Extensions serveur Gargantua

Carnet d'adresses

Table des matières

Description du module 5

Mode d'emploi 6

Extensions Gargantua 7

Extensions serveur 7

Extensions base de données 7

Extensions formulaires 7

Carnet d'adresses 7

Extensions processus 7

Scripting pour le module extension serveur Carnet d'adresses 8

Objets 8

Contact 8

Tag 8

Phone 8

Address 8

Assignment 8

Fonctions 8

AddressBook.newContact 8

AddressBook.newContact 9

AddressBook.createContact 9

AddressBook.readContact 9

AddressBook.updateContact 10

AddressBook.deleteContact 10

AddressBook.idOfEvent 10

AddressBook.enumerateEvents 11

AddressBook.newAddress 12

AddressBook.newAddress 12

AddressBook.newPhone 13

AddressBook.newPhone 13

AddressBook.newTag 14

AddressBook.newTag 14

AddressBook.createTag 14

AddressBook.readTag 15

AddressBook.updateTag 15

AddressBook.addTag 15
AddressBook.removeTag 16
AddressBook.deleteTag 16
AddressBook.idOfTag 16
AddressBook.enumerateTags 17
AddressBook.addAssignment 17
AddressBook.deleteAssignment 18
AddressBook.enumerateAssignments 18
Exemples 19
Création d'un contact de type personne physique, en utilisant des objets 19
Création d'un contact de type personne physique, en utilisant des propriétés 20
Création d'un contact de type personne morale, en utilisant des objets 22
Création d'un contact de type personne morale, en utilisant des propriétés 23
Modification d'un contact de type personne physique 24
Modification d'un contact de type personne morale 25
Suppression d'un contact 25
Récupération d'un contact 26
Liste des contacts 26
Création d'une affectation 27
Suppression d'une affectation 28
Création d'une étiquette 28
Mise à jour d'une étiquette 28
Suppression d'une étiquette 29
Liste des étiquettes 29
Liste des affectations d'une personne physique 30
Liste des affectations 31
Liste des affectations 32

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 serveur

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 serveur

Scripting pour le module extension serveur Carnet d'adresses

Le module extension serveur Carnet d'adresses propose désormais une API pour le scripting.

Objets

Contact

Tag

Phone

Address

Assignment

Fonctions

AddressBook.newContact

```
public Contact newContact()
```

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

JS call:

```
AddressBook.newContact()
```

Paramètres :

–

Retourne :

Un contact vide

Exceptions :

`SiatelException` - erreur gérée

AddressBook.newContact

```
public Contact newContact(DTO contactData)
```

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

JS call:

```
AddressBook.newContact(contactData)
```

Parameters :

`contactData` - données initiales pour le contact

Retourne :

Un objet contact initialisé

Exceptions :

`SiatelException` - erreur gérée

AddressBook.createContact

```
public void createContact(Contact contact)
```

Crée un nouveau contact permanent ; quand la fonction retourne l'id, l'objet contact est initialisé.

JS call:

```
AddressBook.createContact(contact)
```

Paramètres :

`contact` - un objet contact

Exceptions :

`SiatelException` - erreur gérée

AddressBook.readContact

```
public Event readContact(Integer contactId)
```

Remonte un contact spécifique depuis le stockage permanent.

JS call:

```
AddressBook.readContact(contactId)
```

Paramètres :

`contactId` - id de l'événement

Retourne :

L'objet contact

Exceptions :

`SiatelException` - erreur gérée

AddressBook.updateContact

```
public void updateContact(Contact contact)
```

Met à jour un contact donné dans le stockage permanent.

JS call:

```
AddressBook.updateContact(contact)
```

Paramètres :

`event` - Le contact à mettre à jour

Exceptions :

`SiatelException` - erreur gérée

AddressBook.deleteContact

```
public void deleteContact(Integer contactId)
```

Supprime un contact du stockage permanent.

JS call:

```
AddressBook.deleteContact(contactId)
```

Paramètres :

`contactId` - id du contact à supprimer

Exceptions :

`SiatelException` - erreur gérée

AddressBook.idOfEvent

```
public Integer idOfContact(String name)
```

Retrouve l'identifiant d'un contact à partir de son nom.

JS call:

```
AddressBook.idOfContact(name)
```

Paramètres :

`name` - Le nom du contact à rechercher

Retourne :

L'id du contact recherché

Exceptions :

`SiatelException` - erreur gérée

AddressBook.enumerateEvents

```
public void enumerateContacts(DTO criteria, List contacts)
```

Retrouve une liste de contacts qui correspondent aux critères.

JS call:

```
AddressBook.enumerateContacts(criteria, list)
```

Paramètres :

`criteria` - les critères de recherche

`list` - l'objet liste dans lequel seront renseignés les contacts

Exceptions :

`SiatelException` - erreur gérée

Les critères sont transmis à un DTO (objet de transfert de données), les clés suivantes sont disponibles :

tagIds

les id des tags à chercher

type

type de contact

typeCompany

nom de la personne morale

idCompany

identifiant de la personne morale

position

nom de la position de la personne physique

city

nom de la ville

province

nom de la province

country

nom du pays

email

email du contact

phone

numéro de téléphone

fax

numéro de fax

zipcode

code postal

AddressBook.newAddress

```
public Address newAddress()
```

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

JS call:

```
AddressBook.newAddress()
```

Paramètres :

–

Retourne :

Un objet adresse vide

Exceptions :

`SiatelException` - erreur gérée

AddressBook.newAddress

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

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

JS call:

```
AddressBook.newAddress(addressData)
```

Parameters :

`addressData` - données initiales pour le contact

Retourne :

Un objet adresse initialisé

Exceptions :

`SiatelException` - erreur gérée

AddressBook.newPhone

```
public Phone newPhone()
```

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

JS call:

```
AddressBook.newPhone()
```

Paramètres :

–

Retourne :

Un objet téléphone vide

Exceptions :

`SiatelException` - erreur gérée

AddressBook.newPhone

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

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

JS call:

```
AddressBook.newPhone(phoneData)
```

Parameters :

`phoneData` - données initiales pour le téléphone

Retourne :

Un objet téléphone initialisé

Exceptions :

`SiatelException` - erreur gérée

AddressBook.newTag

```
public Tag newTag()
```

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

JS call:

```
AddressBook.newTag()
```

Paramètres :

–

Retourne :

Un objet étiquette vide

Exceptions :

`SiatelException` - erreur gérée

AddressBook.newTag

```
public Tag newTagDTO tagData)
```

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

JS call:

```
AddressBook.newTag(tagData)
```

Parameters :

`tagData` - données initiales pour l'étiquette

Retourne :

Un objet étiquette initialisé

Exceptions :

`SiatelException` - erreur gérée

AddressBook.createTag

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

Crée une nouvelle étiquette permanente ; quand la fonction retourne l'id, l'objet étiquette est initialisé.

JS call:

```
AddressBook.createTag(tag)
```

Paramètres :

`tag` - un objet étiquette

Exceptions :

`SiatelException` - erreur gérée

AddressBook.readTag

```
public Tag readTag(Integer tagId)
```

Remonte une étiquette spécifique depuis le stockage permanent.

JS call:

```
AddressBook.readTag(tagId)
```

Paramètres :

`tagId` - id de l'étiquette

Retourne :

L'objet étiquette

Exceptions :

`SiatelException` - erreur gérée

AddressBook.updateTag

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

Met à jour une étiquette donnée dans le stockage permanent.

JS call:

```
AddressBook.updateTag(tag)
```

Paramètres :

`tag`- L'étiquette à mettre à jour

Exceptions :

`SiatelException` - erreur gérée

AddressBook.addTag

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

Sauvegarde un jeu d'étiquettes pour un contact donné dans le stockage permanent.

JS call:

```
AddressBook.addTag(contact, tagList)
```

Paramètres :

`contact` - Le contact pour lequel les étiquettes doivent être affectées

`tagList` - Liste d'étiquettes à mettre à jour

Exceptions :

`SiatelException` - erreur gérée

AddressBook.removeTag

```
public void removeTag(Contact contact, Tag tag)
```

Supprime une étiquette pour un contact donné dans le stockage permanent.

JS call:

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

Paramètres :

`contact` - Le contact pour lequel l'étiquette doivent être supprimée

`tag` - L'étiquette à supprimer

Exceptions :

`SiatelException` - erreur gérée

AddressBook.deleteTag

```
public void deleteTag(Integer tagId)
```

Supprime une étiquette spécifique depuis le stockage permanent.

JS call:

```
AddressBook.deleteTag(tagId)
```

Paramètres :

`tagId` - id de l'étiquette

Exceptions :

`SiatelException` - erreur gérée

AddressBook.idOfTag

```
public Integer idOfTag(String name)
```

Retrouve l'identification d'une étiquette à partir de son nom.

JS call:

```
AddressBook.idOfTag(name)
```

Paramètres :

`name` - Le nom de l'étiquette à rechercher

Retourne :

L'id de l'étiquette recherchée

Exceptions :

`SiatelException` - erreur gérée

AddressBook.enumerateTags

```
public void enumerateTags(DTO criteria, List tags)
```

Retrouve une liste d'étiquettes qui correspondent aux critères.

JS call:

```
AddressBook.enumerateTags(criteria, list)
```

Paramètres :

`criteria` - les critères de recherche

`list` - l'objet liste dans lequel seront renseignées les étiquettes

Exceptions :

`SiatelException` - erreur gérée

Les critères sont transmis à un DTO (objet de transfert de données), les clés suivantes sont disponibles :

tagIds

les id des tags à chercher

query

une requête particulière portant sur le nom de l'étiquette

AddressBook.addAssignment

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

Crée un lien entre une personne physique et une personne morale.

JS call:

```
AddressBook.addAssignment(assigData)
```

Paramètres :

`assigData` - Les données pour créer l'affectation de la personne physique vers la personne morale

Exceptions :

`SiatelException` - erreur gérée

Les paramètres pour l'objet assigData :

personId

l'identifiant de la personne physique

organizationId

l'identifiant de la personne morale

AddressBook.deleteAssignment

```
public void deleteAssignment(Integer contactId, Integer organizationId)
```

Supprime une affectation d'une personne physique vers une personne morale

JS call:

```
AddressBook.deleteAssignment(contactId, organizationId)
```

Paramètres :

`contactId` - identifiant de la personne physique
`organizationId` - identifiant de la personne morale

Exceptions :

`SiatelException` - erreur gérée

AddressBook.enumerateAssignments

```
public void enumerateAssignments(DTO criteria, List assignments)
```

Retrouve une liste d'affectations qui correspondent aux critères.

JS call:

```
AddressBook.enumerateAssignments(criteria, list)
```

Paramètres :

`criteria` - les critères de recherche
`list` - l'objet liste dans lequel seront renseignées les affectations

Exceptions :

`SiatelException` - erreur gérée

Les critères sont transmis à un DTO (objet de transfert de données), les clés suivantes sont disponibles :

personId

un identifiant d'une personne physique pour laquelle les affectations sont remontées

organizationId

un identifiant d'une personne morale pour laquelle les affectations sont remontées

Exemples

Création d'un contact de type personne physique, en utilisant des objets

```
function createNaturalContact() {
    // natural person
    var TYPE_NATURAL_PERSON = 1;

    var contact = AddressBook.newContact();
    contact.setType(TYPE_NATURAL_PERSON);
    contact.setDateCreation(new java.util.Date());

    contact.setUrl_1( »http://www.siatel.com »);
    contact.setUrl_2( »https://www.linkedin.siatel.com »);
    contact.setNote( »note contact : Jean DUPONT »);

    contact.setSurname( »Jean »);
    contact.setName( »DUPONT »);
    contact.setTypeCompany(null);

    contact.setTitle( »Mr. »);
    contact.setProfTitle( »ab.contact.proftitle.1 »);

    contact.setIsNPAI(0);
```

```
    contact.setIsVIP(0);
    contact.setIsDead(0);

    contact.setPersonalEmail( »jean.dupont@somewhere.net »);
    contact.setPersonalMobile( »+40756443322 »);
    contact.setPersonalPhone( »0250887766 »);
    contact.setPersonalFax( »0250887766 »);

    var homeAddress = AddressBook.newAddress();
    homeAddress.setStreetAddress( »Street Blv. Unirii »);
    homeAddress.setStreetAddress2( »No 53 »);
    homeAddress.setStreetAddress3( »022698 »);
    homeAddress.setZipCode( »001222 »);
    homeAddress.setCity( »Bucharest »);
    homeAddress.setProvince( »District 1 »);
    homeAddress.setCountry( »Romania »);

    contact.setHomeAddress(homeAddress);

    AddressBook.createContact(contact);

    return « Contact name : «  + contact.getName();
}
```

Création d'un contact de type personne physique, en utilisant des propriétés

```
function createNaturalContact2() {
    // natural person
    var TYPE_NATURAL_PERSON = 1;

    var phoneList = new java.util.ArrayList(0);
    phoneList.add(DTOFactory.newDTO("number", "+40767675645", "type", "0"));
    phoneList.add(DTOFactory.newDTO("number", "+40727675645", "type", "1"));
```

```
    var contact = AddressBook.newContact(DTOFactory.newDTO(
          "type",  TYPE_NATURAL_PERSON,
          "surname",  "Jean" ,
          "name",  "DUPONT" ,
          "title",  "Mr." ,
          "professionTitle",  "ab.contact.proftitle.2" ,
          "url",  "http://www.siatel.com" ,
          "linkedin",  "https://www.linkedin.siatel.com" ,
          "note", note,
          "isNPAI", isNPAI,
          "isVIP", isVIP,
          "isDead", isDead,
          "personalEmail",  "jean.dupont@somewhere.net" ,
          "personalMobile",  "+40756443322" ,
          "personalPhone",  "0250887766" ,
          "personalFax",  "0250887766" ,
          "streetAddress",  "Street Blv. Unirii" ,
          "streetAddress2",  "No 53" ,
          "streetAddress3",  "022698" ,
          "zipCode",  "001222" ,
          "city",  "Bucharest" ,
          "province",  "District 1" ,
          "country",  "Romania" ,
          "phones", phoneList));

    AddressBook.createContact(contact);

    return "Contact name : " + contact.getName();
}
```

Création d'un contact de type personne morale, en utilisant des objets

```
function createLegalContact() {
    // legal person
    var TYPE_LEGAL_PERSON = 2;

    var contact = AddressBook.newContact();
    contact.setSurname("PHILIPS");
    contact.setTypeCompany("electronics");
    contact.setIdCompany("PHILIPS_1");

    contact.setType(TYPE_LEGAL_PERSON);
    contact.setDateCreation(new java.util.Date());

    contact.setUrl_1("http://www.philipsromania.com");
    contact.setUrl_2("https://www.linkedin.philipsromania.com");
    contact.setNote("note contact : PHILIPS");
    contact.setIsNPAI(0);
    contact.setTitle(null);

    contact.setService_1("Service 1 : PHILIPS ROMANIA");
    contact.setService_2("Service 2 : PHILIPS FRANCE");

    contact.setWorkEmail("philips.electro@yahoo.com");
    contact.setWorkMobile("+40756443322");
    contact.setWorkPhone("0250887766");
    contact.setWorkFax("0250887766");

    var workAddress = AddressBook.newAddress();
    workAddress.setStreetAddress("Street Blv. Unirii");
    workAddress.setStreetAddress2("No 53");
    workAddress.setStreetAddress3("022698");
    workAddress.setZipCode("001122");
    workAddress.setCity("Bucharest");
```

```
    workAddress.setProvince("District 1");
    workAddress.setCountry("Romania");

    contact.setWorkAddress(workAddress);

    AddressBook.createContact(contact);

    return "Contact surname : " + contact.getSurname();
}
```

Création d'un contact de type personne morale, en utilisant des propriétés

```
function createLegalContact2() {
    // legal person
    var TYPE_LEGAL_PERSON = 2;

     var phoneList = new java.util.ArrayList(0);
    phoneList.add(DTOFactory.newDTO("number", "+40767675645", "type", "0"));
    phoneList.add(DTOFactory.newDTO("number", "+40727675645", "type", "1"));

    var contact = AddressBook.newContact(DTOFactory.newDTO(
          "type",  TYPE_LEGAL_PERSON,
          "organizationName",  "PHILIPS" ,
         "typeCompany",  "electronics",
         "idCompany", "PHILIPS_2",
         "isNPAI", "false" ,
          "url", "http://www.philipsromania.com" ,
          "linkedin", "https://www.linkedin.philipsromania.com" ,
          "note", "note contact : PHILIPS" ,
          "service", "Service 1 : PHILIPS ROMANIA" ,
         "service2", "Service 2 : PHILIPS FRANCE",
         "workEmail", "george.mihai@yahoo.com",
         "workMobile", "+40756443322",
```

```
          "workPhone", "0250887766" ,
           "workFax", "0250887766" ,
           "streetAddress", "Street Blv. Unirii" ,
          "streetAddress2", "No 53",
          "streetAddress3", "022698",
          "zipCode", "001222",
          "city", "Bucharest",
          "province", "District 1",
          "country", "Romania" ,
           "phones", phoneList));

    AddressBook.createContact(contact);

    return "Contact : organization name : " + contact.getSurname();
}
```

MODIFICATION D'UN CONTACT DE TYPE PERSONNE PHYSIQUE

```
function updateNaturalContact() {
    var name = «  »;
    var id = 16;

    var sdf = new java.text.SimpleDateFormat( »dd/MM/yyyy HH:mm »);
    var today = sdf.format(new java.util.Date());

    try {
          var contact = AddressBook.readContact(id);

          contact.setName( »UPDATE_NAME »);
          contact.setNote( »UPDATE_NOTE_ » + today);

          AddressBook.updateContact(contact);

          name = contact.getName();
```

```
     } catch (e) {
     }

    return « Natural Contact : «  + name;
}
```

Modification d'un contact de type personne morale

```
function updateLegalContact() {
    var surname = "";
    var id = 15;

    var sdf = new java.text.SimpleDateFormat("dd/MM/yyyy HH:mm");
    var today = sdf.format(new java.util.Date());

    try {
          var contact = AddressBook.readContact(id);

          contact.setTypeCompany("electronics_RO");
          contact.setNote("UPDATE_NOTE_" + today);

          AddressBook.updateContact(contact);

          surname = contact.getSurname();
    } catch (e) {
     }

    return "Legal Contact : " + surname;
}
```

Suppression d'un contact

```
function deleteContact() {
```

```
    var id = 9;

    try {
          AddressBook.deleteContact(id);
     } catch (e) {
     }
}
```

Récupération d'un contact

```
function getContactId() {
    var surname = « CONTACT_NATURAL_BL »;

    try {
          var id = AddressBook.idOfContact(surname);
          return « Id contact : «  + id;
     } catch (e) {
     }
}
```

Liste des contacts

```
function enumerateContacts() {
    var contacts = new java.util.ArrayList(0);
    var naturalContact = « NATURAL CONTACT »;
    var legalContact = « LEGAL CONTACT »;

    var dto = DTOFactory.newDTO();

    var json = '({ « contactList » : [ ';

    try {
          AddressBook.enumerateContacts(dto, contacts);
          if (contacts != null && !contacts.isEmpty()) {
```

```
                for (var i = 0; i < contacts.size(); i++) {
                     var contact = contacts.get(i);
                     var contactType = contact.getType();

                     if (i > 0) {
                          json = json + ', ';
                     }
                     json = json + '{ « contactId » : « ' + contact.getId()
                          + '« , « contactSurname » : « ' + contact.getSurname()
                          + '« , « contactType » : « '
                          + (contactType == 1 ? naturalContact : legalContact)
                          + '« }';
                }
           }
     } catch (e) {
      }

      json = json + ' ] })';
      return json;
}
```

CRÉATION D'UNE AFFECTATION

```
function addAssignment() {
    var dto = DTOFactory.newDTO(
            »personId », 16,
            »organizationId », 8,
            »department », « software »,
            »position », « engineer »,
            »mobile »,   »+40756443322 »,
            »phone », « 0250887766 »,
            »fax », « 0250887766 »,
            »email », « siatel.software@gmail.com »);
```

```
    AddressBook.addAssignment(dto);
}
```

Suppression d'une affectation

```
function deleteAssignment() {
    var personId = 16;
    var organizationId = 8;

    AddressBook.deleteAssignment(personId, organizationId);
}
```

Création d'une étiquette

```
function createTag() {
    var sdf = new java.text.SimpleDateFormat( »dd/MM/yyyy HH:mm »);
    var name = sdf.format(new java.util.Date());

    var tag = AddressBook.newTag();
    tag.setName( »TAG_ » + name);

    AddressBook.createTag(tag);

    return « Id tag : «  + tag.getId();
}
```

Mise à jour d'une étiquette

```
function updateTag() {
    var sdf = new java.text.SimpleDateFormat("dd/MM/yyyy HH:mm");
    var today = sdf.format(new java.util.Date());
    var title = "UPDATE_TAG_" + today;
```

```
    var id = 13;

    try {
          var tag = AddressBook.readTag(id);
          tag.setName(title);

          AddressBook.updateTag(tag);
     } catch (e) {
     }

    return "Id tag : " + id;
}
```

Suppression d'une étiquette

```
function removeTag() {
    var idContact = 14;
    var idTag = 12;

    var contact = AddressBook.readContact(idContact);
    var tag = AddressBook.readTag(idTag);

    try {
          AddressBook.removeTag(contact, tag);
     } catch (e) {
     }
}
```

Liste des étiquettes

```
function enumerateTags() {
```

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

    var json = '({ « tagList » : [ ';

    try {
          AddressBook.enumerateTags(null, 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;
}
```

LISTE DES AFFECTATIONS D'UNE PERSONNE PHYSIQUE

```
function enumerateAssignments() {
    var personId = 16;
     var dto = DTOFactory.newDTO( »personId », personId);
     var assignments = new java.util.ArrayList(0);
```

```
    var json = '({ « assignmentsList » : [ ';

    try {
          AddressBook.enumerateContactsAssignments (dto,  assignments );

          if ( assignments != null && ! assignments .isEmpty()) {
               for (var i = 0; i <  assignments .size(); i++) {
                    var assignment=  assignments .get(i);
                    var contactType =  assignment .getType();

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

                    json = json + '{ « contactId » : « ' + contact.getId()
                         + '« , « contactSurname » : « ' + contact.getSurname()
                         + '« , « contactType » : « ORGANIZATION »}';
               }
          }
    } catch (e) {
     }

    json = json + ' ] })';

    return json;
}
```

LISTE DES AFFECTATIONS

```
function enumerateAssignments() {
    var personId = 16;
     var dto = DTOFactory.newDTO( »personId », personId);
     var assigments = new java.util.ArrayList(0);
```

```
     var json = '({ « assignmentsList » : [ ';
     try {
          AddressBook.enumerateAssignments(dto, assigments);
          if (assigments != null && !assigments.isEmpty()) {
               for (var i = 0; i < assigments.size(); i++) {
                    var assigment = assigments.get(i);
                    if (i > 0) {
                         json = json + ', ';
                    }
                    json = json + '{ « personId » : « ' + assigment.getPersonId()
                         + '« , « organizationId » : « ' +
assigment.getOrganizationId()
                         + '« , « department » : « ' + assigment.getDepartment()
                         + '« , « position » : « ' + assigment.getPosition()
                         + '« , « mobile » : « ' + assigment.getMobile()
                         + '« , « phone » : « ' + assigment.getPhone()
                         + '« , « fax » : « ' + assigment.getFax()
                         + '« , « email » : « ' + assigment.getEmail() + '«  }';
               }
          }
     } catch (e) {
      }

     json = json + ' ] })';
     return json;
}
```

LISTE DES AFFECTATIONS

```
function enumerateAssignments2() {
     var organizationId = 8;
     var dto = DTOFactory.newDTO( »organizationId », organizationId);
     var assigments = new java.util.ArrayList(0);
     var json = '({ « assignmentsList » : [ ';
     try {
```

```
          AddressBook.enumerateAssignments(dto, assigments);
          if (assigments != null && !assigments.isEmpty()) {
               for (var i = 0; i < assigments.size(); i++) {
                    var assigment = assigments.get(i);
                    if (i > 0) {
                         json = json + ', ';
                    }
                    json = json + '{ « personId » : « ' + assigment.getPersonId()
                         + '« , « organizationId » : « ' +
assigment.getOrganizationId()
                         + '« , « department » : « ' + assigment.getDepartment()
                         + '« , « position » : « ' + assigment.getPosition()
                         + '« , « mobile » : « ' + assigment.getMobile()
                         + '« , « phone » : « ' + assigment.getPhone()
                         + '« , « fax » : « ' + assigment.getFax()
                         + '« , « email » : « ' + assigment.getEmail() + '«  }';
               }
          }
     } catch (e) {
      }

      json = json + ' ] })';
      return json;
}
```