durée pendant laquelle nous voulons avoir ces jetons à vendre :

Temps d'expiration = 14 secondes X montant saisi.

Exemple : 14 secondes X 10000 = 140 000 secondes = 1,62 jours

The screenshot shows the ForkDelta trading platform interface. On the left, there's a 'Balance' section with tabs for Deposit, Withdraw, and Transfer. Under the Token tab, it shows MOGY in Wallet (1000000000.000) and ForkDelta (0.00). Below this are input fields for Amount (with placeholder 'Amount') and ETH (with placeholder 'ETH'). To the right of these are blue 'Deposit' buttons. A note at the bottom of this section says: 'Make sure MOGY is the token you actually want to trade. Multiple tokens can share the same name.' On the right side of the screen is the 'Order Book' section, which currently displays: 'There are no orders here.' At the bottom right, there's a 'New Order' form. This form has tabs for Buy and Sell, with 'Sell' being active. It includes fields for MOGY (containing '10000'), MOGY/ETH (containing '0.05'), ETH (containing '500.000'), and Expires (containing '10000'). A large orange 'Sell' button is at the bottom of the form. Five yellow arrows point from the text above to the 'Sell' tab, the 'MOGY' field, the 'MOGY/ETH' field, the 'Expires' field, and the 'Sell' button respectively.

Là, vos nouveaux jetons sont en vente, tout le monde peut venir les acheter. Parfois, il ne reconnaît pas le nouveau jeton et doit donc les vendre à partir de l'application Metamask.

Exemple de consultation sur le site www.etherscan.io du nouveau jeton créé.

The screenshot shows the Etherscan interface for a transaction. The transaction hash is 0xd99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882. The status is Success. It was included in block 11240201 with 224 block confirmations. The timestamp is 47 mins ago (Nov-12-2020 02:49:56 AM +UTC) and it was confirmed within 30 secs. The transaction originated from address 0x4b7355fd05be6dac458b004f54e12d6527a54a58. The recipient is a contract address [Contract 0x54093f2c720b18fd795645b5101a37eb49d1a94a Created]. The value sent is 0 Ether (\$0.00). The transaction fee is 0.011014691 Ether (\$5.06). The gas price is 0.000000041 Ether (41 Gwei) and the gas limit is 500,000. The gas used by the transaction is 268,651 (53.73%).

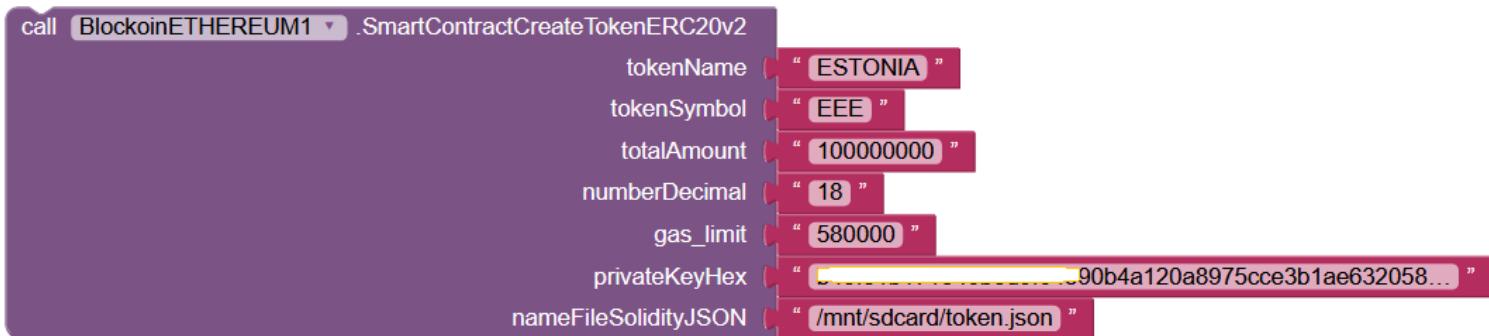
Transaction (Tx_hash) de la création du réseau Ethereum.

Jeton d'adresse de contrat nouvellement créé.

Le réseau Ethereum ne coûte que le coût.

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

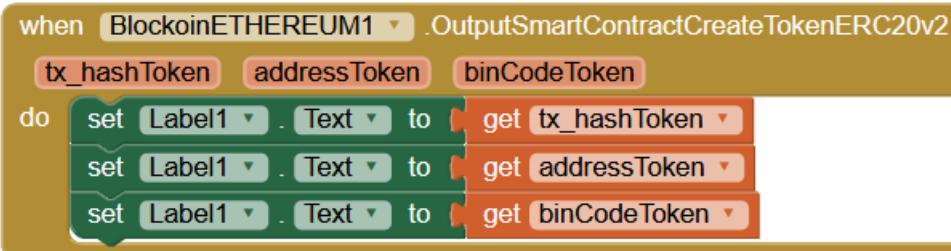
Bloc pour compiler, créer et publier le jeton ERC20 - (**SmartContractCreateTokenERC20v2**)



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

Paramètres de sortie : Événement (**OutputSmartContractCreateTokenERC20v2**).

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



Description : contrat intelligent "Token ERC20" - actif publié sur le réseau Ethereum. La version 2 (v2) permet d'optimiser la valeur de la limite de gaz car, selon le contrat intelligent, la vitesse à laquelle vous souhaitez effectuer la publication sur le réseau d'éthereum est variable. Il est recommandé d'avoir une valeur minimale de 350.000 WEI pour la publication sans échec ou retard.

La différence entre les fonctions de création de TokenERC20v1 et de création de TokenERC20v2 est que dans le cas de la version 1, le paramètre de GasLimit est déjà pré-configuré et dans la version 2, il peut être configuré selon les besoins de l'utilisateur ou du développeur.

Bloc pour appeler ou exécuter le jeton ERC20 - (**SmartContractExecution**)



Paramètres d'entrée : addressSmartContract<String>, nameFileSolidityJSON<String>, functionExecutionSmartContract<String>.

Paramètres de sortie : Exécution de la fonction spécifiée dans le Smart Contract référencé.

Résultats : **Exécution d'un contrat intelligent.**

NOTE : Pour être exécuté, un fichier doit être créé au format JSON qui contient les paramètres de la clé primaire de l'adresse où vous voulez exécuter le contrat Smart, vous devez entrer la limite de gaz (WEI).

Exemple de fichier JSON pour exécuter les fonctions du contrat Smart compilé en exemple de la fonction de compilation mentionnée ci-dessus. Le nom du fichier peut être arbitraire.

Fichier : call.json

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

Exemple de la sortie de la fonction "accueil" du contrat Smart :

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

Bloc d'affichage de la propriété des jetons ERC20 - (**SmartContractGetCreationTx**)

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

Paramètres d'entrée : **txSmartContract<String>**

Paramètres de sortie : affiche les propriétés du contrat Smart référencé par (**Tx_hash**).

Description : présente les principales caractéristiques et le code ABI du contrat Smart référencé.

Exemple de propriétés du jeton ERC20, exemple de tokenName "MOGY" précédemment créé à l'aide de la fonction (**martContractCreateTokenERC20v1**)

```
{
  "block_hash" : "cadb8914c43ed28fb370c9e1b6f5d8818ba31a1e944d1f7049441b982902dea8",
  "block_height" : 11240201,
  "block_index" : 170,
  "hash" : "d99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882",
  "adresses" : [
    "4b7355fd05be6dac458b004f54e12d6527a54a58"
  ],
  "total" : 0,
  "frais" : 110146910000000,
  "taille" : 958,
  "gas_limit" : 500 000,
  "gas_used" : 268651,
  "prix_du_gaz" : 41000000000,
  "contrat_création" : vrai,
  "relayed_by" : "200.77.24.87",
  "confirmed" : "2020-11-12T02:49:56Z",
  "received" : "2020-11-12T02:50:13.185Z",
  "voir" : 0,
  "double_spend" : faux,
```

Bloc pour afficher le code de compilation du jeton ERC20 - ([SmartContractGetcodeABI](#))

Paramètres d'entrée : addressSmartContract<String>

Paramètres de sortie : affiche le code de contrat Smart référencé.

Description : indique le code ABI du contrat Smart référencé.

Exemple de code ABI d'un contrat Smart générique :

```

"solidité" : "contract mortal { \n /* Définir une variable propriétaire
du type address */\n     address owner;\n         /* cette fonction est
exécutée à l'initialisation et fixe le propriétaire du contrat */\n
function mortal() { owner = msg.sender ; }\n         /* Fonction pour
récupérer les fonds sur le contrat */\n         function kill() { if
(msg.sender == owner) suicide(owner) ; }\n}\n\n\n\nnccontract greeter is
mortal {\n     /* définition de la variable greeting du type string */\n
string greeting ;\n         /* ceci s'exécute lorsque le contrat est exécuté
*/\n         fonction greeter(string _greeting) public {\n             greeting =
_greeting;\n         }/* fonction principale */\n         fonction greet()\nconstant returns (string) {\n             return greeting;\n         }\n}","bin" :
"606060405260405161023e38038061023e8339810160405280510160008054600160a060
020a031916331790558060016000509080519060200190828054600181600116156101000
203166002900490600052602060002090601f016020900481019282601f10609f57805160
ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565b50505
061016e806100d06000396000f35b828001600101855582156076579182015b8281111560
7657825182600050559160200191906001019060b0565b509056606060405260e060020a6
00035046341c0e1b58114610026578063cf3ae321714610068575b005b6100246000543373
ffffffffffffffffff0908116911614156101375760005473fff
ffffffffffff0908116911614156101375760005473fff
a06020601f600260001961010086881615020190941693909304928301819004028101604
052608082815292919082820156101645780601f10610139576101008083540402835291
60200191610164565b6040518080602001828103825283818151815260200191508051906
0200190808383829060006004602084601f0104600f02600301f150905090810190601f16
80156101295780820380516001836020036101000a031916815260200191505b509250505
06
"abi" : "[{\\"constant\\":false,\\"inputs\\"
:[],\\"name\\":\\"kill\\",\\"outputs\\"
:[],\\"type\\":\\"function\\"}, {\\"constant\\":true,\\"inputs\\"
:[],\\"name\\":\\"greet\\",\\"outputs\\"
:[{\\"name\\":\"\\"",\\"type\\":\\"string\\"}],\\"type\\":\\"function\\"}, {\\"inputs\\"
:[{\\"name\\":\"\\"_greeting\\\",\"type\\":\\"string\\"}],\\"type\\":\\"constructor\\"}
],",
"creation_tx_hash" :
"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
"created" : "2016-07-20T01:54:50Z",
"address" : "0eb688e79698d645df015cf2e9db5a6fe16357f1"
* Avant d'utiliser le bloc suivant (SmartContractPublish), vous devez utiliser le bloc
(SmartContractCompilerSolidity) pour vérifier si le contrat Smart est bien écrit.

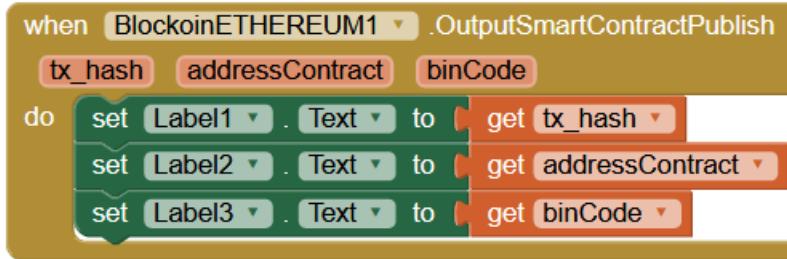
```

Bloque de la publication du jeton ERC20 sur le réseau Ethereum - ([SmartContractPublish](#)).

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

Paramètres d'entrée : **nameFileSolidityJSON<String>**

Paramètres de sortie : affiche les propriétés du contrat Smart référencé. Dans l'éventualité (**OutputSmartContractPublish**).



Description : publier dans le réseau ethereum le contrat Smart référencé dans le fichier JSON.

Exemple de fichier : **PubishSmartContract.json**

```

{
  "solidité" : "contract mortal { \n/* Définir une variable propriétaire\n du type address */\naddress owner;\n/* cette fonction est\n exécutée à l'initialisation et définit le propriétaire du contrat */\nfunction mortal() { owner = msg.sender ; } \n/* Fonction pour\n récupérer les fonds sur le contrat */\nfunction kill() { if\n(msg.sender == owner) suicide(owner) ; } \n}\n\n\ncontract greeter is\nmortal {\n/* définition de la variable greeting du type string */\nstring greeting;\n/* ceci s'exécute lorsque le contrat est exécuté */\nfonction greeter(string _greeting) public {\n_greeting = _greeting;\n} \n/* fonction principale */\nfonction greet()\nconstant returns (string) {\nreturn greeting;\n} \n}",
  "params" : ["Hello Test"],
  "publier" : ["greeter"],
  "privé" : "3ca40...",
  "gas_limit" : 500000
}
  
```

Comme le montre le code ci-dessus, il s'agit du même code JSON utilisé dans l'exemple de la fonction de compilation au même code ; les paramètres ont été ajoutés à la fin du fichier JSON :

Params : paramètres implicites de la contr.

Publier : Nom de la manière dont le contrat Smart sera publié.

Privé : la clé primaire du compte qui exécutera le contrat Smart doit avoir un solde.

Gas_limit : C'est le solde de l'IEM que vous voulez dépenser pour la publication du contrat Smart.

Exemple de sortie lors de l'exécution (publication du contrat Smart) : le contrat Smart est entré dans le réseau ethereum. Il indique la transaction effectuée "**creation_tx_hash**" et l'adresse assignée de l'"**adresse**" du contrat Smart créé.

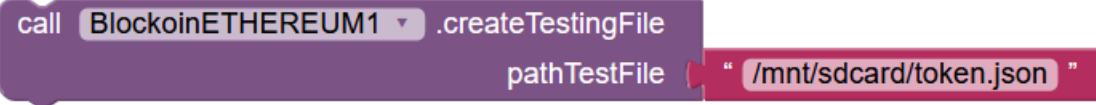
```
[
{
  "nom" : "greeter",
  "solidité" : "contract mortal { \n /* Définir une variable\npropriétaire du type address*/\n      address owner; \n      /* cette\nfonction est exécutée à l'initialisation et définit le propriétaire du\ncontrat */\n      function mortal() { owner = msg.sender ; } \n      /*\nFonction pour récupérer les fonds sur le contrat */\n      function kill()\n{ if (msg.sender == owner) suicide(owner) ; } \n} \n\n\n\nnccontract\ngreeter is mortal { \n      /* définition de la variable greeting du type\nstring */\n      string greeting ; \n      /* ceci s'exécute lorsque le\ncontrat est exécuté */\n      fonction greeter(string _greeting) public\n{ \n          greeting = _greeting; \n      } \n      /* fonction principale */\n      fonction greet() constant returns (string) { \n          return\ngreeting; \n      } \n} ",
  "bin" :
"606060405260405161023e38038061023e8339810160405280510160008054600160a060
020a031916331790558060016000509080519060200190828054600181600116156101000
203166002900490600052602060002090601f016020900481019282601f10609f57805160
ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565b50505
061016e806100d06000396000f35b828001600101855582156076579182015b8281111560
7657825182600050559160200191906001019060b0565b509056606060405260e060020a6
00035046341c0e1b58114610026578063cfae321714610068575b005b6100246000543373
ffffffffffffffffff908116911614156101375760005473fff
ffffffffffff908116911614156101375760005473fff
a06020601f600260001961010086881615020190941693909304928301819004028101604
0526080828152929190828280156101645780601f10610139576101008083540402835291
60200191610164565b6040518080602001828103825283818151815260200191508051906
0200190808383829060006004602084601f0104600f02600301f150905090810190601f16
80156101295780820380516001836020036101000a031916815260200191505b509250505
06
  "abi" : [
    {
      "constant" : faux,
      "intrants" : [],
      "nommer" : "tuer",
      "outputs" : [],
      "type" : "fonction"
    },
    {
      "constant" : vrai,
      "intrants" : [],
      "nom" : "saluer",
      "résultats" : [
        {
          "nom" : "",
          "type" : "chaîne de caractères
        }
      ],
      "type" : "fonction"
    },
    {

```

```

        "intrants" : [
            {
                "nom" : "_salut",
                "type" : "chaîne de caractères
            }
        ],
        "type" : "constructeur
    }
],
"gas_limit" : 500 000,
"creation_tx_hash" :
"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
"address" : "0eb688e79698d645df015cf2e9db5a6fe16357f1",
"params" : [
    "Bonjour Test"
]
},
{
    "nom" : "mortel",
    "solidité" : "contract mortal {\n /* Définir une variable\npropriétaire du type address*/\n    address owner;\n    /* cette\nfonction est exécutée à l'initialisation et définit le propriétaire du\ncontrat */\n    function mortal() { owner = msg.sender ; }\n    /*\nFonction pour récupérer les fonds sur le contrat */\n    function kill()\n    { if (msg.sender == owner) suicide(owner) ; }\n}\n\ncontract\ngreeter is mortal {\n    /* définition de la variable greeting du type\nstring */\n    string greeting;\n    /* ceci s'exécute lorsque le\ncontrat est exécuté */\n    fonction greeter(string _greeting) public\n    {\n        greeting = _greeting;\n    }\n    /* fonction principale */\n    fonction greet() constant returns (string) {\n        return greeting;\n    }\n}",
    "bin" :
"606060405260008054600160a060020a03191633179055605c8060226000396000f36060
60405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffffffffffff
ffffffffffffffffffffffffffff90811691161415605a5760005473fffffffffffffff
ffffffffffffffffffffffff16ff5b56",
    "abi" : [
        {
            "constant" : faux,
            "intrants" : [],
            "nommer" : "tuer",
            "outputs" : [],
            "type" : "fonction
        },
        {
            "intrants" : [],
            "type" : "constructeur
        }
    ],
    "gas_limit" : 500 000,
"params" : ["Hello Test"]
}
]
```

Bloc de test pour créer un fichier - (`createTestingFile`)

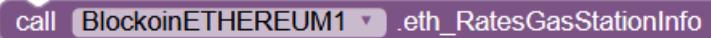


Paramètres d'entrée : `pathTestFile<String>`

Paramètres de sortie : un fichier test est créé dans le chemin référencé.

Description : Cela sert à vérifier le chemin de création du fichier temporaire valide pour s'assurer qu'il est correct lors de l'utilisation du bloc (`SmartContractCreateTokenERC20v1`) ou du bloc (`SmartContractCreateTokenERC20v2`).

Bloc pour obtenir les tarifs du prix du gaz - (`eth_RatesGasStationInfo`).



Paramètres d'entrée : `nameFileSolidityJSON<String>`

Paramètres de sortie : affiche les propriétés du contrat Smart référencé. Dans l'événement (`OutputEth_GasStationInfo`), les valeurs fournies sont données en GWEI.

Le prix du gaz est la valeur qui sera payée aux systèmes qui exécutent les transactions dans le réseau d'Ethereum. Ces systèmes sont communément appelés "mineurs" et la valeur du prix du gaz est fonction de la rapidité (temps et priorité) de la transaction dans le réseau d'Ethereum.

Les valeurs délivrées sont fonction des durées d'exécution suivantes. Ces délais sont approximatifs et peuvent varier en fonction de la façon dont vous traitez les demandes (transactions) sur le réseau Ethereum.

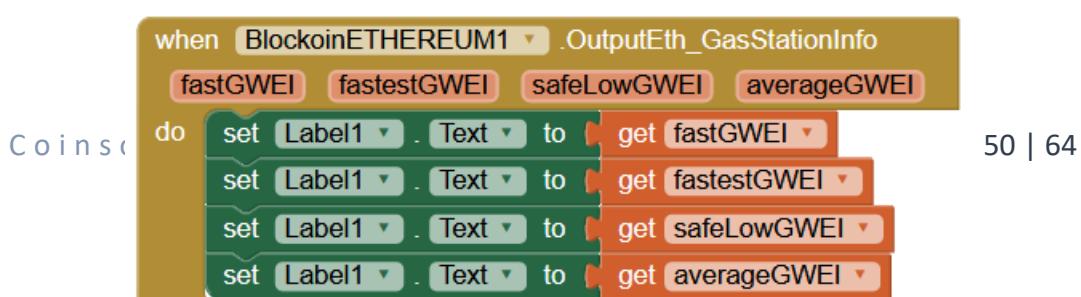
Rapide < 2 minutes.

Le plus rapide < 30 secondes.

SafeLow < 30 minutes.

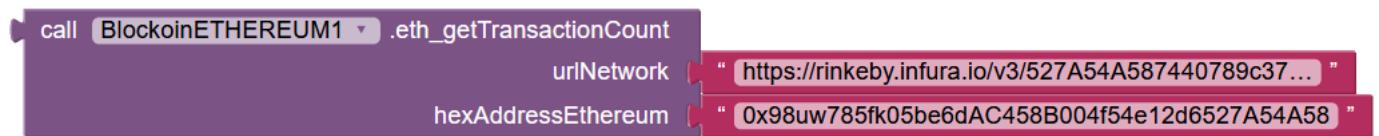
Moyenne < 15 minutes.

Les transactions effectuées avec l'extension de l'éthereum d'échange (EEE) utilisent toujours le prix du gaz = moyenne.



Description : Obtenir le prix du gaz mis à jour au moment de la requête pour créer une nouvelle transaction

Bloquez pour obtenir le nombre "nonce" - (**eth_getTransactionCount**).



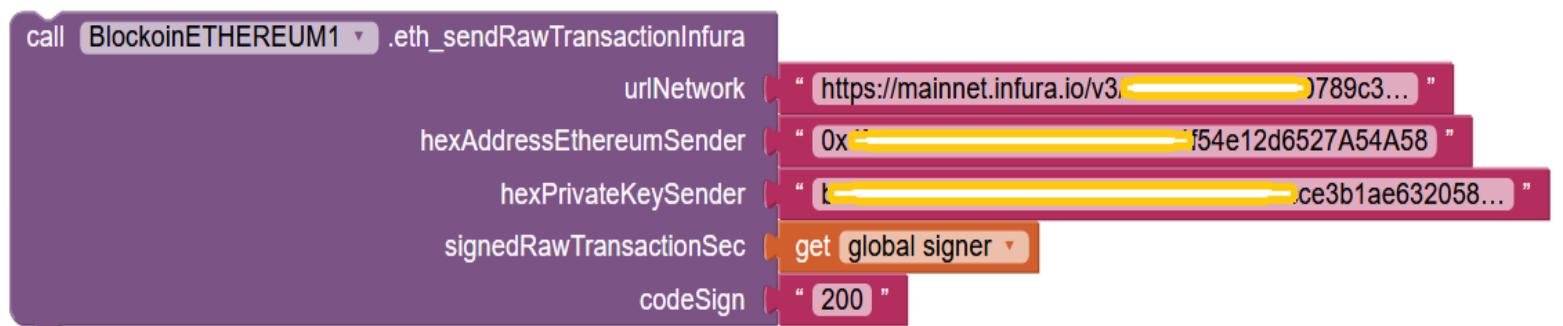
Paramètres d'entrée : **urlNetwork<String>**, **hexAddressEthereum<String>**.

Paramètres de sortie : Il affiche en format hexadécimal le numéro consécutif "nonce" de l'adresse référencée.

Le numéro "nonce" est un numéro supplémentaire qui permet de suivre le nombre de transactions qui ont été effectuées à partir d'une adresse spécifique.

Description : permet d'obtenir le numéro "nonce" de l'adresse référencée.

Bloc pour envoyer une transaction signée - (**eth_SendRawTransactionInfura**).

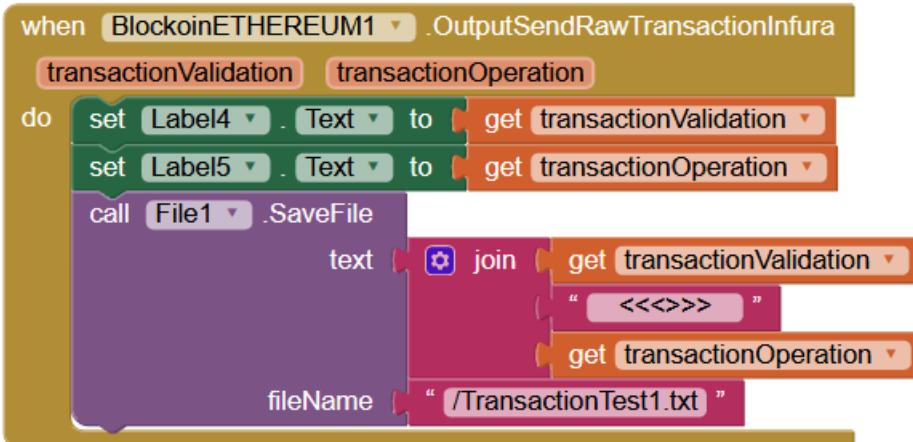


Dépendance(s) requise(s) : Bloc (**eth_getTransactionCount**), Bloc (**SignerGenericPushRawTransactionOffline**). Voir l'exemple du bloc (**SignerGenericPushRawTransactionOffline**).

Paramètres d'entrée : urlNetwork <String>, hexAddressEthereumSender <String>, hexPrivateKeySender <String>, SignedRawTrasactionSec <String>, codeSign <String>.

Paramètres de sortie : Événement (OutputSendRawTransactionInfura)

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



Description : Il fournit deux valeurs hexadécimales à la suite des transactions. La valeur de la transactionValidation est la transaction effectuée sur le réseau Ethereum qui contient le coût implicite de l'Ethereum. La valeur de la transactionOpération est la transaction effectuée dans le réseau Coinsolidation avec un coût de 0,5 cents USD pour chaque transaction basé sur la valeur de l'éther au moment de la transaction. Ce coût est destiné au paiement des services du réseau Coinsolidation.org et est investi dans la maintenance, le support et la création d'extensions pour le secteur de la cryptographie et des actifs dans le monde entier.

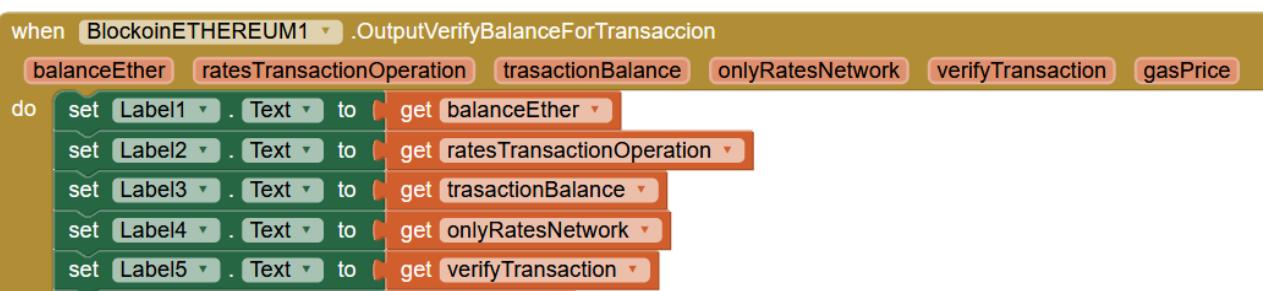
Bloc pour calculer le coût d'une transaction standard - (eth_VerifiBalanceForTransaction)



Paramètres d'entrée : addressEthereumSender<String>, valueEthertoSend<String>.

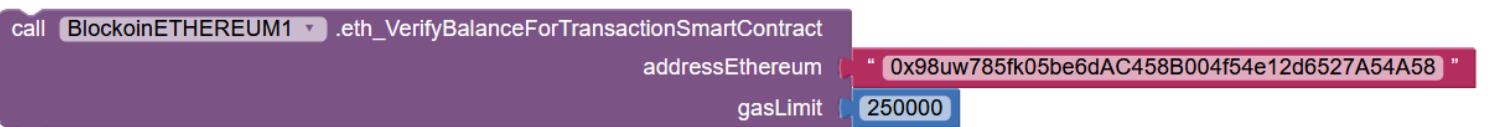
Paramètres de sortie : Événement (OutputVerifyBalanceForTransaction).

Sorties : balanceEther<String> , ratesTransactionOperation<String> ,
 transactionBalance<String> , OnlyRatesNetwork<String> , verifyTransaction<String> ,
 gasPrice<String>.



Description : Fournit des détails sur le coût d'une transaction standard par rapport à l'adresse d'entrée. Le paramètre de sortie "verifyTransaction" nous indique si la transaction peut être effectuée "True" ou si l'adresse référencée n'a pas un solde suffisant nous donnera un "False".

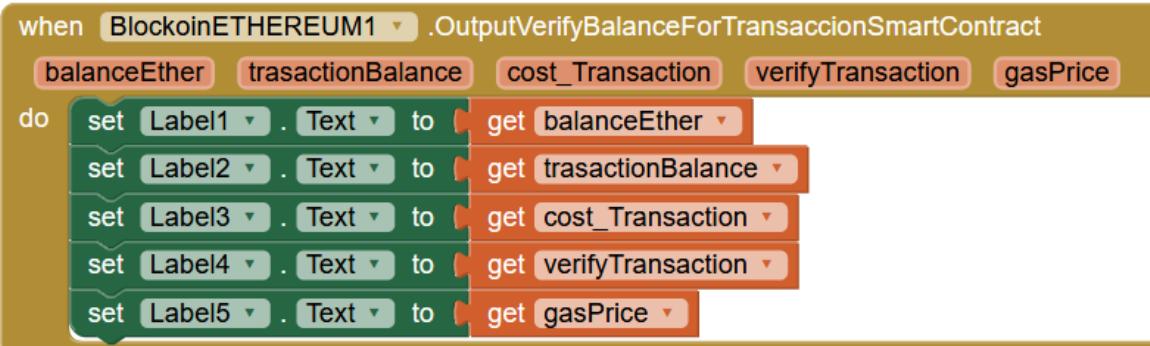
Bloc pour calculer le coût d'une transaction standard -
(eth_VerifiBalanceForTransaccionSmartContract)



Paramètres d'entrée : **addressEthereum<String>**, **gasLimit<String>**.

Paramètres de sortie : Événement (**OutputVerifyBalanceForTransaccionSmartContract**)

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



Description : Fournit des détails sur le coût approximatif d'une transaction standard pour afficher un **contrat Smart**, avec référence à l'adresse d'entrée. Le paramètre de sortie "verifyTransaction" nous indique si la transaction peut être effectuée "True" ou si l'adresse référencée n'a pas un solde suffisant nous donnera un "False".

balanceEther : Le solde de l'adresse référencée est livré en ethers.

trasactionBalance : Solde après que la transaction ait été effectuée.

cost_Transaction : C'est le coût de la transaction pour publier le contrat intelligent.

verifyTransaction : (balanceEther moins cost_Transaction).

gasPrice : valeur actuelle du GasPrice utilisé par les "Mineurs", qui peut varier chaque minute.

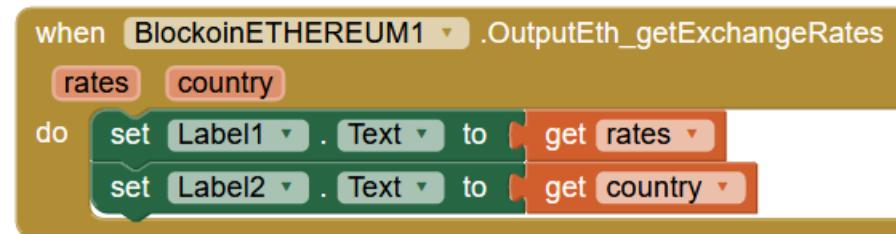
Bloc pour obtenir le prix de l'éther dans la devise du pays spécifié - (**eth_getExchangeRates**).



Paramètres d'entrée : **countryRates<String>**. Vérifiez le paramètre de sortie "pays" où il contient tous les types de devises du pays pour choisir celle qui vous convient.

Paramètres de sortie : Événement (**OutputEth_getExchangeRates**).

Sorties : **taux<Corde>, pays<Corde>** sortie au format JSON tous les taux des pays du monde.



Description : Il fournit le prix actuel d'un éther au taux de change de la monnaie du pays référencé.

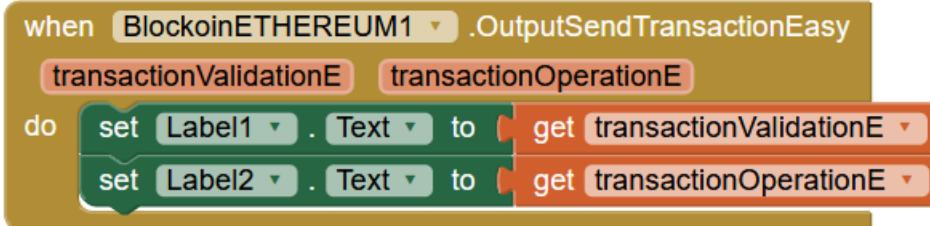
Bloc pour effectuer une transaction standard avec les paramètres d'entrée minimum - (**eth_sendTransactionEasy**).



Paramètres d'entrée : `hexPrivateKeySender<String>`, `toAddress<String>`, `valueEther<String>`.

Paramètres de sortie : Événement (`OutputSendTransactionEasy`)

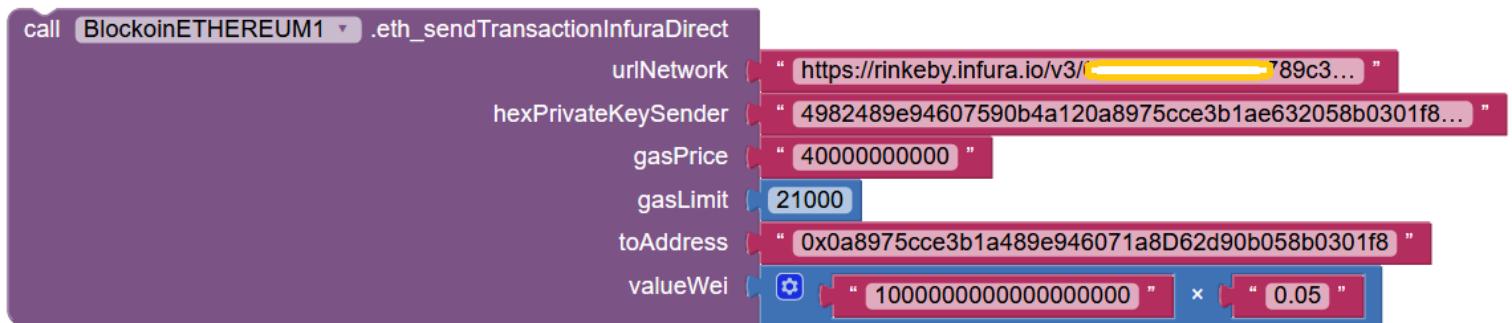
Sorties : `transactionValidation<String>` , `transactionOperation<String>`.



Description : Fonction permettant d'effectuer une transaction standard dans le réseau Ethereum, cette fonction est d'une utilité immédiate ; vous n'avez pas besoin d'avoir un compte dans INFURA et vous n'avez besoin que de 3 paramètres d'entrée, il vous suffit d'avoir un solde suffisant pour effectuer la transaction souhaitée.

Les transactions sont placées sur le réseau Ethereum directement en utilisant la bibliothèque officielle Ethereum Web3j et notre réseau Coinsolidation.org

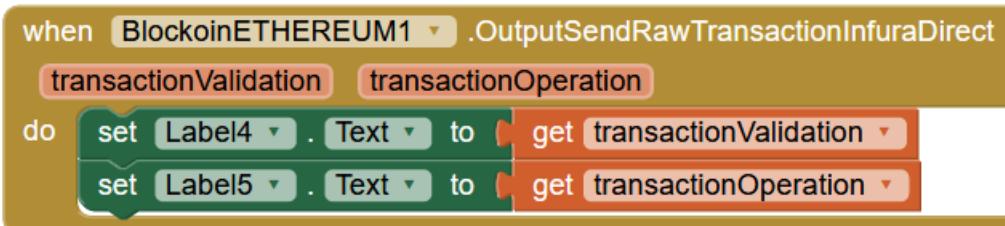
Bloc pour effectuer une transaction standard avec les paramètres d'entrée minimum - (`eth_sendTransactionInfuraDirect`).



Paramètres d'entrée : `urlNetwork<String>`, `hexPrivateKeySender<String>`, `gasPrice<String>` ,
`gasLimit<String>`, `toAddress<String>`, `valueWEI<String>`.

Paramètres de sortie : Événement (`OutputSendTransactionInfuraDirect`)

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



Description : Fonction permettant d'envoyer une transaction standard qui contient déjà la signature numérique implicite. Cette fonction est utile pour les personnes qui ont déjà une connaissance préalable des éléments d'une transaction et qui souhaitent optimiser ces paramètres en fonction de leurs besoins.

11.Calcul du coût de transaction standard et de la transaction contractuelle intelligente

Pour le calcul d'une transaction standard, 3 paramètres sont nécessaires dans le réseau Ethereum.

- 1.- **Prix de l'essence.**
- 2.- **Limite de gaz.**
- 3.- **Valeur actuelle d'un éther** (monnaie numérique Ethereum).

Prix du gaz : il est normalement exprimé en unités de GWEI (GigaWEI), ce qui correspond à 1 GWE = 1 000 000 000 000 000 wei pour 1 éther. Cette unité sert de paiement pour les systèmes qui exécutent toutes les transactions du réseau Ethereum et sont appelés "mineurs", elle est distribuée dans le monde entier. Le prix de l'essence n'est pas une valeur fixe, il est variable et peut varier d'une minute à l'autre ou d'une seconde à l'autre. Ceux qui définissent la valeur du prix du gaz sont les "mineurs" et cela dépend du degré de saturation du réseau Ethereum.

GasLimit : Cette valeur est normalement exprimée en unités de WEI et, dans une transaction standard, la valeur moyenne par défaut est de 21 000 WEI, bien qu'elle puisse être supérieure ou inférieure selon le type de transaction que vous souhaitez effectuer. Dans une transaction standard, nous utilisons la valeur de 40 000 WEI pour nous assurer que nous n'aurons pas de refus pour avoir été en dessous de la limite de gaz.

Valeur de l'éther : Cette valeur est également variable et est due à différents paramètres d'un marché financier mondial. Cette valeur peut être obtenue auprès d'entités qui ont toujours mis à jour la valeur de l'éther dans le monde entier, appelées bourses centralisées et décentralisées.

NOTE IMPORTANTE : Dans l'extension de l'échange Ethereum (EEE), nous utilisons une limite de gaz plus élevée (40.000), ceci est dû au fait qu'en cas de non atteinte du quota minimum

établi par les "mineurs", la transaction n'est pas effective, cependant le réseau Ethereum, s'il va facturer la dépense qu'il a faite dans le calcul de la transaction sans l'effectuer, Pour cette raison, certains utilisateurs n'expliquent pas pourquoi leur transaction n'est pas effectuée, mais ils devront payer un montant ou la totalité du GasLimit qui leur a été proposé lors de la demande de transaction. C'est pourquoi nous devons toujours être clairs sur les valeurs du GasLimit et du prix du gaz.

Le prix du gaz peut être consulté avec la fonction **eth_GetRatesGasStation** et le GasLimit pour les transactions standard doit être supérieur à 21.000 WEI et les transactions pour publier et/ou exécuter le contrat Smart doivent être au moins de 500.000 WEI.

Coût de transaction standard.

Il est défini par la formule suivante :

Coût = (GasLimit x (Prix du gaz / (1.000.000.000.000.000))) x Valeur de l'éther.

Exemple :

Supposons les valeurs suivantes :

GasLimit = 40 000, Prix du gaz = 45 GWEI, Valeur de l'éther = 406 \$.

Coût = (40.000 x (45.000.000.000 / (1.000.000.000.000.000))) x 406

Coût de transaction standard = 0,0018 éther x 406 USD = 0,73 USD

Dans le cas d'une transaction de contrat Smart, il est nécessaire de savoir quel type de contrat Smart sera publié puisque le coût est directement proportionnel à la charge de travail que les "mineurs" devront effectuer pour traiter le contrat Smart, en d'autres termes, la quantité de traitement dans les systèmes informatiques que les "mineurs" manipulent est plus simple.

Dans le cas du contrat Smart, une valeur par défaut serait de commencer le GasLimit avec une valeur de 500 000 WEI.

Un point important à prendre en compte est que lorsqu'on propose un GasLimit, les "mineurs" ne prendront pas nécessairement tout le montant proposé, c'est-à-dire que lorsqu'on envoie une transaction, les "mineurs" calculent l'effort de calcul et retirent ce qui sera retiré du GasLimit, qui peut être inférieur ou égal au GasLimit par défaut de 21.000 WEI offert dans certains cas.

Toutes les transactions pourront être consultées sur le site www.etherscan.io où l'on peut consulter le détail de chaque transaction.

Exemple d'une transaction standard, où la transaction a été envoyée avec une limite de gaz de 40.000 WEI, cependant les "mineurs" lors du calcul du coût de traitement ne prendraient que 52,5%, c'est-à-dire la valeur par défaut qui est de 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 is currently in block 11234630 with 2 block confirmations. It was timestamped 52 seconds ago on Nov-11-2020 at 06:17:24 AM UTC, and it was confirmed within 38 seconds. The transaction originated from address 0x4b7355fd05be6dac458b004f54e12d6527a54a58 and was sent to address 0x5d2acdb34c279aa6d1e94a77f7b18ab938fb2bb. The value transferred was 0.001084598698481562 Ether (\$0.50). The transaction fee was 0.000672 Ether (\$0.31), which is 52.5% of the proposed gas limit. The gas price was 0.000000032 Ether (32 Gwei), and the gas limit was set at 40,000. The actual gas used was 21,000, which is 52.5% of the proposed limit. A yellow arrow points from the text "Opération ou validation de la transaction" to the transaction hash. Another yellow arrow points from the box "Coût de la transaction en Ether :" to the value "21 000 x 0,000000032 = 0,000672 Ether (0,31 USD)". A third yellow arrow points from the box "Limite proposée pour le gaz : 40" to the gas limit value "40,000". A fourth yellow arrow points from the box "Limite de gaz réelle utilisée dans la transaction : 21 000 WEI" to the gas used value "21,000 (52.5%)".

Field	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%)	

En revenant à la transaction du contrat Smart et en prenant le GasLimit de 500 000 WEI, nous obtenons le coût suivant si nous utilisons tout.

Coût de transaction du contrat intelligent.

Il est défini par la formule suivante :

Coût = (GasLimit x (Prix du gaz / (1.000.000.000.000.000))) x Valeur de l'éther.

Exemple :

Supposons les valeurs suivantes :

GasLimit = 500 000, **Prix du gaz** = 45 GWEI, **Valeur de l'éther** = 406 \$.

Coût = (500.000 x (45.000.000.000 / (1.000.000.000.000.000))) x 406

Coût de transaction publier contrat Smart = 0,0225 éther x 406 USD = 9 135 USD

Toutes les transactions peuvent être consultées sur le site www.etherscan.io

REMARQUE : Pour obtenir le coût total de la transaction, vous devez les additionner :

Coût total de la transaction = coût du *réseau Ethereum + frais de Coinsolidation.org*

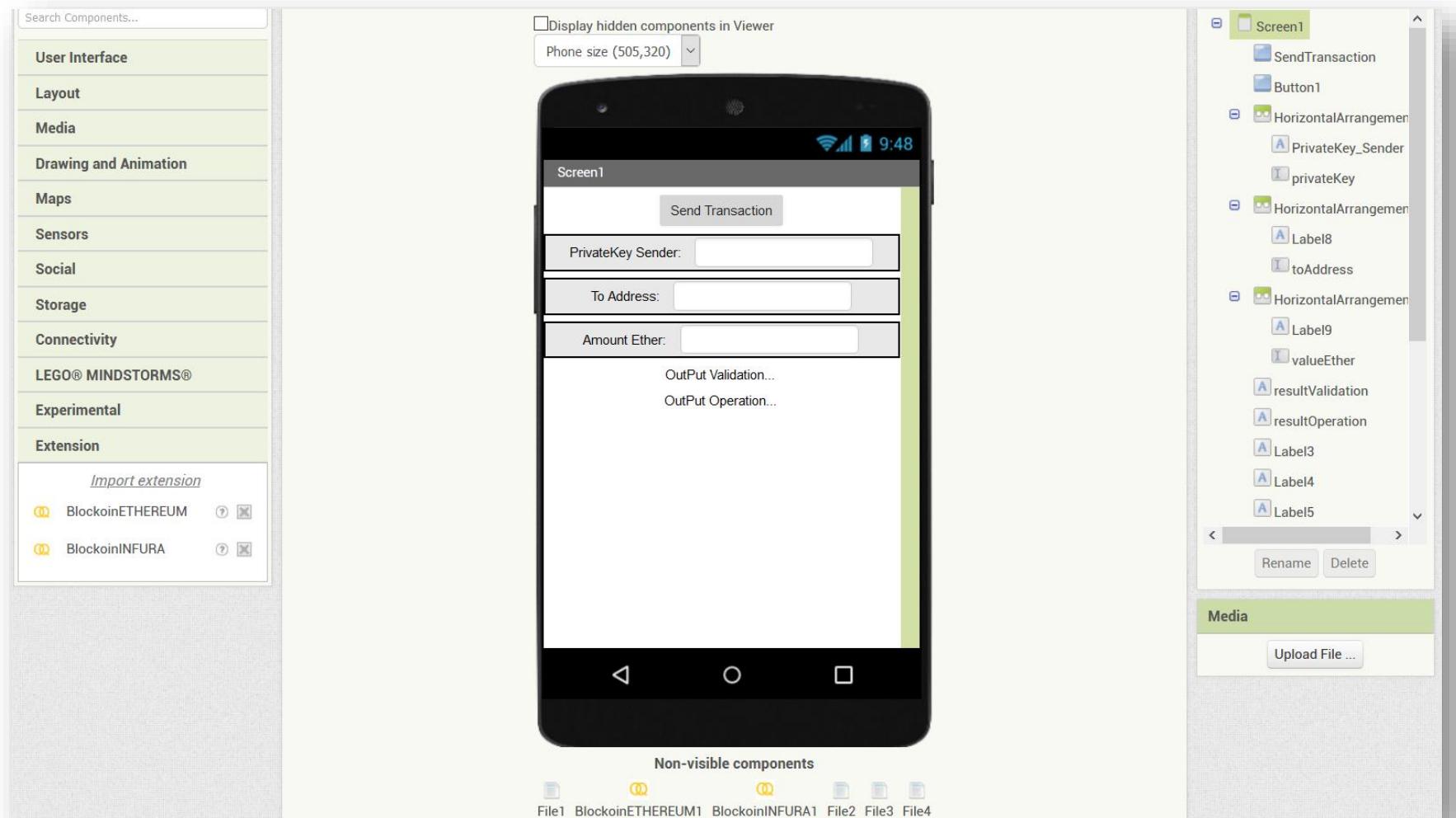
12. Frais de Coinsolidation.org

Transaction standard : 0,5 cents USD + coût du réseau Ethereum.

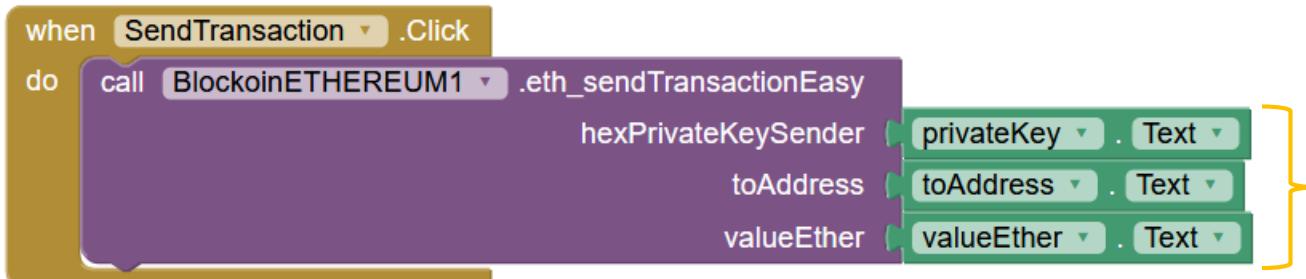
Transaction publier et/ou exécuter un contrat Smart : 15 USD + coût du réseau Ethereum.

13. Créez votre application Android (Exchange) en 15 minutes.

Design in App Inventor (écran). - 5 minutes.



Blocs de fonctions (`eth_SendTransactionEasy`) et événements (`OutPutSendTransactionEasy`) - 5 minutes

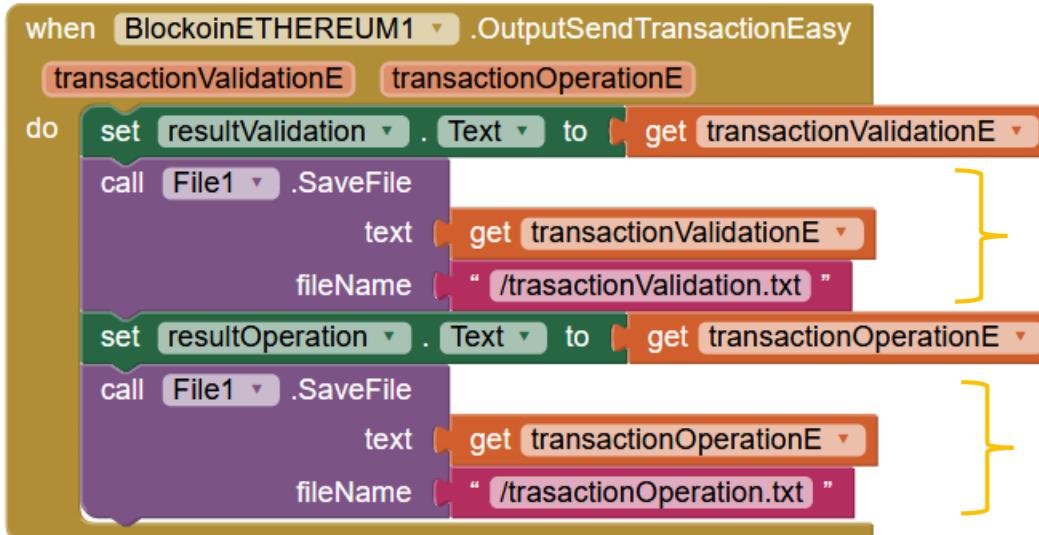


Données d'entrée :

PrivateKey : Clé primaire de l'adresse de l'expéditeur.

toAddress : Adresse hexadécimale du destinataire.

valeurEther : Indiquez la quantité d'éther qui sera envoyée.



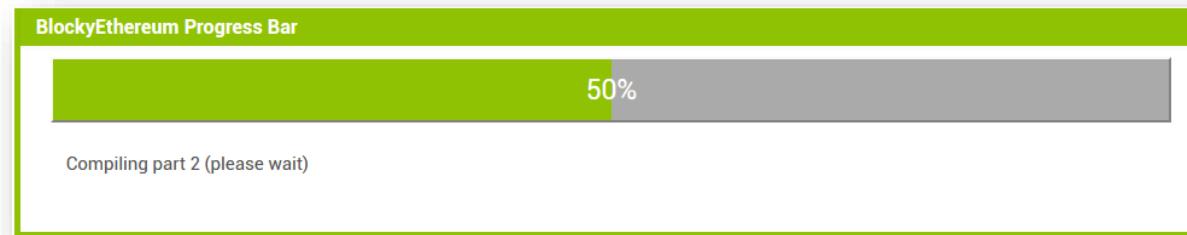
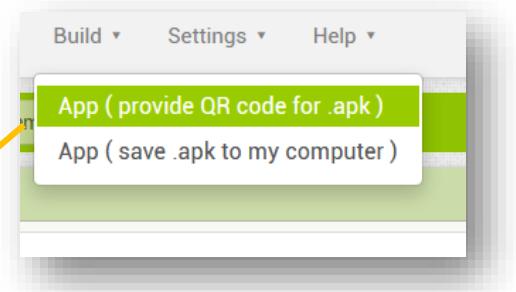
Enregistrez les résultats dans des fichiers texte
:

Fichier de fonction1 : Fichier
trasactionValidation.txt

Enregistrez les résultats dans des fichiers texte
:

Fonction File2 : Fichier **trasactionValidation.txt**

Nous compilons, générons le fichier APK pour l'installer sur l'appareil Android. - 5 minutes



REMARQUE : Lorsque la transaction est exécutée, il faut environ 6 à 8 secondes pour relâcher le bouton "Envoyer la transaction". En raison du temps de connexion avec le réseau Ethereum.

14. Token CoinSolidation.

Token Coinsolidation est un projet comportant trois grandes lignes directrices.

Imaginez avoir votre propre crypto-monnaie Token pour développer votre entreprise. En utilisant Coinsolidation, vous pouvez avoir les choses faciles et simples.

La première est de créer le premier réseau d'extensions basé sur la méthodologie de programmation visuelle Blockly, dans lequel par son utilisation facile et intuitive peut être utilisé par toute personne sans connaissance préalable de la programmation, les extensions qui peuvent être consultées dans notre feuille de route (Livre blanc) sont dirigées vers deux secteurs fondamentaux de l'économie mondiale, le secteur des crypto-monnaies et/ou des jetons et le secteur des monnaies (fiat) ou d'usage courant dans le monde comme le dollar USD, les euros UE, les livres ou toute autre monnaie en usage.

La deuxième ligne directrice du projet CoinSolidation consiste à créer un nouvel algorithme pour consolider les adresses des cryptomontages actuels et futurs. Nous développons un nouvel algorithme de modèle pour créer une adresse qui consolide différentes adresses de différentes chaînes de blocs en une adresse universelle ; voir notre (Livre blanc) à www.CoinSolidation.org ou <https://github.com/coinsolidation/whitepaper>

La troisième ligne directrice consiste à appliquer la technologie de l'informatique quantique à la sécurité de l'environnement CoinSolidation, ce qui sera fait avec les extensions déjà développées des algorithmes de sécurité QRNG (Quantum Random Number Generator) et PQC (Post-Quantum Computing). Elles se trouvent dans le dépôt officiel des extensions sur le site Github, <https://github.com/coinsolidation>

Caractéristiques générales du CryptoToken CoinSolidation

Nom : CoinSolidation

Symbole : CUAG

Type : NFT

Pays de lancement : Estonie

Site officiel : www.Coinsolidation.org

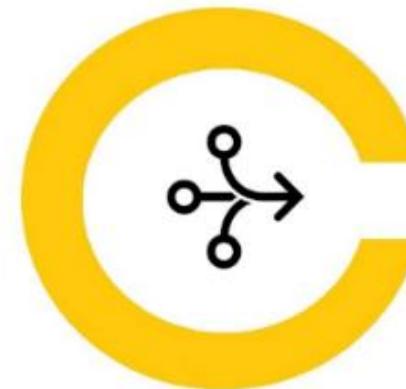
Société : Coinsolidation International.

Date de lancement : 12 décembre 2020

Créé par : Lugu Samaya.

Algorithme de consensus : Preuve de Quantum.

Algorithme d'adresse : Adresse universelle consolidée.



15. Licences et utilisation des logiciels.

Licences, conditions d'utilisation, voir www.coinsolidation.org ou écrire à info@coinsolidation.org