



# smartDirectory (mission smart2)

Document de liaison équipe en date du 29 nov. 2024

## Livrables envisageables

Juillet 2024	Proposition du smartDirectory
	Spécifications générales du smartDirectory
	Encapsulation des fonctions blockchain pour l'app smartphone
	Mise à disposition des API sur GitLab
	Spécifications des smartTokens
	Proposition de l'interface smartphone
	smartDirectory avec test python
	fonction spécifique smartDirectory pour App smartphone
	Application smartphone

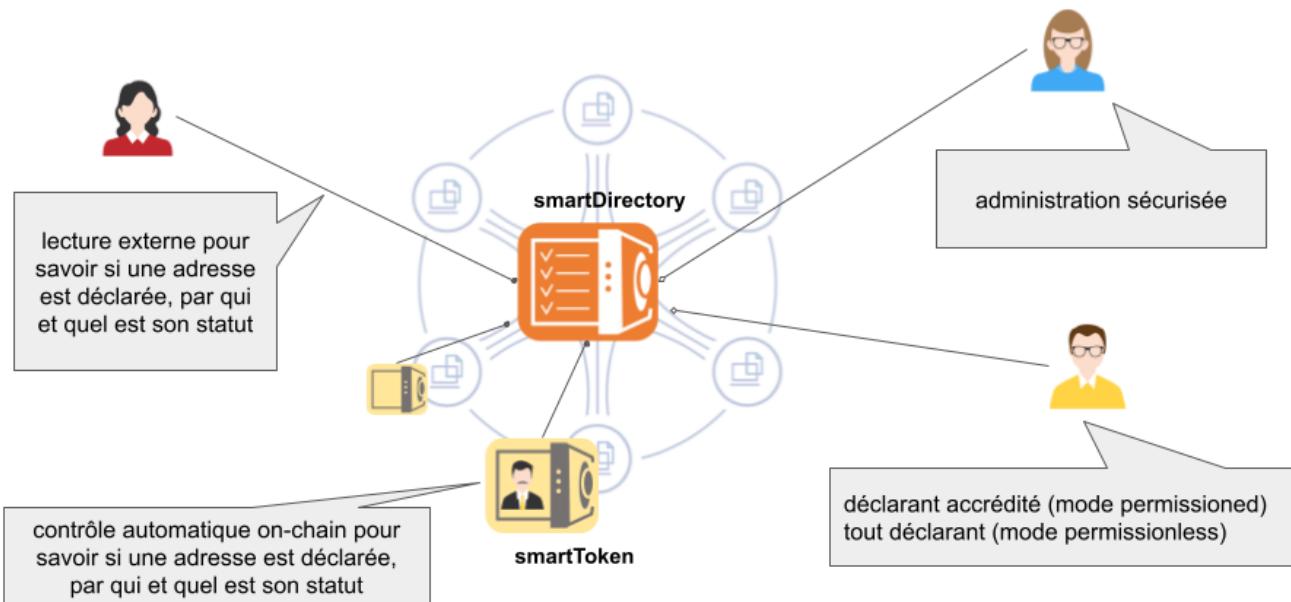


## Rappel des principes initiaux

[Voir le détail complet de la proposition](#)

Avec l'explosion anticipée du nombre de smartContracts due à la tokenisation de l'économie et l'account abstraction, nous pensons que la première étape doit être la facilité de création de listes de références pour des accès privatifs ou publics, hors de la blockchain ou en interne de la blockchain.

### Une liste sécurisée accessible



733

© BPCE2024- Licence MIT

Qaxh.io

En synthèse, nous identifions les besoins génériques suivants :

- une facilité de déploiement de nouveaux smartContracts,
- une capacité de contrôle de la validité de ces smartContracts par un utilisateur externe à la blockchain,
- une capacité de contrôle de la validité de ces smartContracts par un autre smartContract,
- une nécessité d'identifier des écosystèmes de smartContracts pour en faire une analyse,
- une capacité de maintenir à jour ces écosystèmes.



## Cas d'usages

### Les cas d'usages du smartDirectory

- **usage de régulation (administré)**
  - un régulateur peut imposer aux entités régulées de déclarer tous les smartContracts qu'elles émettent
  - Au préalable le régulateur enregistre l'adresse de l'entité comme "déclarant"
  - de plus il peut imposer que l'entité expose une URI ou une API permettant l'accès à plus d'informations. Il fait inscrire par le déclarant l'URI d'accès
- **usage déclaratif par un déployeur (ouvert)**
  - une entité, le déclarant, qui émet des smartContracts les référence lors du déploiement
  - l'entité peut aussi mettre à jour l'URI présente dans la structure du déclarant
- **usage privatif :**
  - l'administrateur d'une entité (le déclarant) liste des smartContracts en y affectant un projectID
  - ceci permet à une application pour des utilisateurs d'avoir toujours la liste des smartContracts à jour en ne connaissant que la seule adresse du smartDirectory et le projectID

746

© BPCE2024- Licence MIT

 Qaxh.io

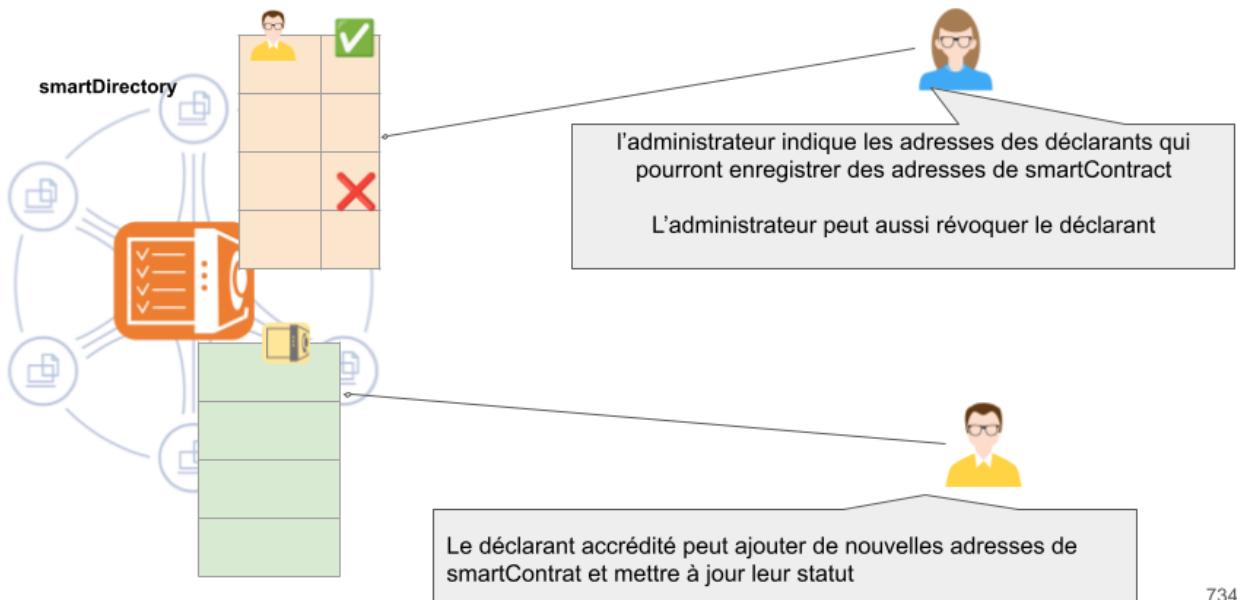
Le smartDirectory possède un paramètre majeur (AdminCode) non modifiable qui permet de choisir lors du déploiement entre :

- la gestion administrée ("parentsAuthorized")
- la gestion ouverte ('selfDeclaration')



## Le mode administré (permissioned)

### gestion administrée (permissioned)



© BPCE2024- Licence MIT

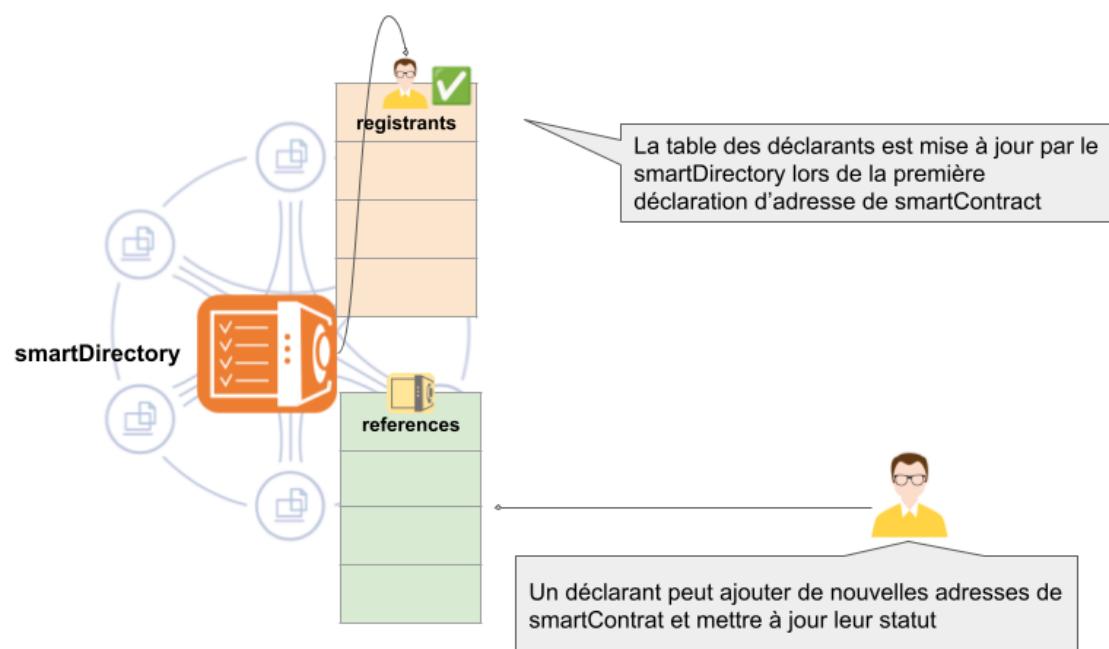
734

Qaxh.io

Dans le cas de la gestion administré, l'administrateur peut indiquer 2 adresses dites “parent”.

## Le mode ouvert (permissionless)

### gestion ouverte (permissionless)



© BPCE2024- Licence MIT

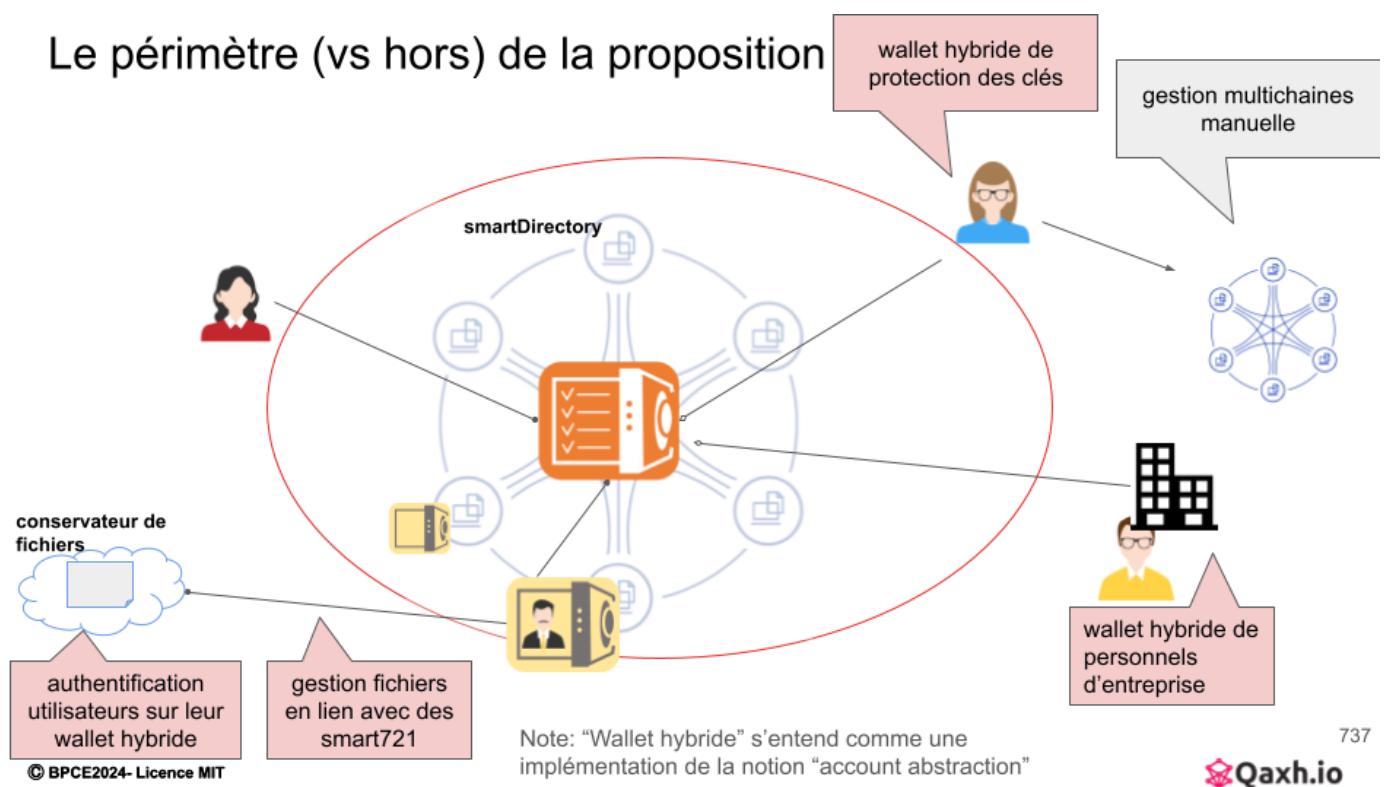
735

Qaxh.io



## Le périmètre du projet

### Le périmètre (vs hors) de la proposition



## Les points de vigilances

L'utilisation d'un smartDirectory implique cependant des points de vigilance complémentaires :

- **Assurer un niveau de sécurité et de résilience** du système d'autorisation et de vérification des informations enregistrées.
  - L'adresse du déclarant est importante mais bien moindre que celle de l'administrateur. La **validité et la permanence de l'adresse de l'administrateur est essentielle** et notre réponse passe par l'utilisation d'un wallet hybride décrit plus loin. Ce wallet hybride peut être personnel ou d'entreprise. De la même manière, un déclarant peut mettre en œuvre un wallet hybride.
  - L'autre point réside dans la capacité de l'administrateur à enregistrer de nouveaux "déclarants" et de les désactiver le cas échéant.
- **Considérer l'utilisation d'un système de stockage décentralisé** au cas où il serait nécessaire de stocker des données plus importantes.
  - Le smartDirectory ne gère que des adresses et, de manière limitée, des URI comme point d'entrée sur le Web2.



- En cas de nécessité de réaliser un écosystème plus large, il est possible de coupler un smartToken de type **ERC721** qui va lister des fichiers identifiés en tant que NFT et gérer les autorisations d'accès vers le stockage. Les fichiers sont potentiellement sur IPFS mais de manière plus pragmatique dans un monde professionnel, sur un “conservateur de fichiers”, c'est-à-dire un serveur de fichiers qui ne délivre des fichiers qu'en fonction des autorisations lues sur le NFT correspondant. De plus, l'authentification des requêtes vers le conservateur de fichiers se fait au travers du wallet hybride pour plus de sécurité : ainsi le conservateur de fichiers opère sans nécessité de lui adjoindre des fonctions d'administration.
- Architecture et fonctionnement dans un **écosystème multichain**.
  - La gestion multichain se fait manuellement par l'administrateur au travers de la création de plusieurs smartDirectory.
  - Une deuxième étape pourrait aborder la faisabilité d'un smartToken d'une chaîne pour lire un smartDirectory d'une autre chaîne. L'ajout d'un paramètre “chainID” en plus de “contractVersion” et de “contractType” serait un minimum. L'utilisation des informations “on chain” par des smartcontracts nécessiterait une passerelle inter-chain.
- Évaluer l'architecture de la solution pour un système de **blockchains permissionnées** (Consortium).
  - La proposition s'accorde bien sur une architecture permissionnée car, même dans un cadre de consortium, il est nécessaire de permettre à ses membres d'exposer l'état de leur smartcontracts (actif, inactif, version ...). L'utilisation des informations du smartDirectory par les smartcontracts permet de restreindre les accès à un sous-ensemble des membres.
- Utilisation éventuelle d'un système d'**autorisation multipartite** (DAO, Multisig).
  - Ceci est possible avec un wallet hybride. Nous pensons qu'il reste préférable que les fonctions du système d'autorisation multipartite restent externes et non intégrées au smartDirectory.



## Zoom sur le wallet hybride (hors proposition)

Hybride signifie ici que le wallet est scindé en 2 parties: une partie sur un moyen d'accès (mobile ou web) et une partie sur un smart contract.

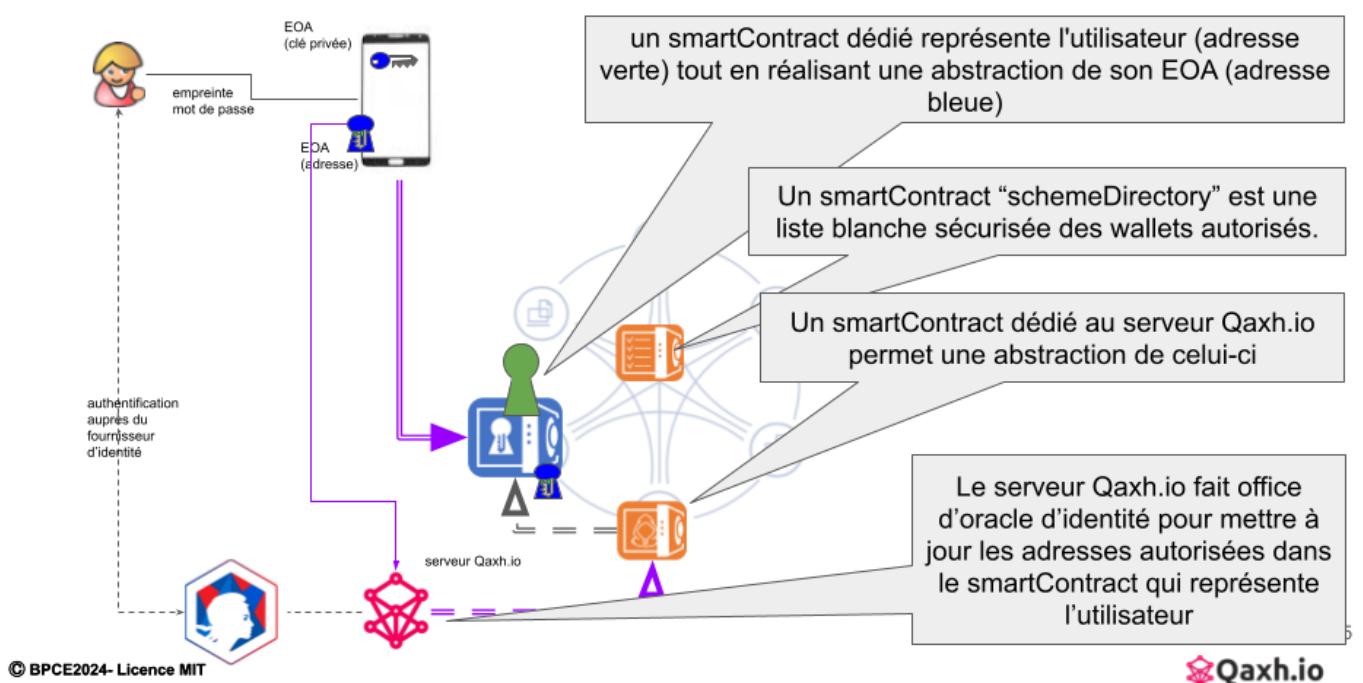
Le smart contract représente la personne ou la société sur la blockchain et son adresse ne change pas.

Les clés d'accès à ce contrat ne sont attribuées qu'après validation auprès d'un fournisseur d'identité numérique. La réattribution en cas de perte suit le même mécanisme.

L'utilisation d'un wallet hybride permet donc la permanence et la conservation des assets même en cas de perte des clés d'accès. Ceci répond aux interrogations sur la sécurité et la résilience.

Le schéma suivant présente sommairement son fonctionnement.

## Zoom sur le wallet hybride (account abstraction)



Le wallet hybride permet de décliner ensuite un wallet pour les administrateurs et collaborateurs d'une entreprise.



# Présentation du smartDirectory

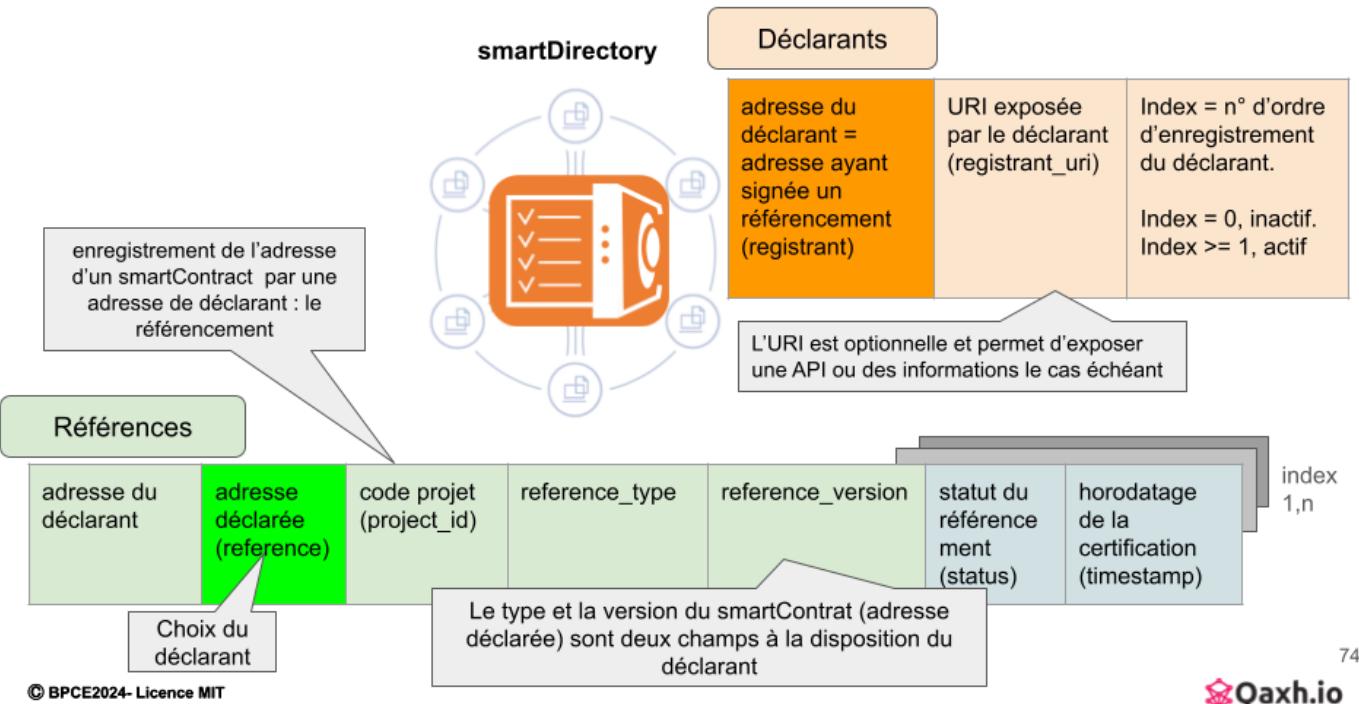
Le smartDirectory est un smartContract qui va faire office de référence et de sommaire pour des adresses de la blockchain utilisée.

## Définitions et nommage

- **smartDirectoryAddress** : adresse du smart contract “smartDirectory”.
- **déclarant (“registrant”)** : acteur externe identifié par son adresse et à même d’enregistrer des “references”
- **registrantAddress** : adresse du déclarant.
- **Reference**: adresse de smart contract déclarée et conservée dans la liste des “references” du smartDirectory.
- **projectId** : code projet du déclarant associé à l’adresse du smartContract déclaré.

## Les structures du smartDirectory

### 2 structures dans le smartDirectory



## Table des références

Cette table contient tous les référencements sur la base du “record” suivant :

- **registrantAddress** : adresse du déclarant (EOA ou smartContract), utilisation possible comme index de lecture de la table.
- **referenceAddress** : adresse du smartContract déclaré (référence). Cette adresse peut aussi être une EOA. index de lecture.

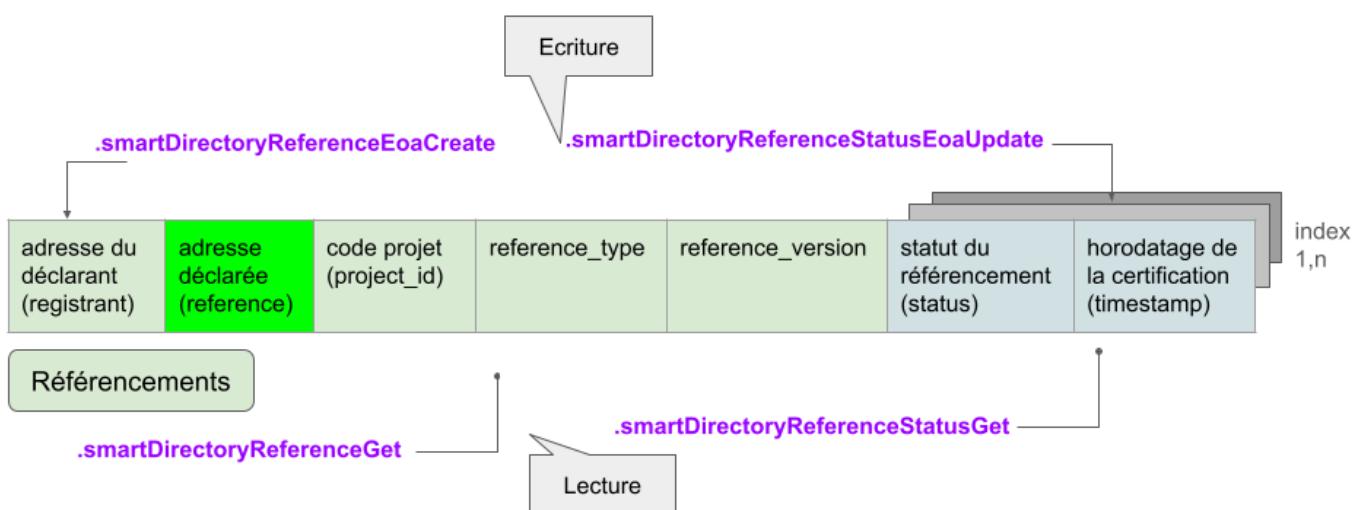


- **projectId** : une chaîne de caractères représentant le code projet.

Variables complémentaires associées à ce record, écrites une fois et non modifiables :

- **referenceType** : chaîne de caractères.
- **referenceVersion** : chaîne de caractères.
- **referenceStatus** : sous-liste qui peut être parcourue (premier index: 1) et comprenant
  - status : le statut du référencement. Champ string, modifiable uniquement par le déclarant. La sémantique de ce statut est à la main du déclarant, une signification explicite est conseillée (par ex: en production, en suspens, abandonné, etc...) ou une URI d'explication.
  - timestamp : horodatage de l'écriture lors de la création ou du changement de statut.

## Fonctions sur la table des référencements



743

© BPCE2024- Licence MIT

Qaxh.io

**.smartDirectoryReferenceEoaCreate (smartDirectoryAddress, referenceAddress, projectId, referenceType, referenceVersion, status)**

Cette fonction permet la création d'un nouvel enregistrement dans la table des référencements.

- L'adresse du déclarant n'est pas explicitement dans les paramètres car c'est l'adresse qui signe la transaction de création
- l'horodatage (timestamp) est mis automatiquement lors de la création du record.
- le statut (status) est une chaîne de caractère libre qui peut être sous forme d'un "URI"
- Valeur de retour: retour un tx\_hash pour vérifier le minage côté client.

S'il existe déjà un record de l'adresse déclarée (référence) par ce déclarant alors faire un "revert".



Il existe 2 stratégies d'utilisation de cette fonction suivant le paramétrage du smartDirectory au moment de sa création :

- Mode permissionné (“parentAuthorized”) : enregistrement préalable des déclarants<sup>43</sup> par les adresses parent du smartDirectory (“AdminCode = 0”):
  - Si le déclarant n'est pas présent lorsqu'il déclare une référence -> rejet de la référence.
  - Si le déclarant est bien présent dans la table des déclarants lorsqu'il déclare une référence -> ajout de la référence.
- Mode permissionless (“self Declaration”) : enregistrement en simultané du déclarant (“AdminCode = 1”):
  - Si le déclarant n'est pas présent -> ajout dans la table des déclarants et ajout de la référence.
  - Si le déclarant est présent -> ajout de la référence.



La fonction solidity appelée est “createReference” :

```
@SimpleFunction(description = "create a new smartContract reference in the SmartDirectory")
public String smartDirectoryReferenceEoaCreate (String smartDirectoryAddress, String
referenceAddress,
                                                String projectId, String referenceType, String referenceVersion,
                                                String status) {

    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,
transactionManager,
        new DefaultGasProvider());

    String tx_hash;
    try {
        tx_hash = doTransaction(
            folderContract.createReference(
                referenceAddress,
                projectId,
                referenceType,
                referenceVersion,
                status
            ),
            "createReference");
    } catch (Exception e) {
        String message = "Error smartDirectoryReferenceEoaCreate: " + e.getMessage();
        android.util.Log.d(LOG_TAG, message);
    }
}
```

<sup>43</sup> La déclaration est réservée aux adresses Parents

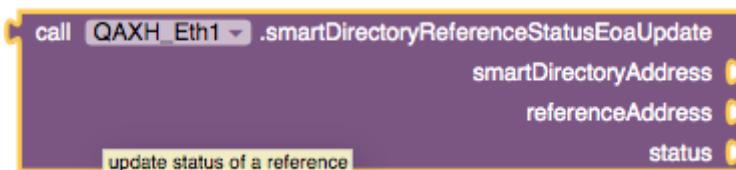


```
    tx_hash = message;
}
return tx_hash;
}
```

### .smartDirectoryReferenceStatusEoaUpdate (**smartDirectoryAddress**, **referenceAddress**, **status**)

Cette fonction permet d'ajouter un nouveau *statut* et le *timestamp* associé dans la sous-liste **referenceStatus** pour la **referenceAddress** passée en paramètre :

- Si la **referenceAddress** n'existe pas faire un “revert”.
- Si la **referenceAddress** existe dans la table des déclarants, la transaction d’update est signée par l’adresse du déclarant et un nouvel enregistrement est fait dans la sous-liste **referenceStatus** avec le nouveau statut et l’horodatage associé.



La fonction solidity appelée est “`updateReferenceStatus`” du smartContract “`SmartDirectory.sol`” :

```
@SimpleFunction(description = "update status of a reference")
public String smartDirectoryReferenceStatusEoaUpdate (String smartDirectoryAddress, String
referenceAddress,
String status) {

    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,
transactionManager,
    new DefaultGasProvider());

    String tx_hash;
    try {
        tx_hash = doTransaction(
            folderContract.updateReferenceStatus(referenceAddress, status),
            "updateReferenceStatus");
    } catch (Exception e) {
        String message = "Error smartDirectoryReferenceStatusEoaUpdate: " + e.getMessage();
        android.util.Log.d(LOG_TAG, message);
        tx_hash = message;
    }
    return tx_hash;
}
```



## .smartDirectoryReferenceGet (smartDirectoryAddress, referenceAddress)

Cette fonction permet la lecture d'une "reference" en connaissant uniquement l'adresse du smartContract référencé (la referenceAddress). C'est la fonction principale d'utilisation du smartDirectory.



La fonction retourne un dictionnaire contenant :

- **registrantAddress** : l'adresse du déclarant.
- **projectId** : l'identification du projet auquel appartient le smartContract référencé.
- **referenceType** : le type de smartContract (champ libre à la main du déclarant).
- **referenceversion** : la version du smartContract (champ libre à la main du déclarant).
- **latestStatus** : le dernier statut déclaré dans la sous-liste referenceStatus.
- **latestTimeStamp** : le timestamp associé au dernier statut déclaré dans la sous-liste referenceStatus.
- **lastStatusIndex** : l'index de ce dernier statut.

Si la référence n'existe pas, le dictionnaire retourné est vide.

La fonction appelle les fonctions "getReference" et "getReferenceLastStatusIndex" du smartContract "SmartDirectory.sol" :

```
@SimpleFunction(description = "get smartContract reference details")
public Object smartDirectoryReferenceGet(String smartDirectoryAddress, String referenceAddress) {

    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,
transactionManager,
        new CustomGasProvider());

    Tuple7<String, String, String, String, String, BigInteger> reference;
    BigInteger lastStatusIndex;
    Map<String, String> result_dict = new HashMap<>();

    try {
        reference = folderContract.getReference(referenceAddress).send();
        lastStatusIndex = folderContract.getReferenceLastStatusIndex(referenceAddress).send();

        result_dict.put("registrantAddress", reference.component1());
        result_dict.put("referenceAddress", reference.component2());
        result_dict.put("projectId", reference.component3());
        result_dict.put("referenceType", reference.component4());
        result_dict.put("referenceVersion", reference.component5());
        result_dict.put("lastStatus", reference.component6());
        result_dict.put("lastTimestamp", reference.component7().toString());
        result_dict.put("lastStatusIndex", lastStatusIndex.toString());
    } catch (Exception e) {
        result_dict.put("Error smartDirectoryReferenceGet", e.getMessage());
    }
}
```



```
    }
    return result_dict;
}
```

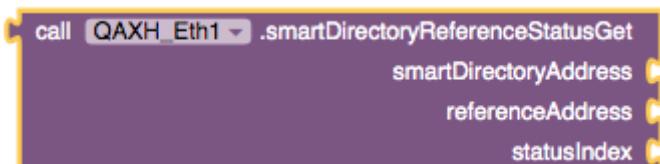
## .smartDirectoryReferenceStatusGet (smartDirectoryAddress, referenceAddress, statusIndex)

Cette fonction permet la lecture d'un index de la sous-liste des statuts.

La fonction .smartDirectoryReferenceGet renvoie le triplet (dernier statut, dernier timestamp, dernier index) ce qui devrait subvenir à la plupart des besoins. Cependant, en ayant le dernier index, il devient facile de faire une boucle dans la requête d'interrogation pour obtenir si besoin l'historique des statuts.

La fonction .smartDirectoryReferenceStatusGet retourne donc :

- le statut à l'index de la requête
- le timestamp à l'index de la requête



La fonction solidity appelée est “getReferenceStatus” du smartContract “SmartDirectory.sol” :

```
@SimpleFunction(description = "return reference status and timestamp at a given index")
public Object smartDirectoryReferenceStatusGet(String smartDirectoryAddress, String
referenceAddress,
int statusIndex) {

    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,
transactionManager,
    new CustomGasProvider());

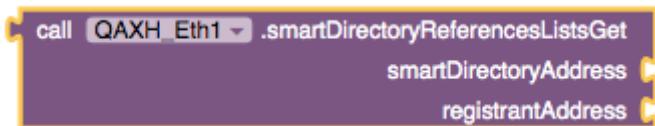
    Tuple2<String, BigInteger> results_raw;
    Map<String, String> result_dict = new HashMap<>();

    try {
        results_raw = folderContract.getReferenceStatus(referenceAddress,
BigInteger.valueOf(statusIndex)).send();
        result_dict.put("status", results_raw.component1());
        result_dict.put("timestamp", results_raw.component2().toString());
    } catch (Exception e) {
        result_dict.put("error", "smartDirectoryReferenceStatusGet " + e.getMessage());
    }
    return result_dict;
}
```

## .smartDirectoryReferencesListsGet (smartDirectoryAddress, registrantAddress)



Cette fonction donne les deux listes (liste des adresses déclarées, liste des codes projets associés) pour l'adresse de déclarant passée en paramètre.



! Comme à chaque adresse déclarée (reference) correspond un code projet (projectId), ces deux listes ont la même taille et sont ordonnées pour que l'index de la liste des adresses déclarées corresponde à l'index du code projet.

La fonction solidity appelée est “`getReferencesLists`” du smartContract “`SmartDirectory.sol`” :

```
@SimpleFunction(description = "return a list of reference/projectId for a given registrant address")
public Object smartDirectoryReferencesListsGet(String smartDirectoryAddress,
                                              String registrantAddress) {

    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,
transactionManager,
    new CustomGasProvider());

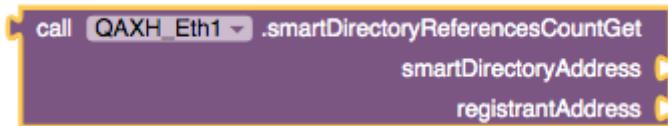
    Tuple2<List<String>, List<String>> results_raw;
    Map<String, List<String>> result_dict = new HashMap<String, List<String>>();

    try {
        results_raw = folderContract.getReferencesLists(registrantAddress).send();
        List<String> listAddresses = new ArrayList<String>();
        List<String> listProjectIDs = new ArrayList<String>();
        for (int i = 0; i < results_raw.component1().size(); i++) {
            listAddresses.add(results_raw.component1().get(i));
            listProjectIDs.add(results_raw.component2().get(i));
        }
        result_dict.put("referenceList", listAddresses);
        result_dict.put("projectIdList", listProjectIDs);
    } catch (Exception e) {
        List<String> listErrors = new ArrayList<String>();
        listErrors.add("smartDirectoryReferencesListsGet: " + e.getMessage());
        result_dict.put("Error", listErrors);
    }
    return result_dict;
}
```



## .smartDirectoryReferencesCountGet (smartDirectoryAddress, registrantAddress)

Cette fonction donne la taille de la liste précédente. Cela peut être utile pour modifier l'UX de l'utilisateur en cas de taille importante.



La fonction solidity appelée est “`getRegistrantReferencesCount`” du smartContract “`SmartDirectory.sol`” :

```
@SimpleFunction(description = "get registrant references count")
public int smartDirectoryReferencesCountGet(String smartDirectoryAddress, String registrantAddress)
{
    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,
transactionManager,
    new DefaultGasProvider());

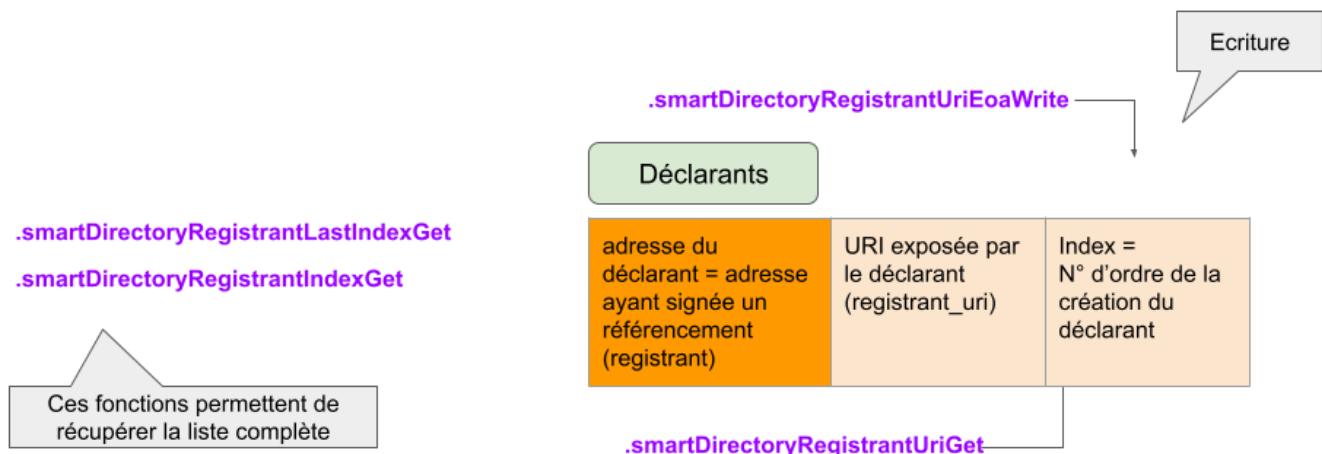
    BigInteger result;
    int resultInt;

    try {
        result = folderContract.getRegistrantReferencesCount(registrantAddress).send();
        resultInt = result.intValue();
    } catch (Exception e) {
        String message = "Error smartDirectoryReferencesCountGet: " + e.getMessage();
        android.util.Log.d(LOG_TAG, message);
        resultInt = -1;
    }
    return resultInt;
}
```



## Table des déclarants (registrants Table)

### Fonctions sur la table des déclarants



© BPCE2024- Licence MIT

744



Une table des déclarants est créée et mise à jour soit explicitement pour chaque nouveau déclarant soit le cas échéant à chaque nouvelle création de record dans la table des référencements, ceci en fonction de la stratégie d'utilisation (voir la fonction [.smartDirectoryReferenceEoaCreate](#))

La stratégie d'utilisation est donnée par le paramétrage de l' "AdminCode" du smartDirectory :

- **Mode permissionné ("parentAuthorized")** : l' "**AdminCode**" = 0. Les adresses des déclarants doivent être préalablement enregistrées par un des parents (parentAddress1 ou 2). Si les parents veulent être eux-mêmes des déclarants, ils doivent s'auto-déclarer dans la table.
- **Mode permissionless ("selfDeclaration")** : l' "**AdminCode**" = 1. Toutes les adresses se déclarent automatiquement lors de la première déclaration.

Cette table des déclarants est constituée de 3 éléments :

1. l'adresse d'un déclarant,
2. une chaîne de caractère à la disposition du déclarant afin d'y déposer une URI d'information. Cette chaîne de caractère est mise à jour dans un deuxième temps par le déclarant.
3. Un index, représentant le n°d'ordre de l'enregistrement du déclarant dans la table.



## .smartDirectoryRegistrantEoaCreate (smartDirectoryAddress, registrantAddress)

Cette fonction permet la création d'un nouveau déclarant par une adresse parent.

L'usage de cette fonction est conditionné :

- Le SmartDirectory doit être dans un état activé (ActivationCode.active). Cela empêche des modifications si le smartDirectory est déployé mais inactif.
- Cette fonction n'est disponible que dans le mode "parentAuthorized" (AdminCode=0) : seuls les administrateurs (parentAddress1 ou 2) peuvent créer un registrant.
- L'adresse donnée (registrantAddress) ne doit pas déjà être enregistrée.

```
call QAXH_Eth1 .smartDirectoryRegistrantEoaCreate
    smartDirectoryAddress
    registrantAddress
```

La fonction solidity appelée est "createRegistrant" du smartContract "SmartDirectory.sol" :

```
@SimpleFunction(description = "create a registrant address in registrants data structure, only parents")
public String smartDirectoryRegistrantEoaCreate(String smartDirectoryAddress, String
registrantAddress) {

    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,
transactionManager,
        new DefaultGasProvider());

    String tx_hash;
    try {
        tx_hash = doTransaction(
            folderContract.createRegistrant(registrantAddress),
            "createRegistrant");
    } catch (Exception e) {
        String message = "Error smartDirectoryRegistrantEoaCreate: " + e.getMessage();
        android.util.Log.d(LOG_TAG, message);
        tx_hash = message;
    }
    return tx_hash;
}
```



## .smartDirectoryRegistrantEoaDelete (smartDirectoryAddress, registrantAddress)

Cette fonction permet de désactiver un déclarant, charge au vérificateur de voir s'il continue de faire confiance aux références même si le déclarant n'est plus autorisé.



La désactivation d'un déclarant se traduit par la mise à 0 de l'index associé à son adresse dans la table des déclarants (et non par l'effacement de son adresse dans la table) :

- Index = 0 => déclarant enregistré mais invalidé (ne peut plus créer de référence).
- Index >= 1 => déclarant enregistré et valide (peut créer des références).

L'usage de cette fonction est conditionné :

- Le SmartDirectory doit être dans un état activé (ActivationCode.active).
- L'index de l'adresse donnée en paramètre est > à 0.
- Cette fonction n'est disponible que dans le mode "parentAuthorized" (AdminCode=0) : seuls les administrateurs (parentAddress1 ou 2) peuvent l'utiliser et invalider un déclarant (registrant).

La fonction solidity appelée est "delRegistrant" du smartContract "SmartDirectory.sol" :

```
@SimpleFunction(description = "del a registrant address in registrants data structure, only parents")
public String smartDirectoryRegistrantEoaDel(String smartDirectoryAddress, String registrantAddress)
{
    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,
transactionManager,
        new CustomGasProvider());

    String tx_hash;
    try {
        tx_hash = doTransaction(
            folderContract.delRegistrant(registrantAddress),
            "delRegistrant");
    } catch (Exception e) {
        String message = "Error smartDirectoryRegistrantEoaDel: " + e.getMessage();
        android.util.Log.d(LOG_TAG, message);
        tx_hash = message;
    }
    return tx_hash;
}
```



## .smartDirectoryRegistrantLastIndexGet (smartDirectoryAddress)

Cette fonction permet de connaître le dernier index de la liste des déclarants.

```
call QAXH_Eth1 .smartDirectoryRegistrantLastIndexGet  
smartDirectoryAddress
```

La fonction solidity appelée est “`getRegistrantLastIndex`” du smartContract “`SmartDirectory.sol`” :

```
@SimpleFunction(description = "get the last index of the declared registrants")  
public int smartDirectoryRegistrantLastIndexGet(String smartDirectoryAddress) {  
    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,  
transactionManager, new DefaultGasProvider());  
  
    BigInteger result;  
    int resultInt;  
  
    try {  
        result = folderContract.getRegistrantLastIndex().send();  
        resultInt = result.intValue();  
    } catch (Exception e) {  
        android.util.Log.e(LOG_TAG, "Exception: smartDirectoryRegistrantLastIndexGet" +  
e.getMessage());  
        resultInt = -1;  
    }  
    return resultInt;  
}
```

## .smartDirectoryRegistrantAtIndexGet (smartDirectoryAddress, index)

Cette fonction permet de lire l'adresse d'un déclarant en donnant son index. Avec le **lastIndex**, il est donc possible de reconstituer la liste complète des déclarants.

```
call QAXH_Eth1 .smartDirectoryRegistrantAtIndexGet  
smartDirectoryAddress  
index
```

La fonction solidity appelée est “`getRegistrantAtIndex`” du smartContract “`SmartDirectory.sol`” :

```
@SimpleFunction(description = "return registrant address and uri at the given index")  
public Object smartDirectoryRegistrantAtIndexGet(String smartDirectoryAddress, String index) {  
  
    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,  
transactionManager,  
new CustomGasProvider());
```

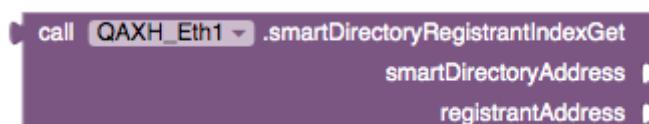


```
Tuple2<String, String> results_raw;  
  
Map<String, String> result_dict = new HashMap<String, String>();  
  
try {  
    results_raw = folderContract.getRegistrantAtIndex(new BigInteger(index)).send();  
    result_dict.put("registrantAddress", results_raw.component1());  
    result_dict.put("registrant_uri", results_raw.component2());  
} catch (Exception e) {  
    String message = "Error smartDirectoryRegistrantAtIndexGet: " + e.getMessage();  
    android.util.Log.e(LOG_TAG, message);  
    result_dict.put("error", message);  
}  
return result_dict;  
}
```

### .smartDirectoryRegistrantIndexGet ((smartDirectoryAddress, registrantAddress))

Cette fonction permet de connaître l'index d'un déclarant. Elle permet aussi de savoir si une adresse valide (autorisée à créer des références) :

- si le retour de cette fonction est “0” (zéro), l'adresse du déclarant n'est pas valide.



La fonction solidity appelée est “getRegistrantIndex” du smartContract “SmartDirectory.sol” :

```
@SimpleFunction(description = "return registrant index given its address")  
public int smartDirectoryRegistrantIndexGet(String smartDirectoryAddress, String registrantAddress) {  
    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress,  
        web3, transactionManager,  
        new CustomGasProvider());  
  
    int result=0;  
    try {  
        result = folderContract.getRegistrantIndex(registrantAddress).send().intValue();  
        return result;  
    } catch (Exception e) {  
        String message = "Error smartDirectoryRegistrantIndexGet: " + e.getMessage();  
        android.util.Log.e(LOG_TAG, message);  
    }  
    return result;  
}
```



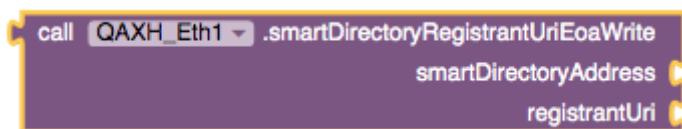
## Enregistrement de l'URI d'un “déclarant” (registrants Table)

Une fois déclaré, l'enregistrement peut être modifié directement par le déclarant.

### .smartDirectoryRegistrantUriEoaWrite (smartDirectoryAddress, registrant\_uri)

Cette fonction permet la mise à jour de la chaîne de caractère de la table des déclarants. La transaction doit être signée par le déclarant (donc l'adresse du déclarant n'est pas dans les paramètres). Sinon, faire un revert.

Si l'adresse de la transaction n'est pas dans la table, faire un revert.



La fonction solidity appelée est “updateRegistrantUri” du smartContract “SmartDirectory.sol” :

```
@SimpleFunction(description = "update uri associated to a registrant")
public String smartDirectoryRegistrantUriEoaWrite(String smartDirectoryAddress, String registrantUri) {

    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,
transactionManager,
    new DefaultGasProvider());

    String tx_hash;
    try {
        tx_hash = doTransaction(
            folderContract.updateRegistrantUri(registrantUri),
            "updateRegistrantUri");
    } catch (Exception e) {
        String message = "Error smartDirectoryRegistrantUriEoaWrite: " + e.getMessage();
        android.util.Log.d(LOG_TAG, message);
        tx_hash = message;
    }
    return tx_hash;
}
```

### .smartDirectoryRegistrantUriGet (smartDirectoryAddress, registrantAddress)

Cette fonction renvoie l'URI de la table des déclarants pour un déclarant donné.

La fonction retourne un dictionnaire :

- dictionnaire vide si l'adresse demandée n'existe pas dans la table
- {"registrant\_uri" : "<string>"} si l'adresse est dans la table



```
call QAXH_Eth1 .smartDirectoryRegistrantUriGet  
    smartDirectoryAddress  
    registrantAddress
```

La fonction solidity appelée est “`getRegistrantUri`” du smartContract “SmartDirectory.sol” :

```
@SimpleFunction(description = "Get uri of the given registrant address")  
public String smartDirectoryRegistrantUriGet(String smartDirectoryAddress, String registrantAddress) {  
    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,  
    transactionManager,  
    new DefaultGasProvider());  
  
    String result;  
    try {  
        result = folderContract.getRegistrantUri(registrantAddress).send();  
    } catch (Exception e) {  
        String message = "Error smartDirectoryRegistrantUriGet: " + e.getMessage();  
        android.util.Log.e(LOG_TAG, message);  
        result = message;  
    }  
    return result;  
}
```



## Création du smartDirectory

Le smartDirectory est un smartContract qui met en œuvre des paramètres afin d'en faciliter la gestion dans les cas d'usage privatifs.

### Variables d'entête

#### Les variables d'en-tête du smartDirectory

parent_address1	adresse	set by API	
parent_address2	adresse	set by API	
contract_version	num	set into the solidity code	version of smart contract
contract_type		42 set by API	
activation_code	0	Attention le code "0" indique la création mais à ce stade le folder n'est pas encore validé : "1" est pour validation par une parentAddress, et "2" signifie que le smartDirectory n'est plus actif	mis à 0 à la création par le hatchServer
contract_uri	string	set by API (could not change)	
admin_code	0,1	"0" les parents doivent enregistrer au préalable les déclarants, "1" toute adresse peut déclarer	

741

© BPCE2024

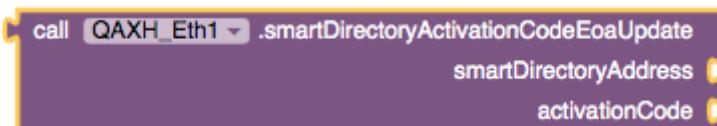
Qaxh.io

- **parent\_address1** : Première adresse du créateur du contrat ou adresse demandée lors de la requête API de création du contrat
- **parent\_address2** : Deuxième adresse du créateur du contrat ou adresse demandée lors de la requête API de création du contrat
- **contract\_version** du contrat pour identifier les évolutions sûrement nécessaire
- **contract\_type** : numéro arbitrairement fixé à "42" permettant de reconnaître en "machine readable" que c'est un smartDirectory
- **activation\_code** : code mis à jour exclusivement par une des deux adresses Parent pour indiquer la validité du smartDirectory :
  - "0" en cours de création, pas encore validé. Aucune référence ne peut être enregistré
  - "1" smartDirectory validé par une parentAddress : toutes les fonctions sont accessibles
  - "2" smartDirectory clôturé par une parentAddress : aucune transaction ni mise à jour ne peut se faire
- **contract\_uri** : string non modifiable à l'usage du créateur du smartDirectory
- **admin\_code** : entier inscrit au déploiement et non modifiable :
  - "0" uniquement les adresses de déclarants enregistrées au préalable par un parent peuvent déclarer
  - "1" toute adresse peut être déclarant (accès d'autodéclaration)



## .smartDirectoryActivationCodeEoaUpdate (smartDirectoryAddress, activationCode)

Cette fonction permet de changer l'activationCode du smartDirectory de "0" vers "1" (ou "2" le cas échéant)  
Le retour de cette fonction est le tx\_Hash de la transaction de mise à jour.



La fonction solidity appelée est “`setSmartDirectoryActivationCode`” du smartContract “`SmartDirectory.sol`” :

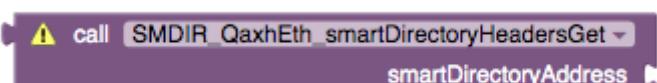
```
@SimpleFunction(description = "Set smartDirectory's activation code")
public String smartDirectoryActivationCodeEoaUpdate(String smartDirectoryAddress, int
activationCode) {
    SmartDirectory folderContract = SmartDirectory.load(smartDirectoryAddress, web3,
transactionManager,
    new DefaultGasProvider());

    return doTransaction(
        folderContract.setSmartDirectoryActivationCode(BigInteger.valueOf(activationCode)),
        "setSmartDirectoryActivationCode");

}
```

## .smartDirectoryHeadersGet (smartDirectoryAddress)

Cette fonction permet la lecture des variables contenues dans les en-têtes sous forme d'un dictionnaire.



Le dictionnaire retourné est constitué de plusieurs appels aux getters solidity dédiés à chaque variables du SmartDirectory :

- `smartDirectoryGetParent1` appelle la fonction solidity “`getParent1`”.
- `smartDirectoryGetParent2` appelle la fonction solidity “`getParent2`”.
- `smartDirectoryGetContractVersion` appelle la fonction solidity “`getContractVersion`”.
- `smartDirectoryGetContractType` appelle la fonction solidity “`getContractType`”.
- `smartDirectoryGetActivationCode` appelle la fonction solidity “`getActivationCode`”.
- `smartDirectoryGetContractUri` appelle la fonction solidity “`getContractUri`”.
- `smartDirectoryGetAdminCode` appelle la fonction solidity “`getAdminCode`”.

L'ensemble des retours de ces appels est ensuite assemblé dans le dictionnaire  
“`smartDirectoryCreateVariablesMap`” lui-même invoqué par la fonction “`smartDirectoryHeadersGet`”.

```
@SimpleFunction(description = "get information about the deployed SmartDirectory")
```



```
private Map<String, String> smartDirectoryCreateVariablesMap(String smartDirectoryAddress) {  
    final Map<String, String> variables = new HashMap<String, String>() {  
        {  
            put("parentAddress1", smartDirectoryGetParent1(smartDirectoryAddress));  
            put("parentAddress2", smartDirectoryGetParent2(smartDirectoryAddress));  
            put("contractVersion", smartDirectoryGetContractVersion(smartDirectoryAddress));  
            put("contractType", smartDirectoryGetContractType(smartDirectoryAddress));  
            put("activationCode", smartDirectoryGetActivationCode(smartDirectoryAddress));  
            put("contractUri", smartDirectoryGetContractUri(smartDirectoryAddress));  
            put("adminCode", smartDirectoryGetMintCode(smartDirectoryAddress));  
        };  
    };  
    return variables;  
}  
  
@SimpleFunction(description = "get all variables of smartDirectory in a dictionary")  
public Object smartDirectoryHeadersGet(String smartDirectoryAddress) {  
    return smartDirectoryCreateVariablesMap(smartDirectoryAddress);  
}
```

En voici un exemple de retour avec AI2



## API de création d'un smartDirectory

Cet API permet pour un utilisateur de créer un smartDirectory facilement sans connaissance préalable de la Blockchain.

### /smart-directory/smardirectorycreate?

**API endpoint:** POST /smart-directory/smardirectorycreate

This API is used to create a smartDirectory smartContract

#### Request Body:

```
{  
    "parent_address1": "0x..",  
    "parent_address2": "0x..",  
    "contract_uri": "uri attaché au smartContract",  
    "admin_code": "1",  
    "chain_id": "80002",  
}
```

"https://smart-directory.qaxh.io/smart-directory/smardirectorycreate?parent\_address1=<parentAddress1>&parent\_address2=<parentAddress2>&contract\_uri=<string>&mint\_code=<1, 2>&chain\_Id=<chainId>

Les autres paramètres nécessaires sont gérés directement par le serveur de déploiement :

- contract\_version : version du code du contrat pour identifier les évolutions
- contract\_type : numéro arbitrairement fixé à “42” permettant de reconnaître en “machine readable” que c'est un smartDirectory
- activation\_code : mis à “0” lors du déploiement

#### Responses:

Success - 200:

```
{  
    "return_code": 200,  
    "tx_hash": "0x....",  
}
```

#### Failures:

- 500 -> mauvais argument dans le call d'API
- 400 -> timer déclenché pendant la déploiement
- 405 -> déploiement en échec

La réponse du serveur peut se faire avant la confirmation du déploiement. C'est à l'entité qui a invoqué l'API de s'assurer que le déploiement est correct en analysant le statut de la transaction de déploiement (tx\_hash) dans la réponse.



## Gestion du smartDirectory pour finaliser le déploiement

Etapes	Commentaires
sur l'APP, remplir les différents éléments de l'API	
valider l'envoi de l'API	
en retour de l'API, attendre la fin du minage du smartDirectory	sur la base du tx_hash reçu
enregistrer l'adresse du smartDirectory dans la base de données de l'APP	
Après fin du déploiement lire les éléments du smartDirectory et les comparer aux éléments demandés et aux spécifications	<a href="#"><b>.smartDirectoryHeadersGet (smartDirectoryAddress)</b></a>
Si tous les contrôles sont ok, proposer à l'utilisateur de valider l'activation du smartDirectory (écran APP)	
signer la transaction d'activation du smartDirectory (nécessite du GAS)	<a href="#"><b>.smartDirectoryActivationCodeEoa Update (smartDirectoryAddress, activationCode)</b></a>
Récupérer le tx_hash de la transaction précédente et attendre le minage	
Si le minage est ok, informer l'utilisateur de l'APP, sinon l'entrée dans la base de données peut être effacée.	



## Diffusion et import de l'adresse du smartDirectory

Etapes	Commentaires
sur l'APP, proposer la liste des smartDirectory	
Après sélection par l'utilisateur afficher l'adresse du smartDirectory dont un QRcode ethereum:<adresse du smartDirectory>@<chain_Id> (EIP 681)	
l'écran de détail doit prévoir un bouton "partager" qui permet l'envoi du QRcode et de l'adresse en chaîne de caractères vers un email, whatsapp,...	

Etapes	Commentaires
sur l'APP, proposer une fonction d'import de smartDirectory qui renvoie sur l'appareil photo	
Analyser le QRcode et le retenir s'il est de la forme "ethereum:<adresse du smartDirectory>[@<chain_Id>] (EIP 681) [entre crochet est optionnel]	
EN cas de syntaxe correcte, lire l'adresse pour voir si elle correspond à un smartDirectory et proposer sa sauvegarde à l'utilisateur	