Extensions serveur Gargantua

Carnet d'adresses

Table des matières

Description du module 9

Mode d'emploi 10

Extensions Gargantua 11

Extensions serveur 11
Extensions base de données 11
Extensions formulaires 11
Carnet d'adresses 11
Fiche de l'extension 11
Paramétrage 11
Extensions processus 11

Scripting 12

Objects 12
AddressBook2 12
newContact 13
newContact 14
createContact 17
readContact 20
updateContact 22
updateContactCoordinates 24
deleteContact 26
idOfContact 27
enumerateContacts 29
newPerson 31
newPerson 32
createPerson 35
readPerson 38
updatePerson 39
updatePersonCoordinates 41
markPersonAsDuplicate 43
markPersonAsNonDuplicate 45
addPersonRelationalLink 47
updatePersonRelationalLink 50
removePersonRelationalLink 52

enumeratePersonRelationalLinks *53*
deletePerson *55*
idOfPerson *57*
enumeratePersons *58*
newOrganisation *60*
newOrganisation *61*
createOrganisation *63*
readOrganisation *65*
updateOrganisation *66*
updateOrganisationCoordinates *68*
markOrganisationAsDuplicate *70*
markOrganisationAsNonDuplicate *72*
deleteOrganisation *74*
idOfOrganisation *75*
enumerateOrganisations *76*
newAddress *78*
newAddress *79*
newPhone *81*
newPhone *82*
newTag *83*
newTag *84*
createTag *85*
readTag *86*
updateTag *87*
addTag *89*
removeTag *91*
deleteTag *93*
idOfTag *94*
enumerateTags *95*
newAssignment *97*
newAssignment *98*
createAssignment *100*
readAssignment *102*
updateAssignment *104*
addAssignment *106*
deleteAssignment *108*
enumerateAssignments *110*
enumerateContactsAssignments *112*
Address 114
getDistrict *115*
getRegion *116*
getStreet1 ..... *117*

getStreet2 ... *118*
getStreet3 ... *119*
getPostcode ... *120*
getCity ... *121*
getPerson ... *122*
getCountry ... *123*
getOrganisation ... *124*
setRegion ... *125*
setDistrict ... *126*
setStreet1 ... *127*
setStreet2 ... *128*
setStreet3 ... *129*
setPostcode ... *130*
setCity ... *131*
setCountry ... *132*
setPerson ... *133*
setOrganisation ... *134*
Assignment ... 135
getEmail ... *136*
getMoved ... *137*
getPnta ... *138*
getPerson ... *139*
getOrganisation ... *140*
getDepartments ... *141*
getPhones ... *142*
setEmail ... *143*
setMoved ... *144*
setPnta ... *145*
setPerson ... *146*
setOrganisation ... *147*
setDepartments ... *148*
setPhones ... *149*
Department ... 150
getDepartment ... *151*
getPosition ... *152*
getAssignment ... *153*
setDepartment ... *154*
setPosition ... *155*
setAssignment ... *156*
Organisation ... 157
getTin ... *158*
getVat ... *159*

getName ... *160*
getAcronym ... *161*
getNace ... *162*
getNaceCountry ... *163*
getWorkforce ... *164*
getLegalCategory ... *165*
getLegalCategoryCountry ... *166*
getService ... *167*
getType ... *168*
getClosed ... *170*
getEmail ... *171*
getAddress ... *172*
getAssignments ... *173*
getOtags ... *174*
setTin ... *175*
setVat ... *176*
getPhones ... *177*
setName ... *178*
setAcronym ... *179*
setNaceCountry ... *180*
setNace ... *181*
setWorkforce ... *182*
setLegalCategoryCountry ... *183*
setLegalCategory ... *184*
setService ... *185*
setClosed ... *186*
setType ... *187*
setEmail ... *188*
setAddress ... *189*
setAssignments ... *190*
setOtags ... *191*
setPhones ... *192*
Person ... 193
getTitle ... *194*
getTitleProf ... *195*
getForename ... *196*
getSurname ... *197*
getAlias ... *198*
getDead ... *199*
getElected ... *200*
getUnionRep ... *201*
getPublicPerson ... *202*

getEmail ... *203*
getAddress ... *204*
getAssignments ... *205*
getPtags ... *206*
getPhones ... *207*
getTargetRelationalLinks ... *208*
getSourceRelationalLinks ... *209*
setTitleProf ... *210*
setAlias ... *211*
setTitle ... *212*
setForename ... *213*
setSurname ... *214*
setDead ... *215*
setElected ... *216*
setUnionRep ... *217*
setPublicPerson ... *218*
setEmail ... *219*
setAddress ... *220*
setAssignments ... *221*
setPtags ... *222*
setPhones ... *223*
setSourceRelationalLinks ... *224*
setTargetRelationalLinks ... *225*
Phone ... 226
getNumber ... *227*
getType ... *228*
getInfo ... *230*
getPerson ... *231*
getOrganisation ... *232*
getAssignment ... *233*
setNumber ... *234*
setType ... *235*
setInfo ... *236*
setPerson ... *237*
setOrganisation ... *238*
setAssignment ... *239*
Tag ... 240
getName ... *241*
getOrganisations ... *242*
getPersons ... *243*
setName ... *244*
setOrganisations ... *245*

setPersons *246*
Functions 246

Description du module

Mode d'emploi

Extensions Gargantua

Extensions serveur

Ce module ne propose aucune extension serveur

Extensions base de données

Ce module ne propose aucune extension de base de données

Extensions formulaires

Carnet d'adresses

Fiche de l'extension

Nom

Carnet d'adresses

Description

Fait le lien entre les attributs du formulaire et le module carnet d'adresses. Vous devez sélectionner les attributs pour chaque champ du formulaire. Les données seront récupérées et affichées à travers des champs de saisie avec auto-complétion

Instances multiples

Oui

Paramétrage

Extensions processus

Ce module ne propose aucune extension de processus

Scripting

Objects

AddressBook2

Cette classe utilitaire de script regroupe des méthodes accessibles depuis le moteur de script concernant les fonctionnalités du carnet d'adresses.

Les méthodes de cette classe sont liées aux méthodes et propriétés de l'objet de l'espace de noms JavaScript AddressBook2.

newContact

```
Object newContact(String contactType)
```

Crée un contact qui sera utilisé ensuite par d'autres fonctions de scripting du carnet d'adresses.

JS call:

`AddressBook2.newContact(contactType)`

Paramètres

contactType

Type de contact à créer (par exemple : `person`, `organisation`).

Ce paramètre détermine les champs et options disponibles pour le nouveau contact.

Retour

Object

Un objet représentant le nouveau contact créé, contenant ses propriétés initiales et prêt à être manipulé via l'API du carnet d'adresses.

Exceptions

Exception

Si la création du contact échoue, par exemple en raison de données invalides, d'un problème d'accès au carnet d'adresses ou d'une erreur interne.

Exemple

```
function newContact() {
    var CONTACT_TYPE = "person";
    var person = AddressBook2.newContact(CONTACT_TYPE);
    return "Initially, the person id is : " + person.getId();
}
```

newContact

```
Object newContact(DTO contactData)
```

Crée un contact qui sera utilisé ensuite par d'autres fonctions de scripting du carnet d'adresses.

JS call:

```
AddressBook2.newContact(contactData)
```

Paramètres

contactData

Objet de type `DTO` contenant toutes les informations nécessaires à la création du contact.

Les champs attendus peuvent inclure le type de contact, les coordonnées et autres métadonnées.

Retour

Object

Un objet représentant le nouveau contact créé, contenant ses propriétés initiales et prêt à être manipulé via l'API du carnet d'adresses.

Exceptions

Exception

Si la création du contact échoue, par exemple en raison de données invalides, d'un problème d'accès au carnet d'adresses ou d'une erreur interne.

Exemple

```
function newPerson() {
    var CONTACT_TYPE = "person";
    var COUNTRY_ISO_3 = "FRA";
    var PERSON_TITLE_MR = 1;
    var surname = "Arnault";
    var forename = "Bernard";
    var alias = "BA";
```

```
    var email = "bernard.arnault@lvmh.fr";
    var title = PERSON_TITLE_MR;
    var titleProf = "PDG";
    var url1 = "https://www.lvmh.com";
    var url2 = "https://www.linkedin.com/in/bernard-arnault";
    var note = "Homme d'affaires français, président-directeur général de LVMH";

    var vip = false;
    var moved = true;
    var pnta = false;
    var dead = false;
    var elected = false;
    var unionRep = false;
    var publicPerson = true;

    var street1 = "Tour Eiffel";
    var street2 = "Champ de Mars, 5 Avenue Anatole France";
    var street3 = "";
    var postcode = "75007";
    var city = "Paris";
    var region = "Île-de-France";
    var country = COUNTRY_ISO_3;

    var phoneList = new java.util.ArrayList(0);
    phoneList.add(DTOFactory.newDTO("number", "+33612345678", "type", "1"));
    phoneList.add(DTOFactory.newDTO("number", "+33123456789", "type", "2"));

    var person = AddressBook2.newContact(DTOFactory.newDTO("contactType",
        CONTACT_TYPE, "surname", surname, "forename", forename, "alias", alias, "email",
email, "title", title, "titleProf", titleProf,
       "url1", url1, "url2", url2, "note", note, "vip", vip, "moved", moved, "pnta",
pnta, "dead", dead, "elected", elected, "unionRep", unionRep,
       "publicPerson", publicPerson, "street1", street1, "street2", street2, "street3",
street3, "postcode", postcode, "city", city,
       "region", region, "country", country, "phones", phoneList));

    return "Initially, the person id is : " + person.getId();
```

```
}
```

createContact

```
void createContact(Object contact)
```

Crée un nouveau contact dans le carnet d'adresses à partir de l'objet fourni.

Cette méthode reçoit un objet représentant les données d'un contact et l'enregistre dans le système ou la base de données associée.

L'objet passé en paramètre peut être une instance de `Person` ou de `Organisation`, selon le type de contact à créer.

JS call:

```
AddressBook2.createContact(contact)
```

Paramètres

contact

Objet de type `Person` ou `Organisation` contenant les informations nécessaires à la création du contact (nom, coordonnées, titre, etc.).

Retour

Aucun

Exceptions

Exception

Si la création du contact échoue, par exemple en raison de données invalides, d'un problème d'accès au carnet d'adresses ou d'une erreur interne.

Exemple

```
function createContact() {
    var CONTACT_TYPE = "person";
    var COUNTRY_ISO_3 = "FRA";
    var PERSON_TITLE_MR = 1;
```

```
    var surname = "Arnault";
    var forename = "Bernard";
    var alias = "BA";
    var email = "bernard.arnault@lvmh.fr";
    var title = PERSON_TITLE_MR;
    var titleProf = "PDG";
    var url1 = "https://www.lvmh.com";
    var url2 = "https://www.linkedin.com/in/bernard-arnault";
    var note = "Homme d'affaires français, président-directeur général de LVMH";

    var vip = false;
    var moved = true;
    var pnta = false;
    var dead = false;
    var elected = false;
    var unionRep = false;
    var publicPerson = true;

    var street1 = "Tour Eiffel";
    var street2 = "Champ de Mars, 5 Avenue Anatole France";
    var street3 = "";
    var postcode = "75007";
    var city = "Paris";
    var region = "Île-de-France";
    var country = COUNTRY_ISO_3;

    var phoneList = new java.util.ArrayList(0);
    phoneList.add(DTOFactory.newDTO("number", "+33612345678", "type", "1"));
    phoneList.add(DTOFactory.newDTO("number", "+33123456789", "type", "2"));

    var person = AddressBook2.newContact(DTOFactory.newDTO("contactType",
        CONTACT_TYPE, "surname", surname, "forename", forename, "alias", alias, "email",
email, "title", title, "titleProf", titleProf,
       "url1", url1, "url2", url2, "note", note, "vip", vip, "moved", moved, "pnta",
pnta, "dead", dead, "elected", elected, "unionRep", unionRep,
       "publicPerson", publicPerson, "street1", street1, "street2", street2, "street3",
street3, "postcode", postcode, "city", city,
```

```
       "region", region, "country", country, "phones", phoneList));

    AddressBook2.createContact(person);

    return "Person id / surname : " + person.getId() + " / " + person.getSurname();
}
```

readContact

```
Object readContact(Long id, String contactType)
```

Lit un contact existant à partir de son identifiant et de son type.

Cette méthode recherche et retourne les informations d'un contact enregistré dans le carnet d'adresses.

Le contact est identifié par son identifiant unique et par son type (par exemple : « person » ou « organisation »).

JS call:

```
AddressBook2.readContact(id, contactType)
```

Paramètres

id

Identifiant unique du contact à lire.

contactType

Type du contact à lire (par exemple : « person » ou « organisation »).

Utilisé pour déterminer la structure et les champs à récupérer.

Retour

Object

Un objet représentant le contact trouvé, contenant ses propriétés et données associées.

Peut retourner `null` si aucun contact ne correspond aux critères.

Exceptions

Exception

Si la lecture du contact échoue, par exemple en raison d'un identifiant inexistant, d'un type invalide ou d'une erreur d'accès aux données.

Exemple

```
function readContact() {
```

```
    var CONTACT_TYPE = "person";

    var contactId = 71210;

    var person = AddressBook2.readContact(contactId, CONTACT_TYPE);


    return "Person surname / forename : " + person.getSurname() + " / " +
person.getForename();

}
```

updateContact

```
void updateContact(Object contact)
```

Met à jour les informations d'un contact existant dans le carnet d'adresses.

Cette méthode reçoit un objet représentant un contact déjà enregistré et applique les modifications fournies à ses données.

L'objet passé en paramètre peut être une instance de `Person` ou de `Organisation`, selon le type de contact à mettre à jour.

JS call:

```
AddressBook2.updateContact(contact)
```

Paramètres

contact

Objet de type `Person` ou `Organisation` contenant les nouvelles informations à enregistrer pour ce contact.

Les champs non modifiés peuvent être laissés inchangés.

Retour

Aucun

Exceptions

Exception

Si la mise à jour échoue, par exemple en raison d'un contact inexistant, de données invalides ou d'une erreur d'accès au stockage.

Exemple

```
function updateContact() {
    var CONTACT_TYPE = "person";
    var contactId = 71210;
    var name = "";
```

```
    try {
        var person = AddressBook2.readContact(contactId, CONTACT_TYPE);
        person.setSurname("Dubois");
        person.setForename("Pierre");
        person.setAlias("PD");
        person.setEmail("pierre.dubois@example.fr");

        AddressBook2.updateContact(person);

        name += person.getSurname() + " / " + person.getForename() + " / "  +
person.getAlias();
    } catch (e) {
     }
    return "Person surname / forename / alias : " + name;
}
```

updateContactCoordinates

```
void updateContactCoordinates(Object contact)
```

Met à jour l'indicateur de coordonnées pour un contact donné.

Le contact peut être une instance de `Person` ou `Organisation`.

Par défaut, le contact est considéré comme ayant des coordonnées (`withCoordinates = true`).

La méthode vérifie ensuite si cette hypothèse est valide :

- Si le contact est une `Person`, elle doit posséder au moins une des informations suivantes : email, adresse ou téléphone.
- Si aucune information n'est présente, la méthode vérifie si la personne est assignée à une ou plusieurs organisations qui possèdent elles-mêmes des coordonnées.
- Si le contact est une `Organisation`, elle est analysée directement pour la présence d'email, adresse ou téléphone.
- Si aucune donnée pertinente n'est trouvée, l'indicateur `withCoordinates` est mis à `false`.

JS call:

```
AddressBook2.updateContactCoordinates(contact)
```

Paramètres

contact

L'objet `Object` représentant un contact à analyser.

Doit être une instance de `Person` ou `Organisation`.

Retour

Aucun

Exceptions

Exception

Si l'objet contact est nul, d'un type inattendu, ou si une erreur survient lors de l'accès aux données ou des vérifications.

Exemple

```
function updateContactCoordinates() {
    var CONTACT_TYPE = "organisation";
    var contactId = 2023;
    var organisation = AddressBook2.readContact(contactId, CONTACT_TYPE);
    organisation.setEmail(null);

    AddressBook2.updateContact(organisation);
    AddressBook2.updateContactCoordinates(organisation);

    return "With coordinates : " + organisation.getWithCoordinates();
}
```

deleteContact

```
void deleteContact(Long id, String contactType)
```

Supprime un contact du système en fonction de son identifiant et de son type.

JS call:

```
AddressBook2.deleteContact(id, contactType)
```

Paramètres

id

L'identifiant unique du contact à supprimer. Ne doit pas être `null`.

contactType

Le type du contact à supprimer (par exemple : « person » ou « organisation »).

Retour

Aucun

Exceptions

Exception

Si l'identifiant est invalide, si le type est inconnu, ou si une erreur survient lors de la suppression du contact.

Exemple

```
function deleteContact() {
    var CONTACT_TYPE = "organisation";
    var contactId = 2023;
    try  {
        AddressBook2.deleteContact(contactId, contactType);
    }  catch(e)  {
     }
}
```

idOfContact

```
Long idOfContact(String name, String contactType)
```

Récupère l'identifiant unique d'un contact en fonction de son nom et de son type.

Le contact peut être de type `Person` ou `Organisation`, selon la valeur du paramètre `contactType`.

- Si le contact est une `Person`, la recherche est effectuée sur le champ `surname`.
- Si le contact est une `Organisation`, la recherche est effectuée sur le champ `name`.

JS call:

```
AddressBook2.idOfContact(name, contactType)
```

Paramètres

name

Le nom du contact à rechercher :

- Pour une `Person`, il s'agit du `surname`.
- Pour une `Organisation`, il s'agit du `name`.

Ne doit pas être `null` ni vide.

contactType

Le type du contact à rechercher. Doit être une valeur valide (ex. « person » ou « organisation »).

Retour

Long

L'identifiant unique du contact correspondant.

Exceptions

Exception

Si les paramètres sont invalides, si le contact n'est pas trouvé, ou si une erreur survient lors de la recherche.

Exemple

```
function idOfContact() {
    var name = " Air France ";
    var CONTACT_TYPE = "organisation";
    var id = AddressBook2. idOfContact( name, CONTACT_TYPE);

    return "Organisation id : " +  id;
}
```

enumerateContacts

```
void enumerateContacts(DTO criteria, List<Object> contacts)
```

Énumère les contacts correspondant à des critères spécifiques.

Cette méthode parcourt une liste de contacts (de type `Person` ou `Organisation`) et applique les filtres définis dans l'objet `DTO` criteria.

Les contacts valides sont ensuite enrichis, transformés ou préparés pour un traitement ultérieur.

JS call:

```
AddressBook2.enumerateContacts(criteria, contacts)
```

Paramètres

criteria

Un objet `DTO` contenant les critères de filtrage ou de sélection.

Peut inclure des champs tels que le type de contact, le nom, la localisation, etc.

contacts

Une liste d'objets `Object` représentant les contacts à analyser.

Chaque élément doit être une instance de `Person` ou `Organisation`.

Retour

Aucun

Exceptions

Exception

Si les paramètres sont invalides, si un type inattendu est rencontré dans la liste, ou si une erreur survient lors du traitement des contacts.

Exemple

```

```

```
function enumerateContacts() {
    var contacts = new java.util.ArrayList(0);
    var criteria = DTOFactory.newDTO();

    var json = '{ "contactList" : [ ';

    try {
       AddressBook2.enumerateContacts(criteria, contacts);
       if (contacts != null && !contacts.isEmpty()) {
          for (var i = 0; i < contacts.size(); i++) {
              var contact = contacts.get(i);
              var contactDTO = contact.toDTO();
              var contactId = contactDTO.get("id");
              var contactName = contactDTO.get("surname") != null ?
contactDTO.get("surname") : contactDTO.get("name");
              var contactType = contactDTO.get("surname") != null ? "person" :
"organisation";

              if (i > 0) {
                 json = json + ', ';
              }
              json = json + '{ "contactId" : "' + contactId  + '", "contactName" : "'
+ contactName  + '", "contactType" : "' + contactType + '"}';
            }
       }
    } catch (e) {
    }
    json = json + ' ] }';

    return json;
}
```

newPerson

```
Person newPerson()
```

Crée et retourne une nouvelle instance de `Person`.

JS call:

```
AddressBook2.newPerson()
```

Paramètres

Aucun

Retour

Person

Une nouvelle instance de `Person`, prête à être complétée ou utilisée.

Exceptions

Exception

Si une erreur survient lors de l'initialisation de l'objet.

Exemple

```
function newPerson() {
    var person = AddressBook2.newPerson();
    return « Initially, the person id is : «  + person.getId();
}
```

newPerson

```
Person newPerson(DTO personData)
```

Crée et retourne une nouvelle instance de `Person` à partir des données fournies.

Cette méthode initialise un objet `Person` en utilisant les informations contenues dans l'objet `DTO personData`.

Elle permet de préremplir les champs tels que le nom, le prénom, l'adresse, l'email, le téléphone, etc., selon les données disponibles.

JS call:

```
AddressBook2.newPerson(personData)
```

Paramètres

personData

Un objet `DTO` contenant les données nécessaires à la création d'une personne.

Ne doit pas être `null`.

Retour

Person

Une nouvelle instance de `Person` initialisée avec les données fournies.

Exceptions

Exception

Si l'objet `personData` est invalide, incomplet ou si une erreur survient lors de l'initialisation.

Exemple

```
function newPerson() {
    var COUNTRY_ISO_3 = "FRA";
    var PERSON_TITLE_MR = 1;

    var surname = "Arnault";
```

```
    var forename = "Bernard";
    var alias = "BA";
    var email = "bernard.arnault@lvmh.fr";
    var title = PERSON_TITLE_MR;
    var titleProf = "PDG";
    var url1 = "https://www.lvmh.com";
    var url2 = "https://www.linkedin.com/in/bernard-arnault";
    var note = "Homme d'affaires français, président-directeur général de LVMH";

    var vip = false;
    var moved = true;
    var pnta = false;
    var dead = false;
    var elected = false;
    var unionRep = false;
    var publicPerson = true;

    var street1 = "Tour Eiffel";
    var street2 = "Champ de Mars, 5 Avenue Anatole France";
    var street3 = "";
    var postcode = "75007";
    var city = "Paris";
    var region = "Île-de-France";
    var country = COUNTRY_ISO_3;

    var phoneList = new java.util.ArrayList(0);
    phoneList.add(DTOFactory.newDTO("number", "+33612345678", "type", "1"));
    phoneList.add(DTOFactory.newDTO("number", "+33123456789", "type", "2"));

    var person = AddressBook2.newPerson(DTOFactory.newDTO("surname", surname,
"forename", forename, "alias", alias, "email", email, "title", title,
       "titleProf", titleProf,  "url1", url1, "url2", url2, "note", note, "vip",
vip, "moved", moved, "pnta", pnta, "dead", dead, "elected", elected, "unionRep",
unionRep,
       "publicPerson", publicPerson, "street1", street1, "street2", street2, "street3",
street3, "postcode", postcode, "city", city,
       "region", region, "country", country, "phones", phoneList));
```

```
    return "Initially, the person id is : " + person.getId();
}
```

createPerson

```
void createPerson(Person person)
```

Crée une nouvelle personne dans le système.

Cette méthode enregistre une instance de `Person` dans le référentiel ou la base de données.

Elle effectue les validations nécessaires sur les données fournies et peut lever une exception en cas d'erreur ou d'incohérence.

JS call:

```
AddressBook2.createPerson(person)
```

Paramètres

person

L'objet `Person` à créer. Ne doit pas être `null` et doit contenir les informations minimales requises (ex. : `surname`, coordonnées).

Retour

Aucun

Exceptions

Exception

Si l'objet `Person` est invalide, incomplet, ou si une erreur survient lors de l'enregistrement dans le système.

Exemple

```
function createPerson() {
    var COUNTRY_ISO_3 = "FRA";
    var PERSON_TITLE_MR = 1;
    var surname = "Arnault";
    var forename = "Bernard";
    var alias = "BA";
    var email = "bernard.arnault@lvmh.fr";
```

```
    var title = PERSON_TITLE_MR;
    var titleProf = "PDG";
    var url1 = "https://www.lvmh.com";
    var url2 = "https://www.linkedin.com/in/bernard-arnault";
    var note = "Homme d'affaires français, président-directeur général de LVMH";

    var vip = false;
    var moved = true;
    var pnta = false;
    var dead = false;
    var elected = false;
    var unionRep = false;
    var publicPerson = true;

    var street1 = "Tour Eiffel";
    var street2 = "Champ de Mars, 5 Avenue Anatole France";
    var street3 = "";
    var postcode = "75007";
    var city = "Paris";
    var region = "Île-de-France";
    var country = COUNTRY_ISO_3;

    var phoneList = new java.util.ArrayList(0);
    phoneList.add(DTOFactory.newDTO("number", "+33612345678", "type", "1"));
    phoneList.add(DTOFactory.newDTO("number", "+33123456789", "type", "2"));

    var person = AddressBook2.newPerson(DTOFactory.newDTO( "surname", surname,
"forename", forename,
       "alias", alias, "email", email, "title", title, "titleProf", titleProf,
      "url1", url1, "url2", url2, "note", note, "vip", vip, "moved", moved, "pnta",
pnta, "dead", dead
       "elected", elected, "unionRep", unionRep,  "publicPerson", publicPerson,
"street1", street1,
       "street2", street2, "street3", street3, "postcode", postcode, "city",
city,
      "region", region, "country", country, "phones", phoneList));

    AddressBook2.createPerson(person);
```

```
    return "Person id / surname : " + person.getId() + " / " + person.getSurname();
}
```

readPerson

```
Person readPerson(Long id)
```

Récupère une instance de `Person` à partir de son identifiant unique.

JS call:

```
AddressBook2.readPerson(id)
```

Paramètres

id

L'identifiant unique de la personne à récupérer. Ne doit pas être `null`.

Retour

Person

L'objet `Person` correspondant à l'identifiant fourni.

Exceptions

Exception

Si l'identifiant est invalide, si aucune personne n'est trouvée, ou si une erreur survient lors de la récupération des données.

Exemple

```
function readPerson() {
    var id = 71210;
    var person = AddressBook2.readPerson(id);

    return "Person surname / forename : " + person.getSurname() + " / " +
person.getForename();
}
```

updatePerson

```
void updatePerson(Person person)
```

Met à jour les informations d'une personne existante dans le système.

Cette méthode remplace les données actuelles de l'objet `Person` identifié par son identifiant unique avec les nouvelles valeurs fournies.

JS call:

`AddressBook2.updatePerson(person)`

Paramètres

person

L'objet `Person` contenant les nouvelles données à enregistrer. Doit inclure les champs à mettre à jour.

Retour

Aucun

Exceptions

Exception

Si l'objet `person` est `null`, si l'identifiant est absent ou invalide, ou si une erreur survient lors de la mise à jour.

Exemple

```
function updatePerson() {
    var personId = 71210;
    var name = "";

    try {
        var person = AddressBook2.readPerson(personId);
        person.setSurname("Dubois");
        person.setForename("Pierre");
```

```
        person.setAlias("PD");
        person.setEmail("pierre.dubois@example.fr");

        AddressBook2.updatePerson(person);

        name += person.getSurname() + " / " + person.getForename() + " / "  +
person.getAlias();
    } catch (e) {
    }
    return "Person surname / forename / alias : " + name;
}
```

updatePersonCoordinates

```
void updatePersonCoordinates(Person person)
```

Met à jour l'indicateur de coordonnées pour une personne donnée.

Par défaut, une personne est considérée comme ayant des coordonnées (`withCoordinates = true`).

La méthode vérifie ensuite si cette hypothèse est valide :

- Si la personne possède au moins une des informations suivantes : email, adresse ou téléphone, elle conserve l'indicateur.
- Sinon, la méthode vérifie si la personne est assignée à une ou plusieurs organisations.
- Pour chaque organisation assignée, elle vérifie si l'une des coordonnées (email, adresse ou téléphone) est présente.
- Si aucune donnée pertinente n'est trouvée ni sur la personne ni sur les organisations auxquelles elle est assignée, l'indicateur `withCoordinates` est mis à `false`.

JS call:

`AddressBook2.updatePersonCoordinates(person)`

Paramètres

person

L'objet `Person` à analyser et mettre à jour. Ne doit pas être `null`.

Retour

Aucun

Exceptions

Exception

Si l'objet `person` est nul ou une erreur survient lors de l'accès aux données.

Exemple

```
function updatePersonCoordinates() {
    var personId = 1012;
```

```
    var person = AddressBook2.readPerson(personId);
    person.setEmail(null);

    AddressBook2.updatePerson(person);
    AddressBook2.updatePersonCoordinates(person);

    return « With coordinates : «  + person.getWithCoordinates();
}
```

markPersonAsDuplicate

```
void markPersonAsDuplicate(Long personId, Long duplicateId)
```

Marque une personne comme étant un doublon d'une autre.

Cette méthode est utilisée pour indiquer que deux enregistrements représentent la même personne réelle.

Elle met à jour l'état de la personne identifiée par `duplicateId` pour refléter qu'elle est un doublon de `personId`.

JS call:

```
AddressBook2.markPersonAsDuplicate(personId, duplicateId)
```

Paramètres

personId

L'identifiant unique de la personne principale (celle conservée).Ne doit pas être `null`.

duplicateId

L'identifiant unique de la personne considérée comme doublon.Ne doit pas être `null`.

Retour

Aucun

Exceptions

Exception

Si une erreur survient lors du traitement (identifiants invalides, accès à la base de données échoué, etc.)

Exemple

```
function markPersonAsDuplicate() {
    var contactType = "person";
    var personSurname = "Dupont";
    var personId = AddressBook2.idOfContact(personSurname, contactType);

    var duplicateSurname = "Dupond";
```

```
    var duplicateId = AddressBook2.idOfContact(duplicateSurname, contactType);


    AddressBook2.markPersonAsDuplicate(personId, duplicateId);
}
```

markPersonAsNonDuplicate

```
void markPersonAsNonDuplicate(Long personId, Long duplicateId)
```

Marque explicitement deux personnes comme étant distinctes (non doublons).

Cette méthode est utilisée pour indiquer que les enregistrements identifiés par `personId` et `duplicateId` représentent deux personnes différentes, et ne doivent pas être fusionnés ou considérés comme des doublons.

JS call:

```
AddressBook2.markPersonAsNonDuplicate(personId, duplicateId)
```

Paramètres

personId

L'identifiant unique de la première personne.

duplicateId

L'identifiant unique de la seconde personne.

Retour

Aucun

Exceptions

Exception

Si une erreur survient lors du traitement (identifiants invalides, accès à la base de données échoué, etc.)

Exemple

```
function markPersonAsNonDuplicate() {
    var contactType = "person";
    var personSurname = "Durand";
    var personId = AddressBook2.idOfContact(personSurname, contactType);

    var duplicateSurname = "Durande";
```

```
    var duplicateId = AddressBook2.idOfContact(duplicateSurname, contactType);


    AddressBook2.markPersonAsNonDuplicate(personId, duplicateId);
}
```

addPersonRelationalLink

```
void addPersonRelationalLink(Long personId, Long personRlinkId, Integer personRlinkType)
```

Ajoute un lien relationnel entre deux personnes dans le système.

Cette méthode permet de définir une relation spécifique entre deux personnes, identifiées par `personId` et `personRlinkId`, selon un type de lien précisé par `personRlinkType`.

Les types de lien peuvent inclure des relations telles que parent, enfant, conjoint, collègue, etc., selon les conventions définies dans le système.

JS call:

```
AddressBook2.addPersonRelationalLink(personId, personRlinkId, personRlinkType)
```

Paramètres

personId

L'identifiant unique de la personne principale.

personRlinkId

L'identifiant unique de la personne liée.

personRlinkType

Le type de relation.

Type de lien relationnel	Valeur	Description de la relation
`PERSON_RLINK_HUSBAND`	1	Mari
`PERSON_RLINK_COHABITANT`	2	Partenaire en concubinage
`PERSON_RLINK_FATHER_SON`	3	Père → Fils
`PERSON_RLINK_SON_FATHER`	4	Fils → Père
`PERSON_RLINK_BROTHER`	5	Frère

Type de lien relationnel	Valeur	Description de la relation
PERSON_RLINK_GRANDFATHER_GRANDSON	6	Grand-père → Petit-fils
PERSON_RLINK_GRANDSON_GRANDFATHER	7	Petit-fils → Grand-père
PERSON_RLINK_HALF_BROTHER	8	Demi-frère
PERSON_RLINK_UNCLE_NEPHEW	9	Oncle → Neveu
PERSON_RLINK_NEPHEW_UNCLE	10	Neveu → Oncle
PERSON_RLINK_COUSIN	11	Cousin / Cousine
PERSON_RLINK_FATHER_IN_LAW_SON_IN_LAW	12	Beau-père → Gendre
PERSON_RLINK_SON_IN_LAW_FATHER_IN_LAW	13	Gendre → Beau-père
PERSON_RLINK_ACADEMIC	14	Relation académique
PERSON_RLINK_FRIENDSHIP	15	Amitié
PERSON_RLINK_EXTENDED_FAMILY	16	Famille élargie
PERSON_RLINK_PROFESSIONAL	17	Relation professionnelle
PERSON_RLINK_NEIGHBORHOOD	18	Voisinage
PERSON_RLINK_OTHER	19	Autre type de relation

Retour

Aucun

EXEMPLE

```
function addPersonRelationalLink() {

    var PERSON_RLINK_BROTHER = 5;

    var personId = 12121;

    var personRlinkId = 12122;


    AddressBook2.addPersonRelationalLink(personId, personRlinkId,
PERSON_RLINK_BROTHER);

}
```

updatePersonRelationalLink

```
void updatePersonRelationalLink(Long personId, Long personRlinkId, Integer
personRlinkType)
```

Met à jour le type de lien relationnel entre deux personnes.

Cette méthode permet de modifier une relation existante entre deux personnes, identifiées par `personId` et `personRlinkId`, en lui attribuant un nouveau type défini par `personRlinkType`.

Les types de relation disponibles sont identiques à ceux utilisés dans la méthode `addPersonRelationalLink`.

JS call:

```
AddressBook2.updatePersonRelationalLink(personId, personRlinkId,
personRlinkType)
```

Paramètres

personId

L'identifiant unique de la personne principale.

personRlinkId

L'identifiant unique de la personne liée.

personRlinkType

Le nouveau type de relation à appliquer, selon les valeurs définies.

Retour

Aucun

Exemple

```
function updatePersonRelationalLink() {
    var PERSON_RLINK_HALF_BROTHER = 8;
    var personId = 10112;
    var personRlinkId = 10113;

    AddressBook2.updatePersonRelationalLink(personId, personRlinkId,
PERSON_RLINK_HALF_BROTHER);
```

```
}
```

removePersonRelationalLink

```
void removePersonRelationalLink(Long personId, Long personRlinkId)
```

Supprime le lien relationnel existant entre deux personnes.

Cette méthode permet de retirer une relation précédemment enregistrée entre `personId` et `personRlinkId`.

Elle est utilisée lorsque le lien n'est plus valide, a été ajouté par erreur, ou doit être révoqué pour des raisons métier.

JS call:

```
AddressBook2.removePersonRelationalLink(personId, personRlinkId)
```

Paramètres

personId

L'identifiant unique de la personne principale.

personRlinkId

L'identifiant unique de la personne liée.

Retour

Aucun

Exemple

```
function removePersonRelationalLink() {
    var personId = 10101;
    var personRlinkId = 20202;

    AddressBook2.removePersonRelationalLink(personId, personRlinkId);
}
```

enumeratePersonRelationalLinks

```
void enumeratePersonRelationalLinks(DTO criteria, List<DTO> rlinks)
```

Énumère les liens relationnels entre personnes selon des critères donnés.

Cette méthode permet de rechercher et de collecter tous les liens relationnels correspondant aux critères spécifiés dans l'objet `criteria`. Les résultats sont ajoutés à la liste `rlinks` fournie en paramètre.

Parmi les critères possibles, on peut inclure :

- `personId` — identifiant de la personne concernée
- `rlinkType` — type de lien relationnel à filtrer

JS call:

```
AddressBook2.enumeratePersonRelationalLinks(criteria, rlinks)
```

Paramètres

criteria

Un objet `DTO` contenant les critères de recherche des liens relationnels.

rlinks

Une liste `DTO` vide qui sera remplie avec les résultats correspondants.

Retour

Aucun

Exemple

```
function enumeratePersonRelationalLinks() {
    var personId = 75895;
    var PERSON_RLINK_BROTHER = 5;

    var rlinks = new java.util.ArrayList(0);
    var criteria = DTOFactory.newDTO("personId", personId, "rlinkType",
PERSON_RLINK_BROTHER);

    AddressBook2.enumeratePersonRelationalLinks(criteria, rlinks);
```

```
    var json = '{ "rlinks" : [ ';


    if (rlinks != null && !rlinks.isEmpty()) {
       for (var i = 0; i < rlinks.size(); i++) {
           var rlink = rlinks.get(i);
           var person = rlink.xget("person");
           var rlinkType = rlink.getInteger("rlinkType");


           if (i > 0) {
              json = json + ', ';
           }
           json = json + '{ "personId" : "' + person.getId() + '",  "rlinkType" : "' +
rlinkType + '"}';
       }
    }
    json = json + ' ] }';


    return json;
}
```

deletePerson

```
void deletePerson(Long id)
```

Supprime une personne du système selon son identifiant.

Cette méthode permet de retirer définitivement une personne de la base de données, en fonction de son identifiant unique `id`.

Toutes les données associées à cette personne peuvent également être supprimées ou désactivées selon les règles métier.

JS call:

```
AddressBook2.deletePerson(id)
```

Paramètres

id

L'identifiant unique de la personne à supprimer.

Retour

Aucun

Exceptions

Exception

Si une erreur survient lors de l'opération, par exemple :

- identifiant inexistant ou invalide
- conflits avec des dépendances relationnelles
- échec d'accès à la base de données

Exemple

```
function deletePerson() {
    var id = 70335;
    try {
       AddressBook2.deletePerson(id);
```

```
    } catch (e) {
    }
}
```

idOfPerson

```
Long idOfPerson(String name)
```

Récupère l'identifiant unique d'une personne à partir de son nom de famille.

Cette méthode permet d'interroger le système pour retrouver l'identifiant `Long` associé à une personne dont le nom de famille est fourni en paramètre.

JS call:

`AddressBook2.idOfPerson(name)`

Paramètres

name

Le nom de famille (`surname`) de la personne recherchée.

Retour

Long

L'identifiant unique de la personne correspondante.

Exceptions

Exception

Si aucune correspondance n'est trouvée, si le nom est invalide, ou en cas d'erreur d'accès à la base de données.

Exemple

```
function idOfPerson() {
    var surname = "Roslansky";
    var id = AddressBook2.idOfPerson(surname);

    return "Person id : " + id;
}
```

enumeratePersons

```
void enumeratePersons(DTO criteria, List<Person> persons)
```

Énumère les personnes correspondant aux critères spécifiés.

Cette méthode permet de rechercher et de collecter toutes les personnes qui répondent aux critères définis dans l'objet `criteria`. Les résultats sont ajoutés à la liste `persons` fournie en paramètre.

Parmi les critères possibles, on peut inclure :

- `surname` — nom de famille
- `forename` — prénom
- `register` — code contact
- `email` — adresse électronique
- tout autre champ pertinent selon la structure du `DTO`

JS call:

```
AddressBook2.enumeratePersons(criteria, persons)
```

Paramètres

criteria

Un objet `DTO` contenant les critères de recherche des personnes.

persons

Une liste vide qui sera remplie avec les personnes correspondantes.

Retour

Aucun

Exceptions

Exception

Si une erreur survient lors de la récupération des données.

EXEMPLE

```
function enumeratePersons() {
    var persons = new java.util.ArrayList(0);
    var criteria = DTOFactory.newDTO();

    var json = '{ "persons" : [ ';

    try {
       AddressBook2.enumeratePersons(criteria, persons);

       if (persons != null && !persons.isEmpty()) {
          for (var i = 0; i < persons.size(); i++) {
             var person = persons.get(i);
             var name = person.getSurname() + " " + person.getForename();
             var alias = person.getAlias() != null ? person.getAlias() : "";

             if (i > 0) {
                json = json + ', ';
             }
             json = json + '{ "id" : "' + person.getId() + '", "name" : "' + name + '",
"alias" : "' + alias + '"}';
          }
       }
    } catch (e) {
    }

    json = json + ' ] }';

    return json;
}
```

newOrganisation

```
Organisation newOrganisation()
```

Crée et retourne une nouvelle instance de `Organisation`.

JS call:

```
AddressBook2.newOrganisation()
```

Paramètres

Aucun

Retour

Organisation

Une nouvelle instance de `Organisation`, prête à être complétée ou utilisée.

Exceptions

Exception

Si une erreur survient lors de l'initialisation de l'objet.

Exemple

```
function newOrganisation() {
    var organisation = AddressBook2.newOrganisation();
    return « Initially, the organisation id is : «  + organisation.getId();
}
```

newOrganisation

```
Organisation newOrganisation(DTO organisationData)
```

Crée et retourne une nouvelle instance de `Organisation` à partir des données fournies.

Cette méthode initialise un objet `Organisation` en utilisant les informations contenues dans l'objet `DTO organisationData`.

Elle permet de préremplir les champs tels que le nom, l'adresse, l'email, le téléphone, etc., selon les données disponibles.

JS call:

```
AddressBook2.newOrganisation(organisationData)
```

Paramètres

organisationData

Un objet `DTO` contenant toutes les informations nécessaires à la création de l'organisation.

Ne doit pas être `null`.

Retour

Organisation

Une nouvelle instance de `Organisation` initialisée avec les données fournies.

Exceptions

Exception

Si l'objet `organisationData` est invalide, incomplet ou si une erreur survient lors de l'initialisation.

Exemple

```
function newOrganisation() {
    var COUNTRY_ISO_3 = "FRA";

    var name = "Air France";
```

```
    var email = "corporate.communications@airfrance.fr";
    var url1 = "https://www.airfrance.com";
    var url2 = "https://www.linkedin.com/company/air-france";
    var note = "Compagnie aérienne française.";

    var street1 = "Tour Eiffel";
    var street2 = "Champ de Mars, 5 Avenue Anatole France";
    var street3 = "";
    var postcode = "75007";
    var city = "Paris";
    var region = "Île-de-France";
    var country = COUNTRY_ISO_3;

    var phoneList = new java.util.ArrayList(0);
    phoneList.add(DTOFactory.newDTO("number", "+33612345678", "type", "1"));
    phoneList.add(DTOFactory.newDTO("number", "+33123456789", "type", "2"));

    var organisation = AddressBook2.newOrganisation(DTOFactory.newDTO("name", name,
"email", email, "url1", url1, "url2", url2, "note", note, "street1", street1,
       "street2", street2, "street3", street3, "postcode", postcode, "city",
city,
      "region", region, "country", country, "phones", phoneList));

    return "Initially, the organisation id is : " + organisation.getId();
}
```

createOrganisation

```
void createOrganisation(Organisation organisation)
```

Crée une nouvelle organisation dans le système.

Cette méthode enregistre une instance de `Organisation` dans le référentiel ou la base de données.

Elle effectue les validations nécessaires sur les données fournies et peut lever une exception en cas d'erreur ou d'incohérence.

JS call:

```
AddressBook2.createOrganisation(organisation)
```

Paramètres

organisation

L'objet `Organisation` à créer. Ne doit pas être `null` et doit contenir les informations minimales requises (ex. : `name`, coordonnées).

Retour

Aucun

Exceptions

Exception

Si l'objet `Organisation` est invalide, incomplet, ou si une erreur survient lors de l'enregistrement dans le système.

Exemple

```
function createOrganisation() {
    var COUNTRY_ISO_3 = "FRA";
    var name = "Air France";
    var acronym = "AIR FRA";
    var email = "air.france@contact.fr";
    var url1 = "https://www.airfrance.com";
```

```
    var url2 = "https://www.linkedin.com/in/air-france";


    var street1 = "Tour Eiffel";
    var street2 = "Champ de Mars, 5 Avenue Anatole France";
    var street3 = "";
    var postcode = "75007";
    var city = "Paris";
    var region = "Île-de-France";
    var country = COUNTRY_ISO_3;


    var phoneList = new java.util.ArrayList(0);
    phoneList.add(DTOFactory.newDTO("number", "+33612345678", "type", "1"));
    phoneList.add(DTOFactory.newDTO("number", "+33123456789", "type", "2"));


    var organisation = AddressBook2.newOrganisation(DTOFactory.newDTO("name", name,
"acronym", acronym,
       "email", email,  "url1", url1, "url2", url2,  "street1", street1,
"street2", street2, "street3", street3,          "postcode", postcode, "city",
city,  "region", region, "country", country, "phones", phoneList));


    AddressBook2.createOrganisation(organisation);


    return "Organisation id / name : " + organisation.getId() + " / " +
organisation.getName();
}
```

readOrganisation

```
Organisation readOrganisation(Long id)
```

Récupère une instance de `Organisation` à partir de son identifiant unique.

JS call:

```
AddressBook2.readOrganisation(id)
```

Paramètres

id

L'identifiant de l'organisation à lire. Ne doit pas être `null`.

Retour

Organisation

L'objet `Organisation` correspondant à l'identifiant fourni.

Exceptions

Exception

Si une erreur survient lors de la lecture de l'organisation.

Exemple

```
function readOrganisation() {
    var id = 75210;
    var organisation = AddressBook2.readOrganisation(id);

    return "Organisation name : " + organisation.getName();
}
```

updateOrganisation

```
void updateOrganisation(Organisation organisation)
```

Met à jour les informations d'une organisation existante.

JS call:

`AddressBook2.updateOrganisation(organisation)`

Paramètres

organisation

L'objet `organisation` contenant les nouvelles données à enregistrer.

Retour

Aucun

Exceptions

Exception

Si l'objet `organisation` est `null`, si l'identifiant est absent ou invalide, ou si une erreur survient lors de la mise à jour.

Exemple

```
function updateOrganisation() {
    var organisationId = 71210;
    var name = "";

    try {
        var organisation = AddressBook2.readOrganisation(organisationId);
        organisation.setName("Air FRANCE");
        organisation.setAcronym("AFR");
        organisation.setEmail("air.france@example.fr");

        AddressBook2.updateOrganisation(organisation);
```

```
        name += organisation.getName() + " / " + organisation.getAcronym() ;
    } catch (e) {
    }
    return "Organisation name / acronym : " + name;
}
```

updateOrganisationCoordinates

```
void updateOrganisationCoordinates(Organisation organisation)
```

Met à jour l'indicateur de coordonnées pour une organisation donnée.

Par défaut, une organisation est considérée comme ayant des coordonnées (`withCoordinates = true`).

La méthode vérifie ensuite si cette hypothèse est valide :

- Si l'organisation possède au moins une des informations suivantes : email, adresse ou téléphone, elle conserve l'indicateur.
- Si aucune donnée pertinente n'est trouvée, l'indicateur `withCoordinates` est mis à `false`.

JS call:

```
AddressBook2.updateOrganisationCoordinates(organisation)
```

Paramètres

organisation

L'objet `Organisation` à analyser et mettre à jour. Ne doit pas être `null`.

Retour

Aucun

Exceptions

Exception

Si l'objet `organisation` est nul ou une erreur survient lors de l'accès aux données.

Exemple

```
function updateOrganisationCoordinates() {
    var organisationId = 1012;
    var organisation = AddressBook2.readOrganisation(organisationId);
    organisation.setEmail(null);
```

```
    AddressBook2.updateOrganisation(organisation);
    AddressBook2.updateOrganisationCoordinates(organisation);

    return « With coordinates : «  + organisation.getWithCoordinates();
}
```

markOrganisationAsDuplicate

```
void markOrganisationAsDuplicate(Long organisationId, Long duplicateId)
```

Marque une organisation comme étant un doublon d'une autre organisation.

Cette méthode permet d'indiquer qu'une organisation identifiée par `organisationId` est considérée comme un doublon de l'organisation identifiée par `duplicateId`. Cette information peut être utilisée pour éviter les incohérences dans les données, fusionner des enregistrements ou signaler des doublons dans l'application.

JS call:

```
AddressBook2.markOrganisationAsDuplicate(organisationId, duplicateId)
```

Paramètres

organisationId

L'identifiant unique de l'organisation principale (de référence). Ne doit pas être `null`.

duplicateId

L'identifiant unique de l'organisation considérée comme doublon. Ne doit pas être `null`.

Retour

Aucun

Exceptions

Exception

Si une erreur survient lors du traitement (identifiants invalides, accès à la base de données échoué, etc.)

Exemple

```
function markOrganisationAsDuplicate() {
    var contactType = « organisation »;
    var organisationName = « Air France »;
    var organisationId = AddressBook2.idOfContact(organisationName, contactType);

    var duplicateName = «  Aire France »;
```

```
    var duplicateId = AddressBook2.idOfContact(duplicateName, contactType);


    AddressBook2.markOrganisationAsDuplicate(organisationId, duplicateId);
}
```

markOrganisationAsNonDuplicate

```
void markOrganisationAsNonDuplicate(Long organisationId, Long duplicateId)
```

Marque explicitement deux organisations comme étant distinctes (non doublons).

Cette méthode est utilisée pour indiquer que les enregistrements identifiés par `organisationId` et `duplicateId` représentent deux organisations différentes et ne doivent pas être fusionnés ou considérés comme des doublons.

JS call:

```
AddressBook2.markOrganisationAsNonDuplicate(organisationId, duplicateId)
```

Paramètres

organisationId

L'identifiant unique de l'organisation principale (référence).

duplicateId

L'identifiant unique de l'organisation précédemment marquée comme doublon.

Retour

Aucun

Exceptions

Exception

Si une erreur survient lors du traitement (identifiants invalides, accès à la base de données échoué, etc.)

Exemple

```
function markOrganisationAsNonDuplicate() {
    var contactType = « organisation »;
    var organisationName = « Air France »;
    var organisationId = AddressBook2.idOfContact(organisationName, contactType);

    var duplicateName = « Aire France »;
```

```
    var duplicateId = AddressBook2.idOfContact(duplicateName, contactType);


    AddressBook2.markOrganisationAsNonDuplicate(organisationId, duplicateId);
}
```

deleteOrganisation

```
void deleteOrganisation(Long id)
```

Supprime une organisation du système.

Cette méthode efface définitivement l'organisation identifiée par `id` ainsi que toutes les données qui lui sont directement associées. L'opération est irréversible et doit être utilisée avec précaution.

JS call:

```
AddressBook2.deleteOrganisation(id)
```

Paramètres

id

L'identifiant unique de l'organisation à supprimer.

Retour

Aucun

Exceptions

Exception

Si une erreur survient lors de l'opération, par exemple :

- identifiant inexistant ou invalide
- conflits avec des dépendances relationnelles
- échec d'accès à la base de données

Exemple

```
function deleteOrganisation() {
    var id = 70335;
    try {
       AddressBook2.deleteOrganisation(id);
    } catch (e) {
    }
}
```

idOfOrganisation

```
Long idOfOrganisation(String name)
```

Retourne l'identifiant unique d'une organisation à partir de son nom.

Cette méthode recherche dans le système l'organisation dont le nom correspond à la valeur fournie en paramètre `name` et renvoie son identifiant unique.

JS call:

```
AddressBook2.idOfOrganisation(name)
```

Paramètres

name

Le nom de l'organisation dont on souhaite obtenir l'identifiant. Ne doit pas être `null` ni vide.

Retour

Long

L'identifiant unique de l'organisation trouvée.

Exceptions

Exception

Si aucune correspondance n'est trouvée, si le nom est invalide, ou en cas d'erreur d'accès à la base de données.

Exemple

```
function idOfOrganisation() {
    var name = « Air France »;
    var id = AddressBook2.idOfOrganisation(name);

    return « Organisation id : «  + id;
}
```

enumerateOrganisations

```
void enumerateOrganisations(DTO criteria, List<Organisation> organisations)
```

Remplit la liste fournie avec les organisations correspondant aux critères spécifiés.

Cette méthode effectue une recherche dans le système en fonction des paramètres contenus dans l'objet `criteria` et ajoute à la liste `organisations` toutes les organisations qui répondent à ces critères.

Les critères peuvent inclure, entre autres :

- `name` — nom de l'organisation
- `register` — code contact ou numéro d'enregistrement
- `email` — adresse électronique
- tout autre champ pertinent selon la structure du `DTO`

JS call:

```
AddressBook2.enumerateOrganisations(criteria, organisations)
```

Paramètres

criteria

Un objet `DTO` contenant les critères de filtrage.

organisations

La liste dans laquelle seront ajoutées les organisations trouvées.

Retour

Aucun

Exceptions

Exception

Si une erreur survient lors de la récupération des données.

Exemple

```
function enumerateOrganisations() {
```

```
    var organisations = new java.util.ArrayList(0);
    var criteria = DTOFactory.newDTO();

    var json = '{ "organisations" : [ ';

    try {
       AddressBook2.enumerateOrganisations(criteria, organisations);

       if (organisations != null && !organisations.isEmpty()) {
          for (var i = 0; i < organisations.size(); i++) {
             var organisation = organisations.get(i);
             var name = organisation.getName();
             var acronym = organisation.getAcronym() != null ?
organisation.getAcronym() : "";

             if (i > 0) {
                json = json + ', ';
             }
             json = json + '{ "id" : "' + organisation.getId() + '", "name" : "' + name
+ '", "acronym" : "' + acronym + '"}';
          }
       }
    } catch (e) {
    }

    json = json + ' ] }';

    return json;
}
```

newAddress

```
Address newAddress()
```

Crée et retourne une nouvelle instance d'adresse.

Cette méthode initialise un objet `Address` avec des valeurs par défaut (par exemple, champs vides ou valeurs null) afin qu'il puisse être complété et utilisé dans le système.

JS call:

```
AddressBook2.newAddress()
```

Paramètres

Aucun

Retour

Address

Un objet `Address` nouvellement créé, prêt à être renseigné.

Exceptions

Exception

Si une erreur survient lors de l'initialisation de l'objet.

Exemple

```
function newAddress() {
    var address = AddressBook2.newAddress();
    return « Initially, the address id is : «  + address.getId();
}
```

newAddress

```
Address newAddress(DTO addressData)
```

Crée une nouvelle instance `Address` à partir des données fournies.

JS call:

```
AddressBook2.newAddress(addressData)
```

Paramètres

addressData

Un objet `DTO` contenant les informations suivantes :

- street1
- street2
- street3
- postcode
- city
- district
- region
- country

Retour

Address

Une nouvelle instance de `Address` initialisée avec les données fournies.

Exceptions

Exception

Si une erreur survient lors de l'initialisation de l'objet.

Exemple

```
function newAddress() {
    var addressData = DTOFactory.newDTO();
    addressData.put("street1", "10 Rue de Rivoli");
```

```
    addressData.put("street2", "");
    addressData.put("street3", "");
    addressData.put("postcode", "75001");
    addressData.put("city", "Paris");
    addressData.put("district", "1er arrondissement");
    addressData.put("region", "Île-de-France");
    addressData.put("country", "France");

    var address = AddressBook2.newAddress(addressData);

    return "Initially, the address id is : " + address.getId();
}
```

newPhone

```
Phone newPhone()
```

Crée une nouvelle instance de `Phone`.

JS call:

```
AddressBook2.newPhone()
```

Paramètres

Aucun

Retour

Phone

Une nouvelle instance de `Phone`.

Exceptions

Exception

Si une erreur survient lors de l'initialisation de l'objet.

Exemple

```
function newPhone() {
    var phone = AddressBook2.newPhone();
    return « Initially, the phone id is : «  + phone.getId();
}
```

newPhone

```
Phone newPhone(DTO phoneData)
```

Crée une nouvelle instance de `Phone` à partir des données fournies.

JS call:

```
AddressBook2.newPhone(phoneData)
```

Paramètres

phoneData

Un objet `DTO` contenant les informations suivantes :

- number
- type (voir la documentation de la classe `Phone` pour les valeurs possibles)
- info

Retour

Phone

Une nouvelle instance de `Phone` initialisée avec les données fournies.

Exceptions

Exception

Si une erreur survient lors de l'initialisation de l'objet.

Exemple

```
function newPhone() {

    var phone = AddressBook2.newPhone(DTOFactory.newDTO( »number »,  »+40750665523 »,
« type », « 1 »)));

    return « Initially, the phone id is : «  + phone.getId();

}
```

newTag

```
Tag newTag()
```

Crée une nouvelle instance de `Tag`.

JS call:

```
AddressBook2.newTag()
```

Paramètres

Aucun

Retour

Tag

Une nouvelle instance de `Tag`.

Exceptions

Exception

Si une erreur survient lors de l'initialisation de l'objet.

Exemple

```
function newTag() {
    var tag = AddressBook2.newTag();
    return « Initially, the tag id is : «  + tag.getId();
}
```

newTag

```
Tag newTag(DTO tagData)
```

Crée une nouvelle instance de `Tag`.

JS call:

```
AddressBook2.newTag(tagData)
```

Paramètres

tagData

Un objet `DTO` qui contient uniquement le champ `name`.

Retour

Tag

Une nouvelle instance de `Tag` initialisée avec les données fournies.

Exceptions

Exception

Si une erreur survient lors de l'initialisation de l'objet.

Exemple

```
function newTag() {
    var tag = AddressBook2.newTag(DTOFactory.newDTO("name", "SIATEL")));
    return "Initially, the tag id is : " + tag.getId();
}
```

createTag

```
void createTag(Tag tag)
```

Crée et enregistre un nouvel objet `Tag`.

JS call:

```
AddressBook2.createTag(tag)
```

Paramètres

tag

Un objet `Tag` contenant le champ `name`.

Retour

Aucun

Exceptions

Exception

Si l'objet `Tag` est invalide, incomplet, ou si une erreur survient lors de l'enregistrement dans le système.

Exemple

```
function createTag() {
    var tag = AddressBook2.newTag();
    tag.setName("SIATEL");

    AddressBook2.createTag(tag);

    return "Tag id : " + tag.getId();
}
```

readTag

```
Tag readTag(Long id)
```

Récupère un objet `Tag` existant à partir de son identifiant unique.

JS call:

```
AddressBook2.readTag(id)
```

Paramètres

id

L'identifiant unique du tag à lire.

Retour

Tag

L'objet `Tag` correspondant à l'identifiant fourni.

Exceptions

Exception

Si l'identifiant est nul, inexistant ou si une erreur survient lors de la récupération du tag.

Exemple

```
function readTag() {
    var id = 73588;
    var tag = AddressBook2.readTag(id);

    return "Tag name : " + tag.getName();
}
```

updateTag

```
void updateTag(Tag tag)
```

Met à jour un objet `Tag` existant avec de nouvelles informations.

JS call:

`AddressBook2.updateTag(tag)`

Paramètres

tag

Un objet `Tag` contenant les nouvelles informations (par ex. un nouveau nom).

Retour

Aucun

Exceptions

Exception

Si l'objet `Tag` est invalide, si l'identifiant est manquant ou inexistant, ou si une erreur survient lors de la mise à jour.

Exemple

```
function updateTag() {
    var newName = "SIATEL_TAG";
    var id = 73588;

    try {
       var tag = AddressBook2.readTag(id);
       tag.setName(newName);

       AddressBook2.updateTag(tag);
    } catch (e) {
    }
```

```
}
```

addTag

```
void addTag(Object contact, List<Tag> tags)
```

Associe une ou plusieurs étiquettes (`Tag`) à un contact donné.

JS call:

```
AddressBook2.addTag(contact, tags)
```

Paramètres

contact

L'objet représentant le contact auquel les tags doivent être ajoutés.

tags

La liste des objets `Tag` à associer au contact.

Retour

Aucun

Exceptions

Exception

Si le contact est invalide, si la liste de tags est nulle ou vide, ou si une erreur survient lors de l'association.

Exemple

```
function addTag() {
    var contactId = 73512;
    var contactType = "person";
    var contact = AddressBook2.readContact(contactId, contactType);

    var tagId1 = 75673;
    var tag1 = AddressBook2.readTag(tagId1);

    var tagId2 = 67044;
```

```
    var tag2 = AddressBook2.readTag(tagId2);


    var tags = new java.util.ArrayList(0);
    tags.add(tag1);
    tags.add(tag2);


    try {
       AddressBook2.addTag(contact, tags);
    } catch (e) {
    }
}
```

removeTag

```
void removeTag(Object contact, Tag tag)
```

Supprime l'association d'une étiquette (`Tag`) spécifique d'un contact donné.

JS call:

```
AddressBook2.removeTag(contact, tag)
```

Paramètres

contact

L'objet représentant le contact dont le tag doit être retiré.

tag

L'objet `Tag` à supprimer de ce contact.

Retour

Aucun

Exceptions

Exception

Si le contact est invalide, si le tag est nul ou inexistant, ou si une erreur survient lors de la suppression.

Exemple

```
function removeTag() {
    var contactId = 73512;
    var tagId = 71316;

    var person = AddressBook2.readContact(contactId, "person");
    var tag = AddressBook2.readTag(tagId);

    try {
       AddressBook2.removeTag(person, tag);
```

```
    } catch (e) {
    }
}
```

deleteTag

```
void deleteTag(Long id)
```

Supprime définitivement un objet `Tag` existant à partir de son identifiant unique.

JS call:

```
AddressBook2.deleteTag(id)
```

Paramètres

id

L'identifiant unique du tag à supprimer.

Retour

Aucun

Exceptions

Exception

Si l'identifiant est nul, inexistant ou si une erreur survient lors de la suppression.

Exemple

```
function deleteTag() {
    var id = 75678;
    try {
       AddressBook2.deleteTag(id);
    } catch (e) {
    }
}
```

idOfTag

```
Long idOfTag(String name)
```

Récupère l'identifiant unique d'un `Tag` à partir de son nom.

JS call:

`AddressBook2.idOfTag(name)`

Paramètres

name

Le nom du tag dont on souhaite obtenir l'identifiant.

Retour

Long

L'identifiant unique du tag correspondant.

Exceptions

Exception

Si le nom est nul, vide, ou si aucun tag correspondant n'est trouvé, ou encore si une erreur survient lors de la recherche.

Exemple

```
function idOfTag() {
    var name = "SIATEL";
    var id = AddressBook2.idOfTag(name);

    return "Tag id : " + id;
}
```

enumerateTags

```
void enumerateTags(DTO criteria, List<Tag> tags)
```

Remplit une liste de `Tag` en fonction des critères de recherche fournis.

JS call:

```
AddressBook2.enumerateTags(criteria, tags)
```

Paramètres

criteria

Un objet `DTO` pouvant contenir les clés suivantes :

- `query` : une chaîne de caractères (`String`)
- `tagIds` : une liste de chaînes (`List`)
- `tagNames` : une liste de chaînes (`List`)

tags

La liste de `Tag` qui sera remplie avec les résultats correspondant aux critères.

Retour

Aucun

Exceptions

Exception

Si les critères sont invalides, si la liste est nulle, ou si une erreur survient lors de l'énumération.

Exemple

```
function enumerateTags() {
    var tags = new java.util.ArrayList(0);
    var json = '{ "tagList" : [ ';

    try {
       var criteria = DTOFactory.newDTO("query", "SIATEL");
```

```
        AddressBook2.enumerateTags(criteria, tags);

        if (tags != null && !tags.isEmpty()) {
           for (var i = 0; i < tags.size(); i++) {
              var tag = tags.get(i);

              if (i > 0) {
                 json = json + ', ';
              }
              json = json + '{ "tagId" : "' + tag.getId()  + '", "tagName" : "' +
tag.getName() + '"}';
           }
        }
    } catch (e) {
    }

    json = json + ' ] }';

    return json;
}
```

newAssignment

```
Assignment newAssignment()
```

Crée une nouvelle instance `Assignment`.

JS call:

```
AddressBook2.newAssignment()
```

Paramètres

Aucun

Retour

Assignment

Un objet `Assignment` nouvellement créé.

Exceptions

Exception

Si une erreur survient lors de l'initialisation de l'objet.

Exemple

```
function newAssignment() {
    var assignment = AddressBook2.newAssignment();
    return « Initially, the assignment id is : «  + assignment.getId();
}
```

newAssignment

```
Assignment newAssignment(DTO assigData)
```

Crée une nouvelle instance `Assignment` à partir des données fournies.

Cette méthode utilise les informations contenues dans l'objet `DTO assigData` pour initialiser un nouvel objet `Assignment`. Elle peut lever une exception si les données sont invalides ou si une erreur survient lors de la création.

JS call:

```
AddressBook2.newAssignment(assigData)
```

Paramètres

assigData

Les données nécessaires à la création de l'`Assignment`.

Retour

Assignment

Une nouvelle instance de `Assignment` initialisée avec les données fournies.

Exceptions

Exception

Si une erreur survient lors de l'initialisation de l'objet.

Exemple

```
function newAssignment() {
    var phoneList = new java.util.ArrayList(0);
    phoneList.add(DTOFactory.newDTO("number", "+40750665523", "type", "1"));
    phoneList.add(DTOFactory.newDTO("number", "+40250405090", "type", "2"));

    var personId = 75895;
    var organisationId = 71256;
```

```
    var assigData = DTOFactory.newDTO("personId", personId, "organisationId",
organisationId, "department", "Software", "position", "Engineer",

                    "email", "siatel.software@gmail.com", "phones", phoneList);

    var assignment = AddressBook2.newAssignment(assigData);

}
```

createAssignment

```
void createAssignment(Assignment assignment)
```

Enregistre un nouvel objet `Assignment` dans le système.

Cette méthode prend un objet `Assignment` en paramètre et effectue les opérations nécessaires pour le persister ou le traiter selon la logique métier. Elle peut lever une exception si l'objet est invalide ou si une erreur survient lors de l'enregistrement.

JS call:

```
AddressBook2.createAssignment(assignment)
```

Paramètres

assignment

L'objet `Assignment` à créer.

Retour

Aucun

Exceptions

Exception

Si une erreur se produit lors de la création ou de la persistance.

Exemple

```
function createAssignment() {
    var phoneList = new java.util.ArrayList(0);
    phoneList.add(DTOFactory.newDTO("number", "+ 33612345678", "type", "1"));
    phoneList.add(DTOFactory.newDTO("number", "+ 33123456789 ", "type", "2"));

    var personId = 75895;
    var organisationId = 71256;
```

```
    var assigData = DTOFactory.newDTO("personId", personId, "organisationId",
organisationId, "department", "Software", "position", "Engineer",

                    "email", "siatel.software@gmail.com", "phones", phoneList);

    var assignment = AddressBook2.newAssignment(assigData);


    AddressBook2.createAssignment(assignment);

}
```

readAssignment

```
Assignment readAssignment(Long personId, Long organisationId)
```

Récupère un objet `Assignment` associé à une personne et une organisation spécifiques.

Cette méthode recherche un `Assignment` en fonction de l'identifiant de la personne et de l'identifiant de l'organisation. Elle retourne l'objet correspondant s'il est trouvé, ou lève une exception si aucun résultat n'est disponible ou si une erreur survient lors de l'accès aux données.

JS call:

```
AddressBook2.readAssignment(personId, organisationId)
```

Paramètres

personId

L'identifiant unique de la personne.

organisationId

L'identifiant unique de l'organisation.

Retour

Assignment

L'objet `Assignment` correspondant aux identifiants fournis.

Exceptions

Exception

Si une erreur se produit lors de la lecture ou si aucun `Assignment` n'est trouvé.

Exemple

```
function readAssignment() {
    var personId = 70770;
    var organisationId =  70660;

    var assignment = AddressBook2.readAssignment(personId, organisationId);
```

```
    var email = assignment.getEmail();

    var departments = assignment.getDepartments() != null ? (new
java.util.ArrayList(assignment.getDepartments())) : null;

    var department = departments != null ? departments.get(0) : null;

    var deptName = department != null ? department.getDepartment() : "";

    var position = department != null ? department.getPosition() : "";


    var mobile = "";

    var phones = assignment.getPhones() != null ? (new
java.util.ArrayList(assignment.getPhones())) : null;


    if (phones != null) {

       for (var i = 0; i < phones.size(); i++) {

           var phone = phones.get(i);


           if (phone.getType() == 1) {

              mobile = phone.getNumber();

           }

       }

    }

    return "Assignment info : " + email + ", " + deptName + ", " + position + ", "  +
mobile;

}
```

updateAssignment

```
void updateAssignment(Assignment assignment)
```

Met à jour un objet `Assignment` existant dans le système.

Cette méthode prend un objet `Assignment` en paramètre et applique les modifications nécessaires à l'enregistrement correspondant. Elle suppose que l'objet fourni contient les informations actualisées et qu'il est déjà présent dans le système.

JS call:

```
AddressBook2.updateAssignment(assignment)
```

Paramètres

assignment

L'objet `Assignment` contenant les données mises à jour.

Retour

Aucun

Exceptions

Exception

Si une erreur survient lors de la mise à jour ou si l'objet est introuvable.

Exemple

```
function updateAssignment() {
    var personId = 70123;
    var organisationId = 71234;
    var assignment = AddressBook2.readAssignment(personId, organisationId);

    try {
       assignment.setEmail("contact.airfrance@gmail.com");
       assignment.setMoved(true);
```

```
        var phoneList = new java.util.ArrayList(0);

        phoneList.add(AddressBook2.newPhone(DTOFactory.newDTO("number", "+33612345678 ",
"type", "1")));

        phoneList.add(AddressBook2.newPhone(DTOFactory.newDTO("number", "+33123456789 ",
"type", "2")));

        assignment.setPhones(new java.util.HashSet(phoneList));


        AddressBook2.updateAssignment(assignment);

    } catch (e) {

    }


    return "Assignment email : " +  assignment.getEmail();

}
```

addAssignment

```
void addAssignment(DTO assigData)
```

Ajoute un nouvel `Assignment` à partir des données fournies.

Cette méthode utilise l'objet `DTO` contenant les informations nécessaires pour créer et enregistrer un nouvel `Assignment` dans le système. Elle peut lever une exception si les données sont invalides ou si une erreur survient lors du traitement.

JS call:

```
AddressBook2.addAssignment(assigData)
```

Paramètres

assigData

Les données d'entrée nécessaires à la création de l'`Assignment`.

Retour

Aucun

Exceptions

Exception

Si une erreur se produit lors de l'ajout ou de la validation des données.

Exemple

```
function addAssignment() {
    var phoneList = new java.util.ArrayList(0);
    phoneList.add(DTOFactory.newDTO( »number », « +33612345678 », « type », « 1 »));
    phoneList.add(DTOFactory.newDTO( »number », « +33123456789 », « type », « 2 »));

    var personId = 71234;
    var organisationId = 71256;
    var assigData = DTOFactory.newDTO( »personId », personId, « organisationId »,
organisationId, « department », « Software », « position », « Engineer »,
```

```
                « email », « siatel.software@gmail.com », « phones », phoneList);
    AddressBook2.addAssignment(assigData);
}
```

deleteAssignment

```
void deleteAssignment(Long personId, Long organisationId)
```

Supprime un `Assignment` associé à une personne et une organisation spécifiques.

Cette méthode identifie l'objet `Assignment` à supprimer en fonction de l'identifiant de la personne et de celui de l'organisation. Elle effectue la suppression dans le système si l'objet existe, ou lève une exception en cas d'erreur ou si aucun `Assignment` correspondant n'est trouvé.

JS call:

```
AddressBook2.deleteAssignment(personId, organisationId)
```

Paramètres

personId

L'identifiant unique de la personne.

organisationId

L'identifiant unique de l'organisation.

Retour

Aucun

Exceptions

Exception

Si une erreur survient lors de la suppression ou si l'objet est introuvable.

Exemple

```
function deleteAssignment() {
    var personId = 70990;
    var organisationId =  70770;

    try {
        AddressBook2.deleteAssignment(personId, organisationId);
```

```
    } catch (e) {
    }
}
```

enumerateAssignments

```
void enumerateAssignments(DTO criteria, List<Assignment> assignments)
```

Énumère les `Assignment` correspondant aux critères spécifiés.

Cette méthode applique les conditions définies dans l'objet `DTO` afin d'identifier les `Assignment` pertinents et de les ajouter à la liste fournie. Elle peut lever une exception si les critères sont invalides ou si une erreur survient lors du traitement.

Les critères attendus incluent notamment :

- `id` : identifiant unique du contact concerné
- `contactType` : type de contact à prendre en compte (`person`, `organisation`)

JS call:

```
AddressBook2.enumerateAssignments(criteria, assignments)
```

Paramètres

criteria

Les critères de filtrage sous forme de `DTO`, incluant les clés `id` (identifiant du contact) et `contactType` (type de contact).

assignments

La liste de `Assignment` qui sera complétée avec les résultats trouvés.

Retour

Aucun

Exceptions

Exception

Si une erreur se produit lors de l'énumération.

Exemple

```
function enumerateAssignments() {
```

```
    var PERSON_TYPE = "person";
    var personId = 71235;

    var assigs = new java.util.ArrayList(0);

    var criteria = DTOFactory.newDTO();
    criteria.put("id",  personId);
    criteria.put("contactType", PERSON_TYPE);

    var json = '{ "assigsList" : [ ';

    try {
        AddressBook2.enumerateAssignments(criteria, assigs);
        if (assigs != null && !assigs.isEmpty()) {
            for (var i = 0; i < assigs.size(); i++) {
               var assignment = assigs.get(i);
               var assignmentDTO = assignment.toDTO();
               var assigEmail = assignmentDTO.get("email") != null ? assignmentDTO
.get("email")  : "";

               if (i > 0) {
                  json = json + ', ';
               }
               json = json + '{ "assigEmail" : "' + assigEmail + '"}';
            }
        }
    } catch (e) {
    }
    json = json + ' ] }';

    return json;
}
```

enumerateContactsAssignments

```
void enumerateContactsAssignments(DTO criteria, List<Object> contacts)
```

Énumère les contacts affectés à une personne ou à une organisation donnée.

Cette méthode utilise les critères définis dans l'objet `DTO` pour identifier les contacts concernés et complète la liste fournie avec les résultats trouvés.

Les critères attendus incluent notamment :

- `id` : identifiant unique du contact
- `contactType` : type de contact à prendre en compte (ex. `person`, `organisation`)

JS call:

```
AddressBook2.enumerateContactsAssignments(criteria, contacts)
```

Paramètres

criteria

Les critères de filtrage sous forme de `DTO`, incluant au minimum les clés « `id`« quot; (identifiant du contact) et « `contactType`« quot; (type de contact).

contacts

La liste de contacts qui sera complétée avec les résultats trouvés.

Retour

Aucun

Exceptions

Exception

Si une erreur se produit lors de l'énumération.

Exemple

```
function enumerateContactsAssignments() {
    var CONTACT_TYPE = « organisation »;
```

```
    var contactId = 2080;


    var contacts = new java.util.ArrayList(0);


    var criteria = DTOFactory.newDTO();
    criteria.put( »id », contactId);
    criteria.put( »contactType », CONTACT_TYPE);


    var json = '{ « contactsAssigs » : [ ';


    try {
        AddressBook2.enumerateContactsAssignments(criteria, contacts);
        if (contacts != null && !contacts.isEmpty()) {
           for (var i = 0; i < contacts.size(); i++) {
              var contact = contacts.get(i);
              var surname = contact.getSurname() != null ? contact.getSurname() :
«  »;
              var forename = contact.getForename() != null ? contact.getForename() :
«  »;
              var email = contact.getEmail() != null ? contact.getEmail()  : «  »;


              if (i > 0) {
                 json = json + ', ';
              }


              json = json + '{ « personInfo » : « ' + surname + ', ' + forename  + ', '
+ email + '« }';
           }
         }
    } catch (e) {
    }


    json = json + ' ] }';


    return json;
}
```

Address

Représente une adresse pouvant être associée à des entités du domaine applicatif.

Une adresse permet d'identifier et de localiser un contact ou une organisation (exemples : adresse postale, siège social, lieu de facturation).

Caractéristiques techniques :

- Entité JPA persistée dans une table
- Audit activé via Hibernate Envers
- Sérialisation JSON avec gestion d'identité pour éviter les boucles infinies

Utilisation :

Cette classe est utilisée pour stocker et gérer les informations d'adresse des personnes, organisations ou autres entités de l'application. Elle facilite la gestion des coordonnées, l'intégration avec d'autres modules et le suivi des modifications dans le temps.

getDistrict

```
String getDistrict()
```

Paramètres

Aucun

Retour

Exemple

getRegion

```
String getRegion()
```

Paramètres

Aucun

Retour

Exemple

getStreet1

```
String getStreet1()
```

Paramètres

Aucun

Retour

Exemple

getStreet2

```
String getStreet2()
```

Paramètres

Aucun

Retour

Exemple

getStreet3

```
String getStreet3()
```

Paramètres

Aucun

Retour

Exemple

getPostcode

```
String getPostcode()
```

Paramètres

Aucun

Retour

Exemple

getCity

```
String getCity()
```

Paramètres

Aucun

Retour

Exemple

getPerson

```
Person getPerson()
```

Paramètres

Aucun

Retour

Exemple

getCountry

```
String getCountry()
```

Paramètres

Aucun

Retour

Exemple

getOrganisation

```
Organisation getOrganisation()
```

Paramètres

Aucun

Retour

Exemple

setRegion

```
void setRegion(String region)
```

Paramètres

region

...

Retour

Aucun

Exemple

setDistrict

```
void setDistrict(String district)
```

Paramètres

district

...

Retour

Aucun

Exemple

setStreet1

```
void setStreet1(String street1)
```

Paramètres

street1

...

Retour

Aucun

Exemple

setStreet2

```
void setStreet2(String street2)
```

Paramètres

street2

...

Retour

Aucun

Exemple

setStreet3

```
void setStreet3(String street3)
```

Paramètres

street3

...

Retour

Aucun

Exemple

setPostcode

```
void setPostcode(String postcode)
```

Paramètres

postcode

...

Retour

Aucun

Exemple

setCity

```
void setCity(String city)
```

Paramètres

city

...

Retour

Aucun

Exemple

setCountry

```
void setCountry(String country)
```

Paramètres

country

...

Retour

Aucun

Exemple

setPerson

```
void setPerson(Person person)
```

Paramètres

person

...

Retour

Aucun

Exemple

setOrganisation

```
void setOrganisation(Organisation organisation)
```

Paramètres

organisation

...

Retour

Aucun

Exemple

Assignment

Représente une affectation (assignment) dans le module organisationnel de l'application.

Une affectation correspond au lien entre une personne, une organisation ou un département et un rôle ou une responsabilité spécifique.

Caractéristiques techniques :

- Entité JPA persistée dans la table
- Audit activé via Hibernate Envers
- Sérialisation JSON avec gestion d'identité pour éviter les boucles infinies
- Ignorance de certaines propriétés lors de la sérialisation JSON (departments, phones) pour simplifier les échanges

Utilisation :

Cette classe est utilisée pour gérer les relations d'affectation entre les personnes, les organisations et leurs départements. Elle facilite la gestion des rôles, des responsabilités et des rattachements hiérarchiques dans l'application.

getEmail

```
String getEmail()
```

Paramètres

Aucun

Retour

Exemple

getMoved

```
boolean getMoved()
```

Paramètres

Aucun

Retour

Exemple

getPnta

```
boolean getPnta()
```

Paramètres

Aucun

Retour

Exemple

getPerson

```
Person getPerson()
```

Paramètres

Aucun

Retour

Exemple

getOrganisation

```
Organisation getOrganisation()
```

Paramètres

Aucun

Retour

Exemple

getDepartments

```
Set<Department> getDepartments()
```

Paramètres

Aucun

Retour

Exemple

getPhones

```
Set<Phone> getPhones()
```

Paramètres

Aucun

Retour

Exemple

setEmail

```
void setEmail(String email)
```

Paramètres

email

...

Retour

Aucun

Exemple

setMoved

```
void setMoved(boolean moved)
```

Paramètres

moved

...

Retour

Aucun

Exemple

setPnta

```
void setPnta(boolean pnta)
```

Paramètres

pnta

...

Retour

Aucun

Exemple

setPerson

```
void setPerson(Person person)
```

Paramètres

person

...

Retour

Aucun

Exemple

setOrganisation

```
void setOrganisation(Organisation organisation)
```

Paramètres

organisation

...

Retour

Aucun

Exemple

setDepartments

```
void setDepartments(Set<Department> departments)
```

Paramètres

departments

...

Retour

Aucun

Exemple

setPhones

```
void setPhones(Set<Phone> phones)
```

Paramètres

phones

...

Retour

Aucun

Exemple

Department

Représente un département dans le module organisationnel de l'application.

Un département correspond à une unité interne d'une organisation (exemples : service commercial, département RH, département technique).

Caractéristiques techniques :

- Entité JPA persistée dans la table
- Audit activé via Hibernate Envers (sans audit des entités liées)
- Sérialisation JSON avec gestion d'identité pour éviter les boucles infinies

Utilisation :

Cette classe est utilisée pour gérer la structure interne des organisations dans l'application. Elle facilite la classification des entités, la gestion hiérarchique et l'intégration avec d'autres modules liés aux contacts et aux organisations.

getDepartment

```
String getDepartment()
```

Paramètres

Aucun

Retour

Exemple

getPosition

```
String getPosition()
```

Paramètres

Aucun

Retour

Exemple

getAssignment

```
Assignment getAssignment()
```

Paramètres

Aucun

Retour

Exemple

setDepartment

```
void setDepartment(String department)
```

Paramètres

department

...

Retour

Aucun

Exemple

setPosition

```
void setPosition(String position)
```

Paramètres

position

...

Retour

Aucun

Exemple

setAssignment

```
void setAssignment(Assignment assignment)
```

Paramètres

assignment

...

Retour

Aucun

Exemple

Organisation

Représente une organisation dans le module carnet d'adresses de l'application.

Une organisation regroupe les informations relatives à une entité juridique ou commerciale (exemples : entreprise, association, institution).

Caractéristiques techniques :

- Entité JPA persistée dans la table
- Audit activé via Hibernate Envers
- Sérialisation JSON avec gestion d'identité pour éviter les boucles infinies
- Ignorance de certaines propriétés lors de la sérialisation JSON (assignments, phones, dateC, dateM, dateU) pour simplifier les échanges

Utilisation :

Cette classe est utilisée pour gérer les informations des organisations (clients, partenaires, fournisseurs, institutions) dans l'application.

Elle facilite la centralisation des données, le suivi des modifications et l'intégration avec d'autres entités du carnet d'adresses.

getTin

```
String getTin()
```

Paramètres

Aucun

Retour

Exemple

getVat

```
String getVat()
```

Paramètres

Aucun

Retour

Exemple

getName

```
String getName()
```

Paramètres

Aucun

Retour

Exemple

getAcronym

```
String getAcronym()
```

Paramètres

Aucun

Retour

Exemple

getNace

```
String getNace()
```

Paramètres

Aucun

Retour

Exemple

getNaceCountry

```
String getNaceCountry()
```

Paramètres

Aucun

Retour

Exemple

getWorkforce

```
String getWorkforce()
```

Paramètres

Aucun

Retour

Exemple

getLegalCategory

```
String getLegalCategory()
```

Paramètres

Aucun

Retour

Exemple

getLegalCategoryCountry

```
String getLegalCategoryCountry()
```

Paramètres

Aucun

Retour

Exemple

getService

```
String getService()
```

Paramètres

Aucun

Retour

Exemple

getType

```
Integer getType()
```

Paramètres

Aucun

Retour

Integer

Le type de l'organisation

Constantes représentant les différents types d'organisation pouvant être associés à une entité Organisation dans le système.

Ces valeurs sont utilisées pour catégoriser les organisations et faciliter leur traitement dans la logique métier, les filtres, les recherches et les exports.

Types d'organisation	Valeur	Description
`ORGANISATION_TYPE_NONE`	`0`	Aucun type d'organisation spécifié. Utilisé lorsque l'organisation ne correspond à aucune catégorie prédéfinie ou que l'information n'est pas disponible.
`ORGANISATION_TYPE_ASSOCIATION`	`1`	Organisation de type association. Représente une structure associative à but non lucratif (ex. : club, ONG, association culturelle ou sportive).
`ORGANISATION_TYPE_COMPANY`	`2`	Organisation de type entreprise. Représente une société commerciale ou industrielle, quelle que soit sa forme juridique.
`ORGANISATION_TYPE_UNION`	`3`	Organisation de type syndicat. Représente une organisation syndicale ou professionnelle défendant les intérêts de ses membres.

Types d'organisation	Valeur	Description
`ORGANISATION_TYPE_OTHER`	`4`	Organisation d'un autre type. Utilisé pour toute organisation ne correspondant pas aux catégories précédentes (ex. : groupement informel, consortium, institution spécifique).

Exemple

getClosed

```
boolean getClosed()
```

Paramètres

Aucun

Retour

Exemple

getEmail

```
String getEmail()
```

Paramètres

Aucun

Retour

Exemple

getAddress

```
Address getAddress()
```

Paramètres

Aucun

Retour

Exemple

getAssignments

```
Set<Assignment> getAssignments()
```

Paramètres

Aucun

Retour

Exemple

getOtags

```
Set<Tag> getOtags()
```

Paramètres

Aucun

Retour

Exemple

setTin

```
void setTin(String tin)
```

Paramètres

tin

...

Retour

Aucun

Exemple

setVat

```
void setVat(String vat)
```

Paramètres

vat

...

Retour

Aucun

Exemple

getPhones

```
Set<Phone> getPhones()
```

Paramètres

Aucun

Retour

Exemple

setName

```
void setName(String name)
```

Paramètres

name

...

Retour

Aucun

Exemple

setAcronym

```
void setAcronym(String acronym)
```

Paramètres

acronym

...

Retour

Aucun

Exemple

setNaceCountry

```
void setNaceCountry(String naceCountry)
```

Paramètres

naceCountry

...

Retour

Aucun

Exemple

setNace

```
void setNace(String nace)
```

Paramètres

nace

...

Retour

Aucun

Exemple

setWorkforce

```
void setWorkforce(String workforce)
```

Paramètres

workforce

...

Retour

Aucun

Exemple

setLegalCategoryCountry

```
void setLegalCategoryCountry(String legalCategoryCountry)
```

Paramètres

legalCategoryCountry

...

Retour

Aucun

Exemple

setLegalCategory

```
void setLegalCategory(String legalCategory)
```

Paramètres

legalCategory

...

Retour

Aucun

Exemple

setService

```
void setService(String service)
```

Paramètres

service

...

Retour

Aucun

Exemple

setClosed

```
void setClosed(boolean closed)
```

Paramètres

closed

...

Retour

Aucun

Exemple

setType

```
void setType(Integer type)
```

Paramètres

type

...

Retour

Aucun

Exemple

setEmail

```
void setEmail(String email)
```

Paramètres

email

...

Retour

Aucun

Exemple

setAddress

```
void setAddress(Address address)
```

Paramètres

address

...

Retour

Aucun

Exemple

setAssignments

```
void setAssignments(Set<Assignment> assignments)
```

Paramètres

assignments

...

Retour

Aucun

Exemple

setOtags

```
void setOtags(Set<Tag> otags)
```

Paramètres

otags

...

Retour

Aucun

Exemple

setPhones

```
void setPhones(Set<Phone> phones)
```

Paramètres

phones

...

Retour

Aucun

Exemple

Person

Représente une personne dans le module carnet d'adresses de l'application.

Une personne regroupe les informations de contact et d'identification nécessaires pour la gestion des relations (exemples : nom, prénom, coordonnées).

Caractéristiques techniques :

- Entité JPA persistée dans la table
- Audit activé via Hibernate Envers
- Sérialisation JSON avec gestion d'identité pour éviter les boucles infinies
- Ignorance de certaines propriétés lors de la sérialisation JSON (assignments, phones) pour simplifier les échanges

Utilisation :

Cette classe est utilisée pour gérer les informations personnelles et professionnelles des contacts dans l'application. Elle facilite la centralisation des données, le suivi des modifications et l'intégration avec d'autres entités du carnet d'adresses.

getTitle

```
Integer getTitle()
```

Paramètres

Aucun

Retour

Integer

La civilité ou titre de politesse

Ces constantes définissent les valeurs entières associées aux différentes civilités ou titres de politesse pouvant être attribués à une personne dans l'application.

Civilité	Valeur	Description
`PERSON_TITLE_NONE`	`0`	Aucune civilité (valeur par défaut ou non renseignée)
`PERSON_TITLE_MR`	`1`	Monsieur
`PERSON_TITLE_MRS`	`2`	Madame (femme mariée)
`PERSON_TITLE_MS`	`3`	Mademoiselle / Madame (forme neutre, sans indication d'état civil)
`PERSON_TITLE_MRS_MR`	`4`	Madame et Monsieur (couple)
`PERSON_TITLE_LADIES`	`5`	Mesdames
`PERSON_TITLE_GENTLEMEN`	`6`	Messieurs

Exemple

getTitleProf

```
String getTitleProf()
```

Paramètres

Aucun

Retour

Exemple

getForename

```
String getForename()
```

Paramètres

Aucun

Retour

Exemple

getSurname

```
String getSurname()
```

Paramètres

Aucun

Retour

Exemple

getAlias

```
String getAlias()
```

Paramètres

Aucun

Retour

Exemple

getDead

```
boolean getDead()
```

Paramètres

Aucun

Retour

Exemple

getElected

```
boolean getElected()
```

Paramètres

Aucun

Retour

Exemple

getUnionRep

```
boolean getUnionRep()
```

Paramètres

Aucun

Retour

Exemple

getPublicPerson

```
boolean getPublicPerson()
```

Paramètres

Aucun

Retour

Exemple

getEmail

```
String getEmail()
```

Paramètres

Aucun

Retour

Exemple

getAddress

```
Address getAddress()
```

Paramètres

Aucun

Retour

Exemple

getAssignments

```
Set<Assignment> getAssignments()
```

Paramètres

Aucun

Retour

Exemple

getPtags

```
Set<Tag> getPtags()
```

Paramètres

Aucun

Retour

Exemple

getPhones

```
Set<Phone> getPhones()
```

Paramètres

Aucun

Retour

Exemple

getTargetRelationalLinks

```
Set<RelationalLink> getTargetRelationalLinks()
```

Paramètres

Aucun

Retour

Exemple

getSourceRelationalLinks

```
Set<RelationalLink> getSourceRelationalLinks()
```

Paramètres

Aucun

Retour

Exemple

setTitleProf

```
void setTitleProf(String titleProf)
```

Paramètres

titleProf

...

Retour

Exemple

setAlias

```
void setAlias(String alias)
```

Paramètres

alias

...

Retour

Exemple

setTitle

```
void setTitle(Integer title)
```

Paramètres

title

...

Retour

Exemple

setForename

```
void setForename(String forename)
```

Paramètres

forename

...

Retour

Exemple

setSurname

```
void setSurname(String surname)
```

Paramètres

surname

...

Retour

Exemple

setDead

```
void setDead(boolean dead)
```

Paramètres

dead

...

Retour

Exemple

setElected

```
void setElected(boolean elected)
```

Paramètres

elected

...

Retour

Exemple

setUnionRep

```
void setUnionRep(boolean unionRep)
```

Paramètres

unionRep

...

Retour

Exemple

setPublicPerson

```
void setPublicPerson(boolean publicPerson)
```

Paramètres

publicPerson

...

Retour

Exemple

setEmail

```
void setEmail(String email)
```

Paramètres

email

...

Retour

Exemple

setAddress

```
void setAddress(Address address)
```

Paramètres

address

...

Retour

Aucun

Exemple

setAssignments

```
void setAssignments(Set<Assignment> assignments)
```

Paramètres

assignments

...

Retour

Aucun

Exemple

setPtags

```
void setPtags(Set<Tag> ptags)
```

Paramètres

ptags

...

Retour

Aucun

Exemple

setPhones

```
void setPhones(Set<Phone> phones)
```

Paramètres

phones

...

Retour

Aucun

Exemple

setSourceRelationalLinks

```
void setSourceRelationalLinks(Set<RelationalLink> sourceRelationalLinks)
```

Paramètres

sourceRelationalLinks

...

Retour

Exemple

setTargetRelationalLinks

```
void setTargetRelationalLinks(Set<RelationalLink> targetRelationalLinks)
```

Paramètres

targetRelationalLinks

...

Retour

Exemple

Phone

Représente une entité de numéro de téléphone dans le système d’annuaire.

Cette classe est persistée en base de données via JPA et auditable via Hibernate Envers. Elle peut être liée à une personne, une organisation ou une affectation.

getNumber

```
String getNumber()
```

Récupère le numéro de téléphone associé à cette entité.

Ce champ représente la valeur principale du contact téléphonique, telle qu'elle serait saisie par un utilisateur ou importée depuis une source externe.

Il peut s'agir d'un numéro mobile, fixe, fax, ou autre, selon le type défini.

Paramètres

Aucun

Retour

String

le numéro de téléphone sous forme de chaîne de caractères

Exemple

getType

```
Integer getType()
```

Récupère le type de téléphone associé à ce numéro.

Le type est représenté par une constante entière (voir les constantes définies dans la classe).

Il permet de distinguer les usages : mobile principal, fixe secondaire, fax, etc.

Cette information est cruciale pour l'affichage conditionnel, le tri ou les filtres dans l'interface utilisateur.

Paramètres

Aucun

Retour

Le type de téléphone sous forme d'entier.

Type de numéro de téléphone	Valeur	Description
`TELEPHONE_TYPE_MOBILE`	`1`	Représente un numéro de téléphone mobile principal. Ce type est généralement utilisé pour le numéro de portable personnel ou professionnel le plus fréquemment utilisé par une personne ou une organisation.
`TELEPHONE_TYPE_PHONE`	`2`	Représente un numéro de téléphone fixe principal. Ce type correspond au numéro de ligne fixe principal, souvent utilisé dans les bureaux, les domiciles ou les sièges sociaux.
`TELEPHONE_TYPE_MOBILE_2`	`3`	Représente un second numéro de téléphone mobile. Ce type est utile pour enregistrer un numéro de secours, un deuxième appareil mobile, ou un numéro temporaire.
`TELEPHONE_TYPE_PHONE_2`	`4`	Représente un second numéro de téléphone fixe.

Type de numéro de téléphone	Valeur	Description
		Ce type peut être utilisé pour une ligne secondaire, un poste interne ou un numéro alternatif dans une organisation.
`TELEPHONE_TYPE_FAX`	5	Représente un numéro de télécopie (fax). Ce type est utilisé pour les communications par fax, encore présentes dans certains contextes administratifs ou juridiques.

Exemple

getInfo

```
String getInfo()
```

Récupère les informations complémentaires liées à ce numéro.

Ce champ est optionnel et peut contenir des précisions utiles pour l'utilisateur : « ligne directe », « numéro temporaire », « standard accueil », etc.

Il enrichit la compréhension du contexte d'utilisation du numéro.

Paramètres

Aucun

Retour

Les informations complémentaires liées à ce numéro

Exemple

getPerson

```
Person getPerson()
```

Récupère la personne associée à ce numéro de téléphone.

Cette relation permet de lier un numéro à une entité Person, représentant un individu.

Elle est utilisée pour naviguer dans les relations entre contacts et leurs coordonnées.

Paramètres

Aucun

Retour

La personne associée à ce numéro de téléphone

Exemple

getOrganisation

```
Organisation getOrganisation()
```

Récupère l'organisation associée à ce numéro de téléphone.

Cette relation permet de rattacher un numéro à une entité Organisation, utile dans les contextes professionnels ou institutionnels.

Paramètres

Aucun

Retour

L'organisation associée à ce numéro de téléphone

Exemple

getAssignment

```
Assignment getAssignment()
```

Récupère l'affectation liée à ce numéro de téléphone.

Une affectation représente un rôle, un poste ou une mission spécifique dans le système.

Cette relation permet de contextualiser le numéro dans une structure organisationnelle.

Paramètres

Aucun

Retour

L'affectation liée à ce numéro de téléphone

Exemple

setNumber

```
void setNumber(String number)
```

Définit le numéro de téléphone pour cette entité.

Cette méthode permet d'enregistrer ou de modifier la valeur du numéro de téléphone.

Elle doit être utilisée avec des validations en amont pour garantir le format et la cohérence (ex. : format international, absence de caractères non numériques).

Paramètres

number

le numéro de téléphone à enregistrer

Retour

Aucun.

Exemple

setType

```
void setType(Integer type)
```

Définit le type de téléphone pour ce numéro.

Cette méthode doit être utilisée pour catégoriser le numéro selon son usage.

Les valeurs attendues sont des constantes prédéfinies (ex. : `TELEPHONE_TYPE_MOBILE` ou `TELEPHONE_TYPE_FAX`). Voir getType pour détails sur les valeurs possibles.

Paramètres

type

Le type de téléphone associé à ce numéro

Retour

Aucun.

Exemple

setInfo

```
void setInfo(String info)
```

Définit les informations complémentaires pour ce numéro.

Cette méthode permet d'ajouter des précisions textuelles sur le numéro.

Elle est utile pour les interfaces utilisateurs ou les exports de données.

Paramètres

info

Les informations complémentaires pour ce numéro

Retour

Aucun

Exemple

setPerson

```
void setPerson(Person person)
```

Définit la personne associée à ce numéro de téléphone.

Cette méthode établit une relation entre le numéro et une personne.

Elle est utile lors de la création ou de la mise à jour d'un contact.

Paramètres

person

La personne associée à ce numéro de téléphone

Retour

Aucun

Exemple

setOrganisation

```
void setOrganisation(Organisation organisation)
```

Définit l'organisation associée à ce numéro de téléphone.

Cette méthode permet de relier le numéro à une organisation, comme une entreprise ou une administration.

Paramètres

organisation

L'organisation associée à ce numéro de téléphone

Retour

Aucun

Exemple

setAssignment

```
void setAssignment(Assignment assignment)
```

Définit l'affectation liée à ce numéro de téléphone.

Cette méthode permet de relier le numéro à une affectation précise, utile pour les systèmes RH ou les organigrammes.

Paramètres

assignment

L'affectation liée à ce numéro de téléphone

Retour

Aucun

Exemple

Tag

Représente une étiquette (tag) pouvant être associée à des entités `Person` et `Organisation` dans le module carnet d'adresses.

Un tag permet de catégoriser ou de qualifier des personnes et organisations (exemples : « VIP », « Client », « Partenaire stratégique »).

Caractéristiques techniques :

- Entité JPA persistée dans une table
- Audit activé via Hibernate Envers
- Relations bidirectionnelles ManyToMany avec `Person` et `Organisation`
- Sérialisation JSON avec gestion d'identité pour éviter les boucles infinie

Utilisation :

Cette classe est utilisée pour gérer des catégories ou labels appliqués à des contacts et organisations dans l'application. Elle facilite le filtrage, la recherche et la classification des données.

getName

```
String getName()
```

Retourne le nom du tag.

Ce nom est utilisé pour identifier et afficher le tag dans l'interface utilisateur et dans les exports de données.

Paramètres

Aucun

Retour

Le nom du tag sous forme de chaîne.

Exemple

getOrganisations

```
Set<Organisation> getOrganisations()
```

Retourne la liste des organisations associées à ce tag.

Cette relation est bidirectionnelle et gérée par l'attribut `otags` dans la classe `Organisation`.

Paramètres

Aucun

Retour

Set<Organisation>

Un ensemble d'objets `Organisation`

Exemple

getPersons

```
Set<Person> getPersons()
```

Retourne la liste des personnes associées à ce tag.

Cette relation est bidirectionnelle et gérée par l'attribut `ptags` dans la classe `Person`.

Paramètres

Aucun

Retour

Set<Person>

Un ensemble d'objets `Person`

Exemple

setName

```
void setName(String name)
```

Définit le nom du tag.

Il est recommandé de valider que le nom n'est pas vide et qu'il respecte les conventions métier (ex. : longueur maximale, absence de caractères spéciaux).

Paramètres

name

Le nom du tag à enregistrer

Retour

Aucun

Exemple

setOrganisations

```
void setOrganisations(Set<Organisation> organisations)
```

Définit la liste des organisations associées à ce tag.

Paramètres

organisations

Ensemble d'objets `Organisation` à associer

Retour

Aucun.

Exemple

setPersons

```
void setPersons(Set<Person> persons)
```

Définit la liste des personnes associées à ce tag.

Paramètres

persons

Ensemble d'objets `Person` à associer

Retour

Aucun.

Exemple

Functions