Bases Gargantua - Éditeur de scripts

Outil d'administration

TABLE DES MATIÈRES

INTRODUCTION

L'Outil d'administration GARGANTUA permet d'élaborer des scripts afin d'obtenir des fonctionnalités personnalisées.

Il existe quatre types de scripts :

- les scripts de formulaires ;
- les scripts de processus ;
- les scripts de tâches IntelliScan ;
- les autres scripts.

Les scripts de formulaires utilisent la syntaxe JavaScript. Les scripts de processus utilisent la syntaxe JavaScript et des objets Java. Ce document s'adresse à des personnes ayant les connaissances de programmation nécessaires dans ces langages.

Créer un script au niveau d'une base Gargantua

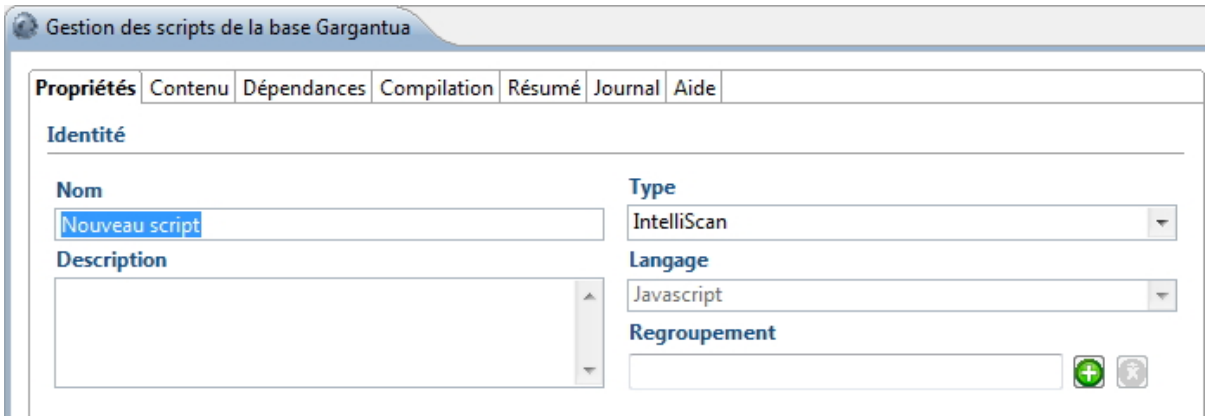
Pour créer un script centralisé au niveau d'une base Gargantua, procédez de comme suit :

1. Dans le volet gauche de l'Outil d'administration, déployez la sous-branche correspondant à la base Gargantua dans laquelle vous souhaitez créer le script.
2. Cliquez sur la sous-branche **Scripts**.

   L'écran **Gestion des scripts de la base Gargantua** s'affiche dans le volet droit.

3. En bas à gauche du volet droit, cliquez sur le bouton **Ajouter**.

   Un nouveau script est créé et son écran de gestion s'affiche dans le volet droit.



4. Dans l'onglet **Propriétés**, indiquez un nom, éventuellement une description et un regroupement pour le script.
5. Sélectionnez un type de script.
6. Dans l'onglet **Contenu**, saisissez le code du script.
7. En bas du volet droit, cliquez sur le bouton **Compiler** pour voir si votre script contient des erreurs.

   L'onglet **Compilation** s'ouvre.

8. Si l'onglet **Compilation** contient des messages d'erreur ou d'avertissement, corrigez le script : vous ne pouvez pas publier un script contenant des erreurs. Pour corriger une erreur, cliquez sur lien correspondant à l'erreur : vous êtes redirigé vers la ligne du script où se trouve l'erreur.

   Si l'onglet **Compilation** ne contient pas de messages d'erreur ou d'avertissement, passez à l'étape suivante de la présente procédure.

9. En bas du volet droit, cliquez sur le bouton **Publier**.

   Le script est maintenant disponible pour la base Gargantua :

   - Les scripts de type **Workflow** sont disponibles dans l'onglet **Script** de l'écran de **Gestion des processus** de la base.

- Les scripts de type **Formulaire** sont disponibles dans l'onglet **Scripts** de l'écran de **Gestion des formulaires** de la base.

Écran de gestion des scripts - Onglets disponibles

L'écran **Gestion des scripts**, recensant tous les scripts de la base GARGANTUA actuelle, contient les onglets suivants :

- L'onglet **Propriétés** permet d'afficher les caractéristiques de base du script : nom, description, regroupement, type de script et historique.
- L'onglet **Contenu** contient un éditeur de script permettant de consulter et de modifier le script.
- L'onglet **Dépendances** permet d'afficher les objets utilisés et qu'utilise le script.
- L'onglet **Compilation** permet d'afficher les éventuelles erreurs présentes dans le script.
- L'onglet **Résumé** permet d'afficher un résumé d'un script.
- L'onglet **Journal** permet de consulter la liste de toutes les opérations effectuées sur le script, en les filtrant éventuellement en fonction de certains critères.
- L'onglet **Aide** permet d'accéder à l'aide contextuelle.

Onglet « Propriétés »

L'onglet **Propriétés** permet de nommer ou renommer un script (champ **Nom**), de rédiger une description (champ **Description**), de l'ajouter à un regroupement (champ **Regroupement**), de définir un type de script (liste déroulante **Type**) et de consulter l'historique de ses versions publiées (tableau **Historique des versions publiées**).

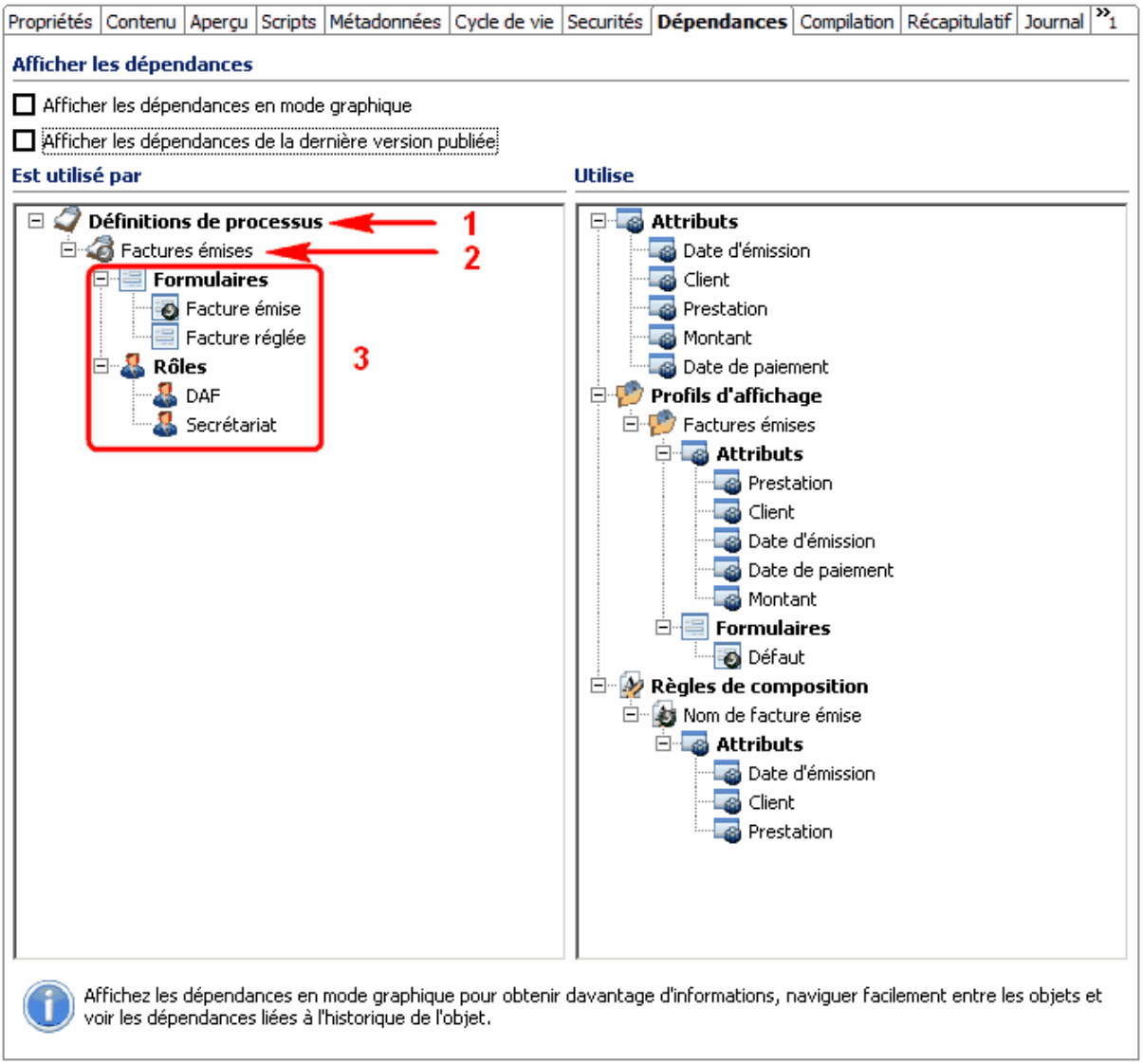
Onglet « Dépendances »

L'onglet **Dépendances** permet d'**analyser les dépendances** de chaque objet, c'est-à-dire de savoir quels sont les autres objets qui l'utilisent et/ou dont il dépend.

Affichage des dépendances en mode arborescence

Par défaut, GARGANTUA affiche les dépendances en **mode arborescence** et en tenant compte de la **version de travail** des différents objets concernés.

Selon que l'objet courant est utilisé par et/ou dépend d'autres objets, l'affichage des dépendances en mode arborescence s'effectue en une ou deux colonnes. Dans le second cas, les objets qui dépendent de l'objet courant sont listés à gauche et les objets qui sont utilisés par l'objet courant, à droite :



1. Catégorie d'objets dont certains entretiennent des dépendances avec l'objet courant ;
2. Nom d'un objet lié à l'objet courant : notre formulaire courant « Facture émise » est utilisé par le processus « Factures émises » ;
3. Ensemble des objets dont dépend le processus « Factures émises ».

Dans les écrans d'analyse des dépendances comme dans le reste de l'interface, l'icône permet de savoir si l'objet concerné est actuellement publié et, si tel est le cas, si la version actuellement publiée coïncide avec la version de travail :

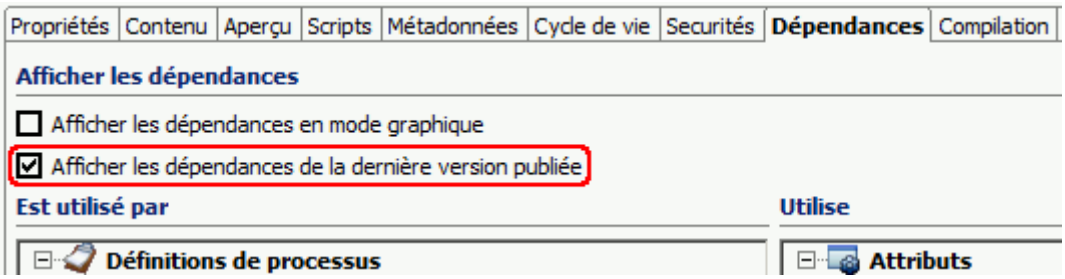
- Icône absente – L'objet n'est pas publié ;
- - La version de travail et la version actuellement publiée sont identiques ;

- - La version de travail et la version actuellement publiée sont différentes.

À QUOI SERT L'OPTION « AFFICHER LES DÉPENDANCES DE LA DERNIÈRE VERSION PUBLIÉE » ?

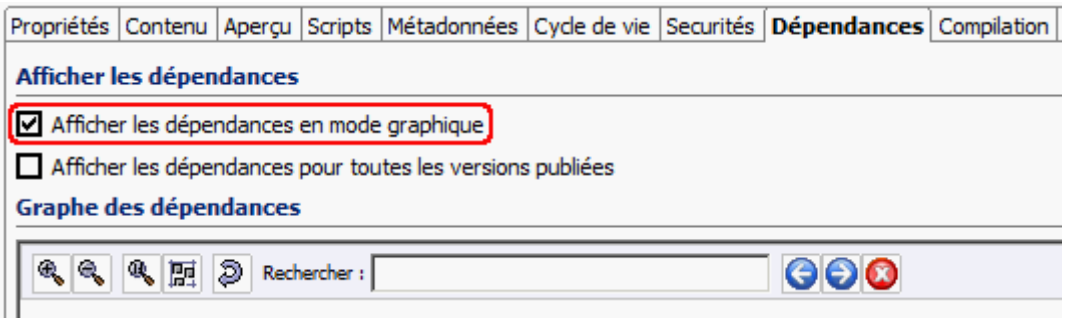
En cochant la case **Afficher les dépendances de la dernière version publiée**, vous pouvez afficher les dépendances de la version de l'objet actuellement publiée au lieu de celles de la version de travail, prise en compte par défaut.

Cette option n'est disponible qu'en mode arborescence, pour les objets qui disposent d'un historique des versions publiées :

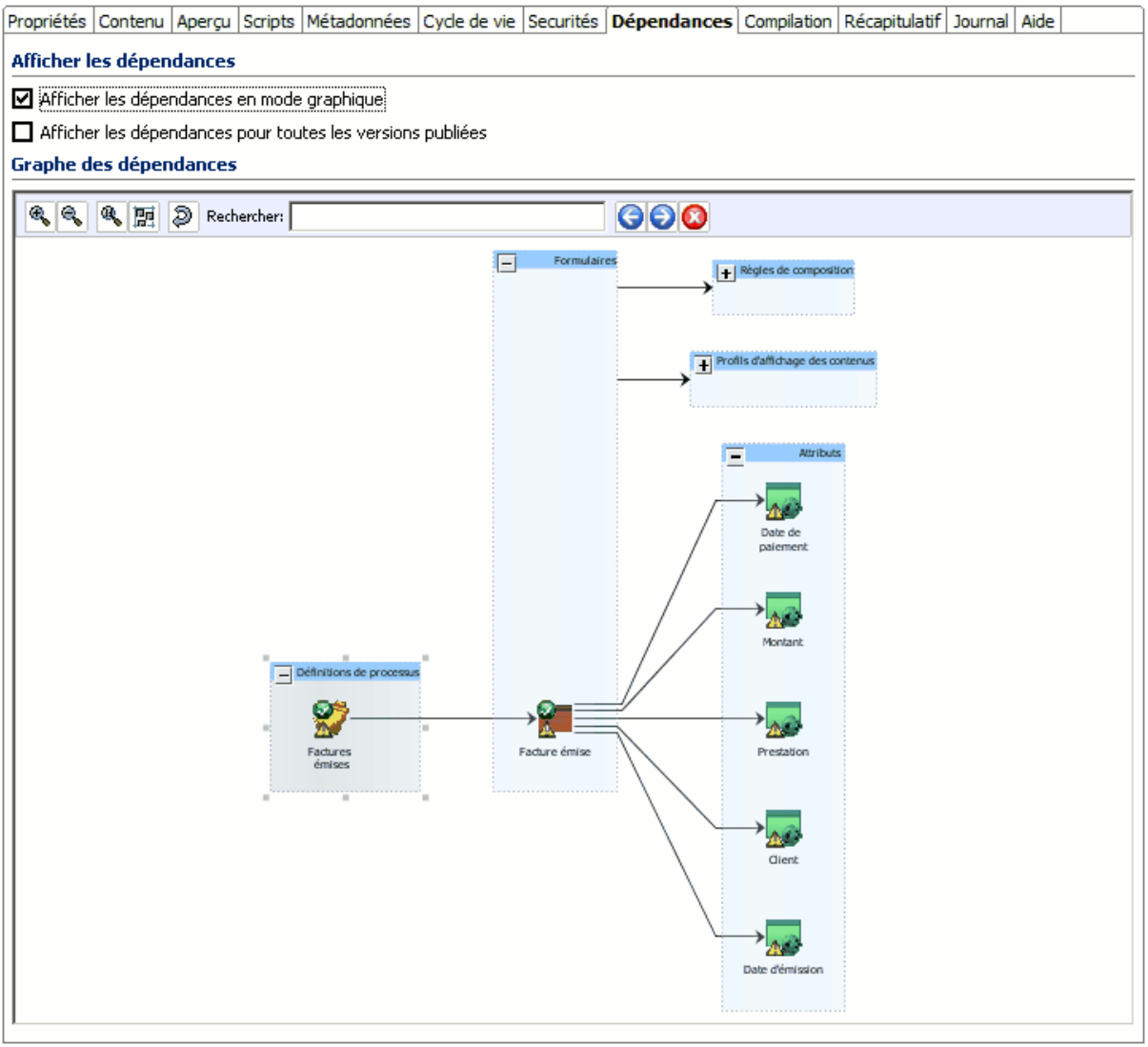


AFFICHAGE DES DÉPENDANCES EN MODE GRAPHIQUE

Pour basculer de l'affichage des dépendances en mode arborescence vers leur affichage en mode graphique, cochez la case **Afficher les dépendances en mode graphique** :



Les dépendances entre les versions de travail des différents objets concernés s'affichent alors sous la forme d'un schéma :



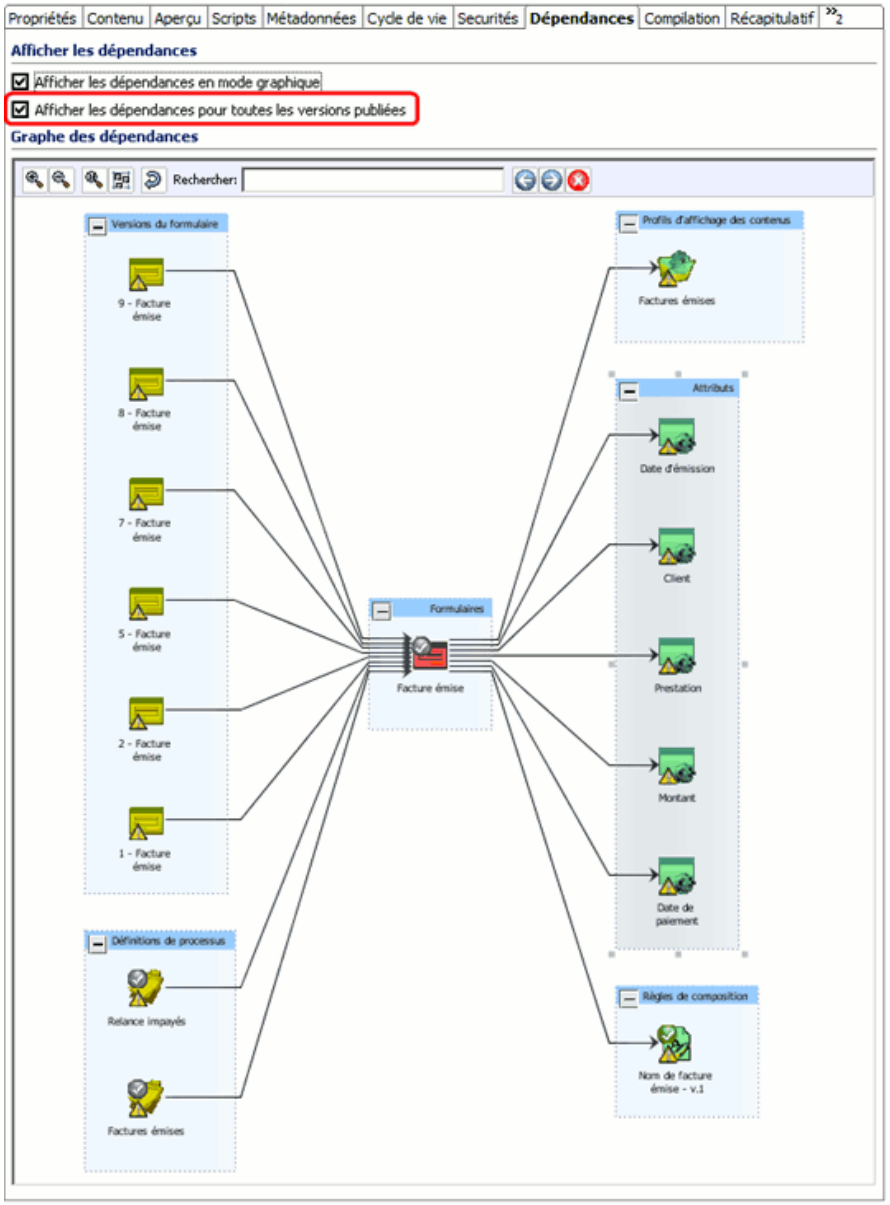
À l'aide de la barre d'outils de l'affichage graphique, vous pouvez :

- Zoomer pour agrandir le graphique ;
- Dézoomer pour rétrécir le graphique ;
- Ramener le graphique à sa taille d'origine ;
- Ajuster le graphique à la taille de l'écran pour optimiser l'espace disponible ;
- Rafraîchir le graphique ;
- Rechercher: pai Rechercher une chaîne de caractères dans les noms des objets affichés ; les objets concernés se mettent alors en surbrillance en bleu clair ;
- / Naviguer entre les occurrences de la chaîne de caractères recherchée ;
- Annuler la mise en surbrillance des objets dont le nom contient la chaîne de caractères recherchée.

À QUOI SERT L'OPTION « AFFICHER LES DÉPENDANCES POUR TOUTES LES VERSIONS PUBLIÉES » ?

En cochant la case **Afficher les dépendances pour toutes les versions publiées**, vous pouvez afficher le graphe des dépendances entre les **versions publiées** successives des différents objets concernés, au lieu des dépendances entre les **versions de travail** actuelles. Si l'objet courant est lui-même doté d'un historique des versions publiées, la liste de ses versions publiées apparaît également dans le graphe.

Cette option n'est disponible qu'en mode graphique :



ONGLET « COMPILATION »

L'onglet **Compilation** permet de diagnostiquer et de corriger les éventuels problèmes avant la publication de l'objet. Cette fonction est également disponible au niveau de la base GARGANTUA, afin de compiler l'ensemble des objets qu'elle contient.

QUAND ET COMMENT UTILISER LE COMPILATEUR GARGANTUA ?

Avant de publier ou de republier des objets modélisés, vous devez les compiler afin de diagnostiquer et de résoudre au préalable les éventuels problèmes.

Pour ce faire :

1. Cliquez sur le nom de l'objet à publier dans le volet gauche de l'Outil d'administration - ou sur le nom de la base si vous souhaitez publier tous les objets qu'elle contient.

   L'écran de gestion correspondant s'affiche.

2. En bas du volet droit de l'Outil d'administration, cliquez sur le bouton **Compiler**.

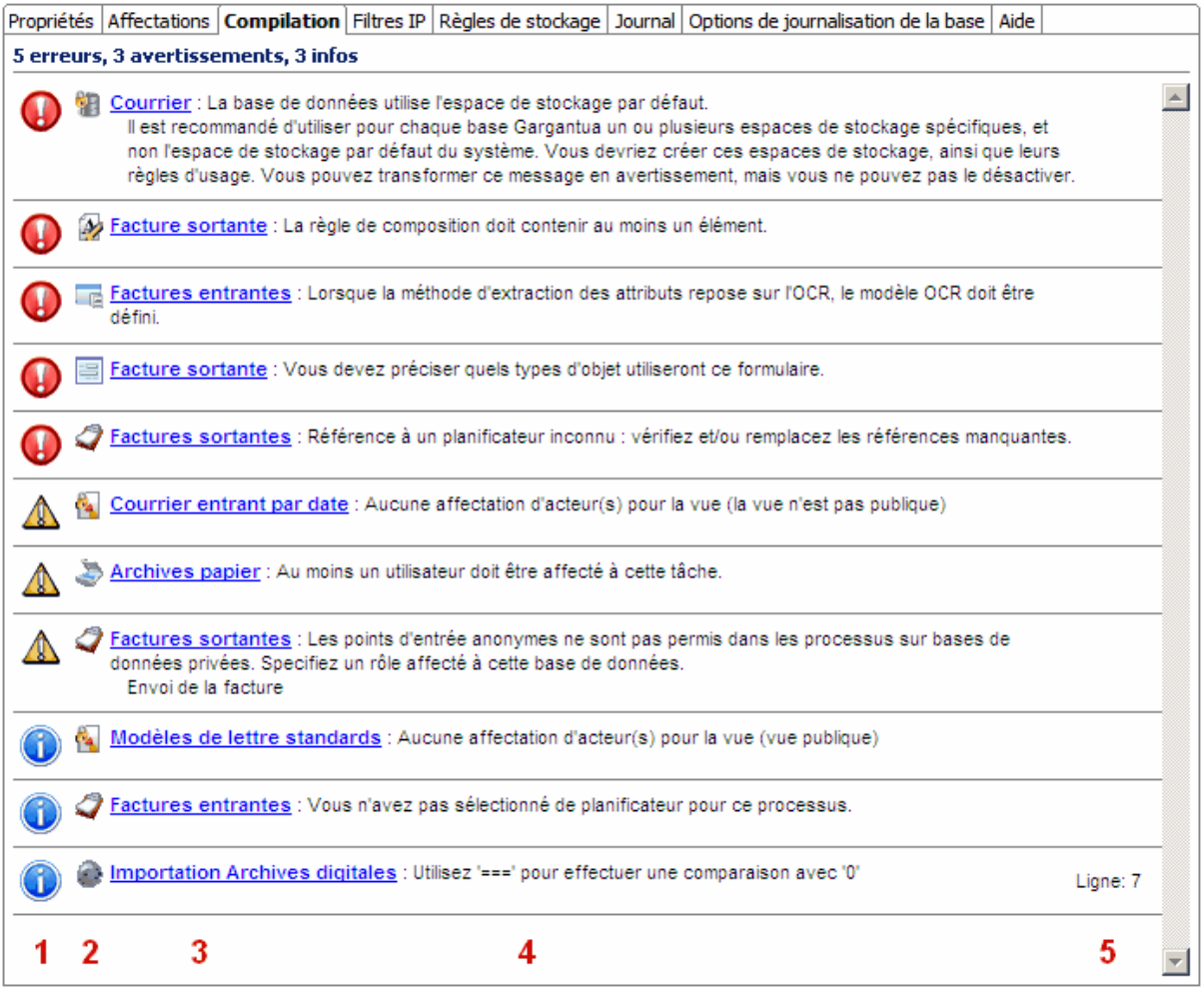
   La compilation s'effectue et son résultat s'affiche sous l'onglet **Compilation**.

3. Si la liste des messages du compilateur ne comprend que des messages de succès et/ou d'information, vous pouvez procéder à la publication.

   Si la liste comprend des messages d'erreur et/ou d'avertissement, vous devez d'abord corriger ces problèmes.

LISTE DES MESSAGES

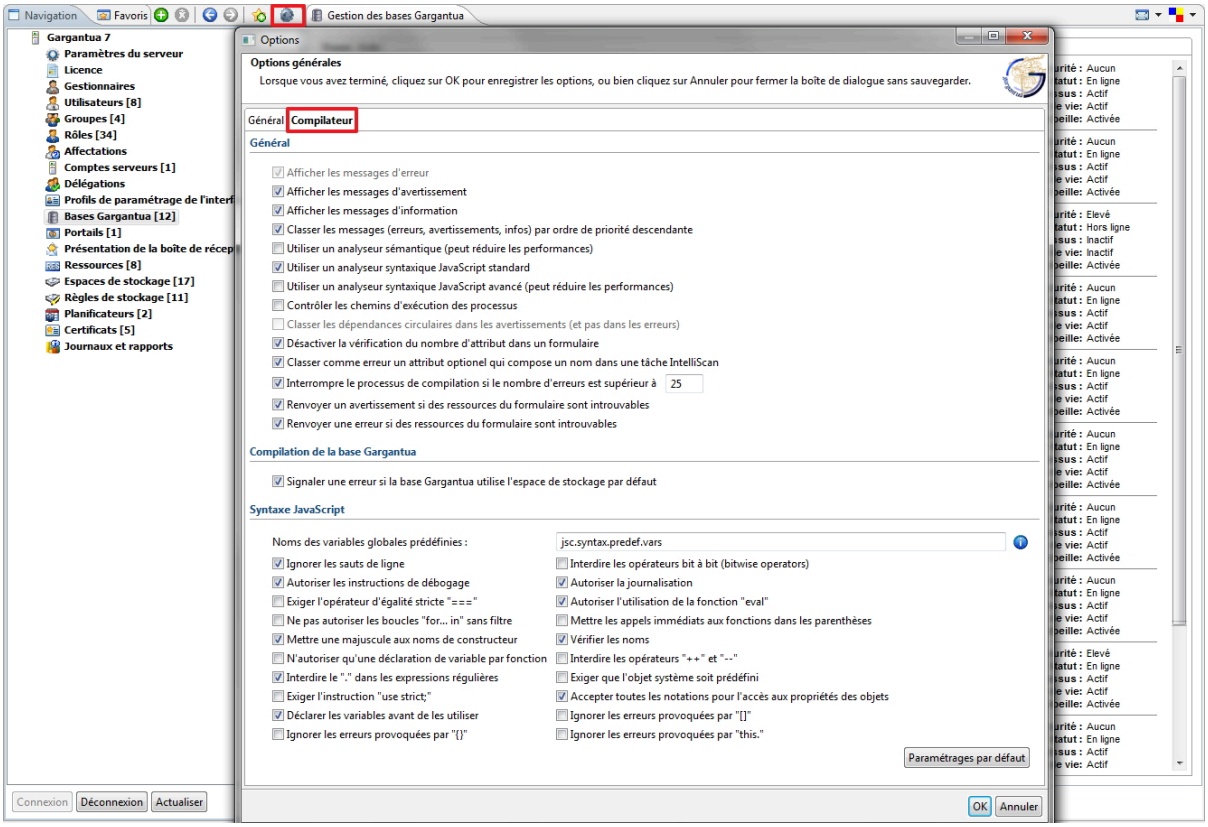
La liste des messages du compilateur se présente ainsi :



1. Icônes indiquant le type de chaque message :
   - Message de succès ;
   - Message d'information ;
   - Message d'avertissement ;
   - Message d'erreur ;
2. Icônes indiquant le type de l'objet concerné par le message ;
3. Liens permettant d'accéder à l'écran de gestion de l'objet concerné par le message ;
4. Messages détaillés ;
5. Si l'objet est un script, ligne concernée par le message.

PERSONNALISER LE COMPILATEUR

Si vous le souhaitez, vous pouvez personnaliser l'affichage de la liste des messages et les paramètres de syntaxe JavaScript du compilateur GARGANTUA, à l'aide de la boîte de dialogue **Options** de l'Outil d'administration :



Pour en savoir plus sur les différentes options disponibles, reportez-vous au chapitre « Généralités – Connexion serveur » du présent manuel.

ONGLET « RÉSUMÉ »

L'onglet **Résumé** permet d'afficher un résumé d'un objet, qui présente ses principales caractéristiques, comme son gestionnaire, le formulaire qui lui est lié, sa priorité, son contenu, etc.

L'onglet **Résumé** est particulièrement pratique pour avoir un aperçu général de l'objet en question, sans avoir à parcourir chaque onglet en détail.

L'onglet **Résumé** dispose en outre d'une option d'enregistrement en format PDF, qui permet donc de visualiser le résumé d'un objet sans avoir l'Outil d'administration GARGANTUA d'ouvert.

ONGLET « JOURNAL »

L'onglet **Journal** permet de visualiser la liste des opérations effectuées sur une des rubriques de l'Outil

d'Administration GARGANTUA.

Vous pouvez filtrer votre recherche grâce aux critères suivants :

- Date de début, date de fin ;
- Nom d'utilisateur ;
- Opération.

Pour filtrer par dates :

1. Renseignez la date à partir de laquelle vous souhaitez visualiser les opérations effectuées, dans le champ **Date de début**.
2. Renseignez la date jusqu'à laquelle vous souhaitez visualiser les opérations effectuées, dans le champ **Date de fin**.
3. Cliquez sur **Afficher**.

   La liste des opérations effectuées entre les deux dates renseignées s'affiche dans le champ **Entrées du journal**.

> ***Vous pouvez également cliquer sur les icônes « De la semaine », « Du mois », « De l'année » ou « Complet », afin de visualiser les opérations effectuées sur des périodes plus vastes.***

Pour filtrer par nom d'utilisateur :

1. Cliquez sur la loupe, à côté du champ **Utilisateur**.
2. Renseignez le nom de l'utilisateur pour lequel vous souhaitez visualiser les opérations effectuées.
3. Cliquez sur **Afficher**.

   La liste des opérations effectuées par l'utilisateur renseigné s'affiche dans le champ Entrées du journal.

Pour filtrer par opération :

1. Cliquez sur la flèche du champ **Opération**.
   Une liste des différentes opérations s'affiche.
2. Cliquez sur le type d'opération souhaité.
3. Cliquez sur **Afficher**.

   La liste des opérations effectuées en rapport avec le type d'opération renseigné s'affiche dans le champ **Entrées du journal**.

*Vous pouvez cocher la case « **Montrer les connexions/déconnexions** » pour afficher les connexions et déconnexions des utilisateurs. Cette fonctionnalité n'est disponible que pour les branches « **Utilisateurs** » et « **Journal du serveur** ».*

*Vous pouvez cocher la case « **Afficher seulement les audits d'erreur** », pour n'afficher que les opérations échouées.*

*Le bouton « **Remettre à zéro** » vous permet d'annuler le dernier filtrage.*

Vous pouvez aussi exporter la liste des opérations effectuées, en format .csv.

Pour exporter une liste des opérations effectuées :

1. Générez une liste, en fonction des critères cités précédemment.
2. Cliquez sur le bouton **Exportation CSV**.
3. Choisissez l'endroit où vous souhaitez enregistrer l'export.
4. Cliquez sur **Sauvegarder**.

   Vous avez exporté une liste des opérations effectuées.

Scripts de formulaires

Qu'est-ce qu'un script de formulaire ?

Les formulaires dans Gargantua 7 sont générés en HTML.

Un formulaire Gargantua est un ensemble d'attributs dont la présentation peut être personnalisée. Les formulaires sont conçus à partir de l'Outil d'administration Gargantua 7.

Chaque attribut est identifié d'une manière unique dans le formulaire. La forme de l'identificateur est : `DFATBBBBNNNNNNNN00000000000000000000000000` :

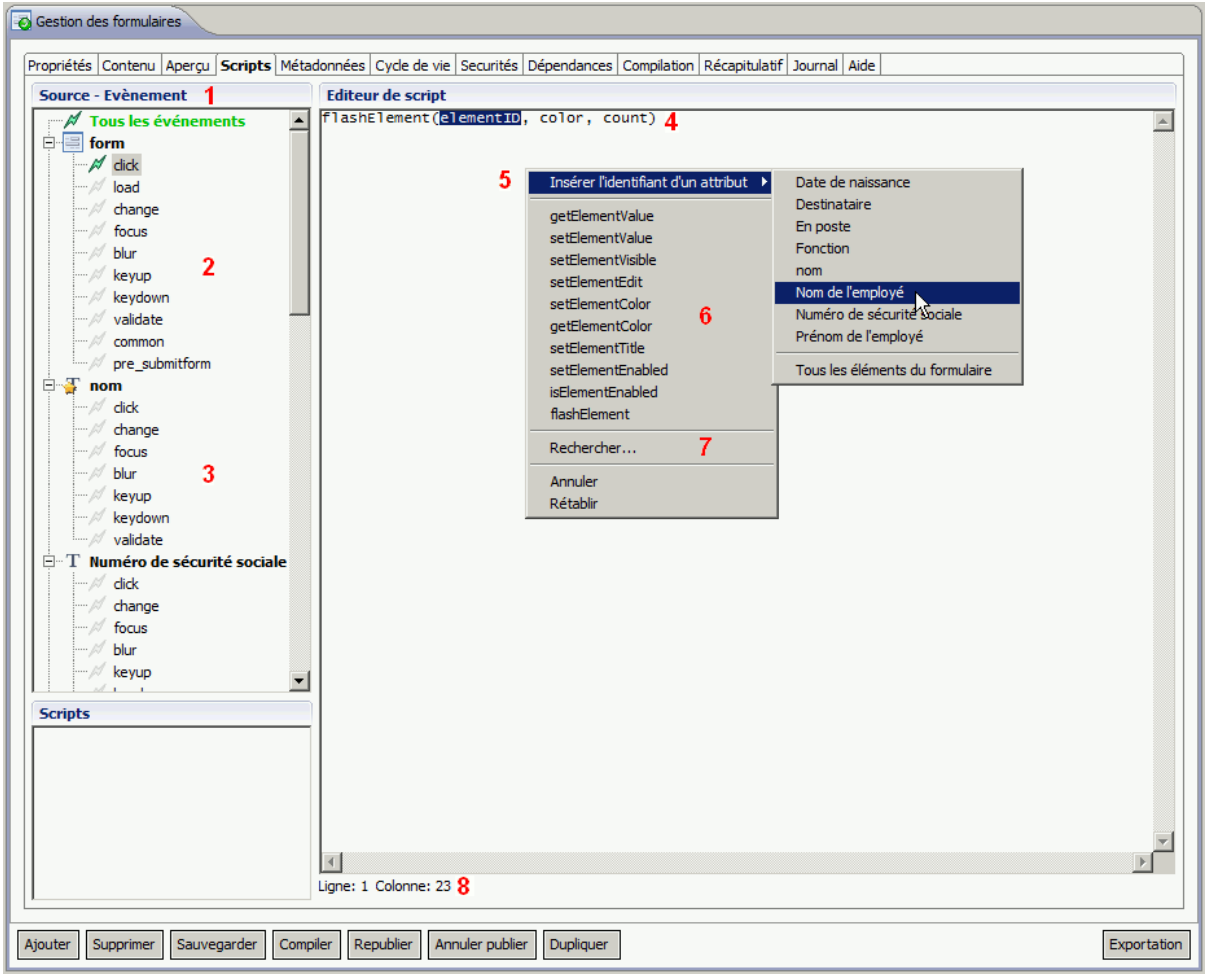
- `BBBB` est l'identificateur de la base de données.
- `NNNNNNNN` est l'identificateur unique de l'attribut dans la base Gargantua.

Les formulaires Gargantua sont affichés dans un navigateur. Ainsi il est possible d'exécuter des scripts écrits en JavaScript pour manipuler les attributs à travers leurs identificateurs.

Éditeur de scripts de formulaire

L'Outil d'administration Gargantua intègre un éditeur de scripts avec un assistant proposant les fonctions JavaScript de l'API Gargantua.

Le clic droit ouvre un menu contextuel qui répertorie les fonctions JavaScript de L'API GARGANTUA.



1. Liste des événements JavaScript regroupés par objet (formulaire + attributs) ;
2. Liste des événements JavaScript du formulaire ;
3. Liste des événements JavaScript d'un attribut du formulaire ;
4. Éditeur de script contenant un appel de fonction de l'API JavaScript GARGANTUA ;
5. Assistant de récupération de l'ID d'un attribut à partir du menu contextuel de l'éditeur ;
6. Assistant de génération de code d'appel à API JavaScript GARGANTUA à partir du menu contextuel de l'éditeur ;
7. Fonction de recherche dans le texte ;
8. Indication du numéro de ligne et de colonne.

API GARGANTUA POUR LES FORMULAIRES

GARGANTUA 7 Serveur fournit un ensemble de fonctions JavaScript qui permettent de modifier le contenu d'un formulaire ou d'effectuer diverses actions.

MACROS SPÉCIALES

SCRIPT.NAME

```
//@script.name " <script_name>"
```

Allows to include a form script within another one, created in the **Scripts** tab.

For instance, we created a script named « library.script.message » in the Scripts tab. To link this script to the form script, we will enter the following code:

```
//@script.name "library.scripts.message".
```

SCRIPT.NO_CHECK

```
//@script.no_check
```

Requests the compiler to ignore compiler errors. The compiler uses a very strict grammar; some code, even if correct, may raise errors due to this.

When using this directive, please make sure you validate the script yourself or be prepared for unexpected errors.

FONCTIONS JAVASCRIPT POUR LES ÉLÉMENTS DU FORMULAIRE

GFXGETATTRIDBYLABEL

```
function gfxGetAttrIdByLabel(label)
```

Retrieves the given attribute's ID based on the label.

gfxGetAttrLabelById

```
function gfxGetAttrLabelById(attributeId)
```

Retrieves the given attribute’s label using it’s ID.

getElementValue

```
function getElementValue(elementID)
```

Returns the value of the specified element.

This API function has been replaced by the gfxVal function; please use the new function whenever possible.

getElementKey

```
function getElementKey(elementID)
```

Returns the key of the specified element if it is a keyed (translated) list or `null` otherwise.

GFXVAL

```
function gfxVal(element [, value])
```

`gfxVal` will either read or write the given attribute's value depeding on the presence of the `value` parameter

gfxEnable

```
function gfxEnable(element)
```

Enables the control for the given attribute.

gfxDisable

```
function gfxDisable(element)
```

Disables the control for the given attribute.

gfxVisible

```
function gfxVisible(element, visible)
```

Changes the visibility of the control for the given attribute.

getElementValueEx

```
function getElementValueEx(elementID)
```

Returns the key of the specified element if it is a keyed (translated) list or the current value otherwise.

setElementValue

```
function setElementValue(elementID, value)
```

Changes the value of the specified element.

This API function has been replaced by the gfxVal function; please use the new function whenever possible.

Parameters

elementID

The ID of the attribute whose value must be changed.

value

New value.

Return value

None.

Example

```
setElementValue("DFATxxxx00000001000000000000000000000000", "test"));
```

will set the label attribute to the value `test`.

setElementKey

```
function setElementKey(elementID, value)
```

Sets the value of an keyed (translated) list attribute to the given key

Parameters

elementID

The ID of the attribute whose value must be changed.

value

New value.

Return value

None.

Example

Let's assume we have the following keyed (translated) list and an attributea of these type.

```
1               one                 premier
   1.1             one.one             premier.premier
   1.2             one.two             premier.deuxième
   1.3             one.three           premier.troisième
2               two                 deuxième
   2.1             two.one             deuxième.premier
   2.2             two.two             deuxième.deuxième
   2.3             two.three           deuxième.troisième
3               three               troisième
4               four                quatrième
5               five                cinquième
```

When the following code is executed

```
setElementKey("DFAT00070000006600000000000000000000000000", "1");
```

The value `premier` will be selected in the form control.

getElementColor

```
function getElementColor(elementID)
```

Returns the color of the specified element.

Parameters

elementID

The ID of the attribute whose value must be returned.

Return value

Color of the specified attribute or `undefined` if there is no custom color set for the control.

Example

```
window.alert(getElementColor("DFAT0007000000660000000000000000000000000"));
```

will display the color of the element or `undefined` if the color is not set.

setElementValueEx

```
function setElementValueEx(elementID, value)
```

Changes the value of the specified element using either the value or the key depending if it is a keyed (translated) list or not.

> *This API function has been replaced by the gfxVal function; please use the new function whenever possible.*

Parameters

elementID

The ID of the attribute whose value must be changed.

value

New key or value.

Return value

None

Example

Let's assume we have the following keyed (translated) list and an attribute of this type.

```
1               one                 premier
   1.1             one.one             premier.premier
   1.2             one.two             premier.deuxième
   1.3             one.three           premier.troisième
2               two                 deuxième
   2.1             two.one             deuxième.premier
   2.2             two.two             deuxième.deuxième
   2.3             two.three           deuxième.troisième
3               three               troisième
4               four                quatrième
5               five                cinquième
```

```
setElementValueEx("DFAT00070000006700000000000000000000000000", "1");
```

will set the attribute to the value `one`.

setElementColor

```
function setElementColor(elementID, color)
```

Changes the color of the specified element.

Parameters

elementID

The ID of the attribute whose color must be changed.

color

New color.

Return value

None.

Example

```
setElementColor("DFAT00070000066000000000000000000000000", "#ff0000");
```

After executing the previous line, the following line

```
window.alert(getElementColor("DFAT00070000066000000000000000000000000"));
```

will display `#ff0000`

setElementColor2

```
function setElementColor2(elementID)
```

This API function has been deprecated in GARGANTUA 8. Please replace it when porting existing forms from older GARGANTUA versions.

Parameters

elementID

The ID of the attribute whose color must be changed.

color

New color.

all

Boolean indicating whether to change the background for all radio inputs in case of a radio element

Return value

None

setElementTitle

```
function setElementTitle(elementID, title)
```

Modifies the tooltip of the specified element.

Parameters

elementID

The ID of the attribute whose tooltip must be modified.

title

New tooltip.

Return value

None

Example

```
setElementTitle("DFAT00070000066000000000000000000000000", "TEST !");
```

setElementVisible

```
function setElementVisible(elementID, value)
```

Changes the visibility of the specified element (visible or invisible).

> *This API function has been replaced by the gfxVisible function; please use the new function whenever possible.*

Parameters

elementID

The ID of the attribute whose visibility needs to be changed.

value

Visibility indicator (»true » or « false »).

Valeur retournée

Aucune

Example

```
setElementVisible("DFAT0007000006600000000000000000000000", false);
```

setElementVisible2

```
function setElementVisible2(elementID)
```

This API function has been deprecated in Gargantua 8. Please replace it when porting existing forms from older Gargantua versions.

Parameters

elementID

The ID of the attribute whose visibility needs to be changed.

Return value

None

setElementEdit

```
function setElementEdit(elementID, value)
```

Modifies the edit state of the specified element (editable or read-only).

> *This API function has been replaced by the gfxEnable and gfxDisable functions; please use the new functions whenever possible.*

Parameters

elementID

The ID of the attribute whose edit state must be changed.

value

Edit state indicator (»true » or « false »).

Valeur retournée

Aucune

Example

```
setElementEdit('DFAT0007000000010000000000000000000000000', false);
```

setElementEdit2

```
function setElementEdit2(elementID)
```

This API function has been deprecated in GARGANTUA 8. Please replace it when porting existing forms from older GARGANTUA versions.

Parameters

elementID

The ID of the attribute whose edit state must be changed.

value

Edit state indicator (»true » or « false »).

Return value

None

Example

```
setElementEdit2('DFAT00070000001000000000000000000000000', false);
```

setElementEnabled

```
function setElementEnabled (elementID, bValue)
```

Modifies the enable state of the specified element (enabled or disabled).

> *This API function has been replaced by the gfxEnable and gfxDisable functions; please use the new functions whenever possible.*

Parameters

elementID

The ID of the attribute whose enable state must be changed.

value

Enable state indicator (»true » or « false »).

Return value

None.

Example

```
setElementEnabled('DFAT0007000000010000000000000000000000000', false);
```

isElementEnabled

```
function isElementEnabled (elementID)
```

Returns the enable state of the specified element (enabled or disabled).

Parameters

elementID

The ID of the attribute whose enable state must be returned.

Return value

The enable state of the attribute.

Example

```
window.alert(isElementEnabled('DFAT00070000066000000000000000000000000'));
```

Will display either `true` or `false`.

disabledEditor

```
function disabledEditor(elementID)
```

This API function has been deprecated in GARGANTUA 8. Please replace it when porting existing forms from older GARGANTUA versions.

Parameters

elementID

The ID of the attribute whose edit state must be changed.

Return value

None.

Example

```
disabledEditor('DFAT0007000000010000000000000000000000000');
```

enabledEditor

```
function enabledEditor(elementID)
```

This API function has been deprecated in Gargantua 8. Please replace it when porting existing forms from older Gargantua versions.

Parameters

elementID

The ID of the attribute whose edit state must be changed.

Return value

None.

Example

```
enabledEditor('DFAT000700000001000000000000000000000000');
```

FORM_GETVALUES

```
function form_getValues()
```

Retrieves all the form values as a JSON object. The returned value also includes hidden attributes.

PARAMETERS

None.

RETURN VALUE

A JSON object with all form values.

EXAMPLE

```
window.alert(JSON.stringify(form_getValues()));
```

will display an output similar to this :

```
{
    "formid":"DFFR000700000007000000000000000000000000",
    "pageid":"Défaut","parentid":"DATA00070000000000000000000000000000000000",
    "objectid":"",
    "command":"view",
    "target":"",
    "multi":"0",
    "search":"0",
    "param1":"",
    "param2":"",
    "param3":"",
    "param4":"",
    "param5":"",
    "DFAT000700000001000000000000000000000000":"",
    "DFAT000700000066000000000000000000000000":"",
    "DFAT000700000067000000000000000000000000":""
}
```

form_getAttributeType

```
function form_getAttributeType(elementID)
```

Retrieves the numeric attribute type for the specified attribute.

Parameters

elementID

The ID of the attribute for which the numeric type ID must be retrieved

Return value

The numeric representation of the type ID for the specified attribute.

Example

```
window.alert(form_getAttributeType('DFAT0007000000660000000000000000000000000'));
```

will display `100`.

form_getAttributeTypeId

```
function form_getAttributeTypeId(elementID)
```

Retrieves the attribute type for the specified attribute.

Parameters

elementID

The ID of the attribute for which the type ID must be retrieved

Return value

The string representation of the type ID for the specified attribute.

Example

```
window.alert(form_getAttributeTypeId('DFAT0007000000660000000000000000000000000'));
```

will display `DFTY0007000000640000000000000000000000000`.

form_getPage

```
form_getPage()
```

Retrieves the name of the current form page.

Parameters

None

Return value

The name of the form page.

Example

```
var page = form_getPage();
```

MISCELLANEOUS JAVASCRIPT FUNCTIONS

FORM_GETVERSION

```
function form_getVersion()
```

Retrieves the version of the GARGANTUA server.

PARAMETERS

None

RETURN VALUE

A string representation of the GARGANTUA 8 server version.

EXAMPLE

```
window.alert(form_getVersion());
```

will display a string like `8.0.7955`.

FORM_LANG

```
function form_lang()
```

Returns the language code of the user interface.

PARAMETERS

None.

RETURN VALUE

Language code. fr: French; ar: Arabic; en: English; es: Spanish; ru: Russian; ro: Romanian; hu: Hungarian.

The GARGANTUA 8 does no longer implement arabic translations, so the « `ar` » return value is no longer used, even for forms that have been migrated from older versions of GARGANTUA.

EXAMPLE

FORM_GETDISPLAYMODE

```
function form_getDisplayMode()
```

Retrieves the current display mode of the form.

PARAMETERS

None.

RETURN VALUE

The current display mode of the form, which can be one of the following values (both symbolic and numeric constants can be used):

```
0 = DISPLAY_NORMAL
1 = DISPLAY_READONLY
2 = DISPLAY_DISABLEDINPUTS
4 = DISPLAY_DISABLEDBUTTONS
```

EXAMPLE

```
window.alert(form_getDisplayMode());
```

form_enableGroup

```
function form_enableGroup(group, enabled);
```

Enables or disables a group of controls at once.

The group property can be set for individual attributes or other form controls in the form designer.

The effect is the same as enabling or disabling individual controls by using gfxEnable and gfxDisable.

Parameters

group

The name of the group.

enabled

Enabled indicator (»true » or « false »).

Return value

None.

Example

```
form_enableGroup("myGroup", false);
```

form_setGroupVisible

```
function form_setGroupVisible(group, visible);
```

Sets the visibility of a group of controls at once.

The group property can be set for individual attributes or other form controls in the form designer.

The effect is the same as showing or hiding individual controls by using gfxVisible.

Parameters

group

The name of the group.

visible

Visibility indicator (»true » or « false »).

Return value

None.

Example

```
form_setGroupVisible("myGroup", false);
```

form_expandSection

```
function form_expandSection(id);
```

Expands a section in grid layout mode.

Parameters

id

The ID of the section.

Return value

None.

Example

```
form_expandSection("theSection");
```

form_collapseSection

```
function form_collapseSection(id);
```

Collapses a section in grid layout mode.

Parameters

id

The ID of the section.

Return value

None.

Example

```
form_collapseSection("theSection");
```

form_toggleSection

```
function form_toggleSection(id);
```

Toggles (expands or collapses) a section in grid layout mode.

Parameters

id

The ID of the section.

Return value

None.

Example

```
form_toggleSection("theSection");
```

FORM_TOGGLEALLSECTIONS

```
form_toggleAllSections(group, showOrHide)
```

Toggles (expands or collapses) all sections (or the sections having the same group) in grid layout mode.

The group property can be set for sections in the form designer.

PARAMETERS

group

(optional) The name of the group

showOrHide

Visibility indicator (»true » or « false »)

RETURN VALUE

None

EXAMPLE

```
form_toggleAllSections("myGroup", false);
form_toggleAllSections(null, true);
```

form_isSectionExpanded

```
function form_isSectionExpanded(id);
```

Retrieves the expanded status of the specified section.

Parameters

id

The ID of the section.

Return value

The expanded status of the specified section.

Example

```
var expanded = form_isSectionExpanded("mySection");
```

form_addSectionIcon

Parameters

id

The ID of the section.

options

A JSON object containing the following key-pairs:

- id - the icon id
- src - the icon source
- position - `start | end`
- className - a class to add to the icon
- title - the icon tooltip
- hidden - boolean indicating if the icon should not be visible
- width - the icon width
- height - the icon height.

Return value

None.

Example

```
form_addSectionIcon('section',{
    id : 'icon',
    src : '/images/plugins/iconpack/svg/bubble.svg#bubble',
    position : 'end',
    hidden : false,
    width : '3rem',
    height : '3rem'
});

form_setSectionIconStyle('icon3', {
    'fill' : 'green'
});
```

form_toggleSectionIcon

```
function form_toggleSectionIcon(id, showOrHide)
```

Function that toggles the visibility of the section icon.

Parameters

id

The ID of the section.

showOrHide

Boolean indicating if the icon should be visible or not.

Return value

None.

Example

```
form_toggleSectionIcon('icon');
```

form_removeSectionIcon

```
function form_removeSectionIcon(id)
```

Function that removes the section icon.

Parameters

id

The ID of the section.

Return value

None.

Example

```
form_removeSectionIcon('icon');
```

form_setSectionIconStyle

```
function form_setSectionIconStyle(id, cssData)
```

Function that sets the section icon's style.

Parameters

id

The ID of the section.

cssData

A JSON object containing the styles to be applied to the icon.

Return value

None.

Example

```
form_addSectionIcon('section',{
    id : 'icon',
    src : '/images/plugins/iconpack/svg/bubble.svg#bubble',
    position : 'end',
    hidden : false,
    width : '3rem',
    height : '3rem'
});

form_setSectionIconStyle('icon3', {
    'fill' : 'green'
});
```

FORM_STICKYSECTION

```
function form_stickySection(id [, options])
```

Positions a section as sticky at the top of the form.

Parameters

id

The section ID.

options

An object containing the following possible options:

- `top` or `stickyTop` parameter indicating the viewport top position when the section should become sticky. By default this is 0.
- `zIndex` or `stickyZIndex` parameter indicating the z-index property for the section. By default this is 10.

Return value

None.

Example

```
form_stickySection('s1');
```

or

```
form_stickySection('s1', { zIndex: 25 });
```

FORM_SECTIONSTOTABS

```
function form_sectionsToTabs(ids, tabId [, options])
```

Transforms the sections identified by the given IDs array into tabs, each section becoming a tab. The section headers will be removed.

Parameters

ids

An array of section IDs.

tabId

The ID of the newly created tabs wrapper.

options

An object containing the following possible options:

- `border` (true/false) parameter indicating weather the tab content should or not have a visible border. By default this is `true`.
- `sticky` (true/false) parameter indicating weather the tab navigation should be rendered as sticky. By default this is `false`.
- `top` or `stickyTop` parameter indicating the viewport top position when the tab navigation should become sticky. Should be specified only in combination with sticky true. By default this is 0.
- `zIndex` or `stickyZIndex` parameter indicating the z-index property for the tab navigation. Should be specified only in combination with sticky true. By default this is 10.
- `height` parameter allowing you to control the height of the tab content. When the content exceeds this height, the content area becomes scrollable.
- `minHeight` parameter allowing you to control the minimum height of the tab content.
- `maxHeight` parameter allowing you to control the maximum height of the tab content. When the content exceeds this height, the content area becomes scrollable.

Return value

None.

Example

```
form_sectionsToTabs(['s1', 's2', 's3', 's4'], 'mytab');
```

or

```
form_sectionsToTabs(['s1', 's2', 's3', 's4'], 'mytab', { border: false });
```

FORM_SECTIONGROUPSTOTABS

```
form_sectionGroupsToTabs(group_names_array, tab_id [, options])
```

Transforms the sections that have the groups specified in the groups array into tabs. This way a tab can contain multiple sections. The group names array should be unique (no duplicates). If any of the sections in a group has the visible mandatory marker then the tab will also have one.

Parameters

group_names_array

An array of group names.

tabId

The ID of the newly created tabs wrapper.

options

An object containing the following possible options:

- `border` (true/false) parameter indicating weather the tab content should or not have a visible border. By default this is `true`.
- `sticky` (true/false) parameter indicating weather the tab navigation should be rendered as sticky. By default this is `false`.
- `top` or `stickyTop` parameter indicating the viewport top position when the tab navigation should become sticky. Should be specified only in combination with sticky true. By default this is 0.
- `zIndex` or `stickyZIndex` parameter indicating the z-index property for the tab navigation. Should be specified only in combination with sticky true. By default this is 10.
- `height` parameter allowing you to control the height of the tab content. When the content exceeds this height, the content area becomes scrollable.
- `minHeight` parameter allowing you to control the minimum height of the tab content.
- `maxHeight` parameter allowing you to control the maximum height of the tab content. When the content exceeds this height, the content area becomes scrollable.
- `tabs` (array of objects) parameter provides custom options for each tab:
  - `id` - the id of the tab (if not provided it will be generated)
  - `name` - the name/label of the tab (if not provided it will be copied from the first section)
  - `onlyContent` (true/false) - indicates weather only the content of the section or the entire section (with header) should be added in the tab content (by default this is `false`).

Return value

None.

Example

```
form_sectionGroupsToTabs(['g1', 'g2', 'g3'], 'mytab', { border: false });
```

or

```
form_sectionGroupsToTabs(['g1', 'g2', 'g3'], 'mytab', {
  border: true,
  sticky: true,
  maxHeight: '200px',
  tabs: [{
    id: 'tab1',
    name: 'First tab'
  }, {
    id: 'tab2',
    name: '<strong>Second tab</strong>',<
    onlyContent: true
  }, {
    id: 'tab3',
    name: '<strong>Third tab</strong>'
  }]
});
```

FORM_REFRESH

```
function form_refresh();
```

Refreshes the entire iframe that contains the form.

PARAMETERS

None.

RETURN VALUE

None.

EXAMPLE

```
form_refresh();
```

form_escapeHtml

```
function form_escapeHtml(text);
```

Escapes a block of text according to HTML escaping rules.

Parameters

text

The text to escape.

Return value

The escaped text.

Example

```
window.alert(form_escapeHtml("\"this is a text");
```

will display

```
&quot;this is a &lt;bracketed&gt; text
```

form_refreshViewer

```
function form_refreshViewer();
```

Refreshes the viewer that may be displayed next to the form. This is useful when form actions trigger document changes and the document must be re-displayed to reflect those changes.

This API function has effect only in the inbox component, where a form may be displayed side by side with a viewer panel.

Parameters

None.

Return value

None.

Example

```
form_refreshViewer();
```

form_cleanupHtml

```
function form_cleanupHtml(text);
```

Removes <p> and
 tags from the input text.

Also replaces any
 tags by a hard line break (\r).

Parameters

text

The text to cleanup.

Return value

The cleaned-up text.

Example

```
window.alert(form_escapeHtml( "<p>this is a paragraph<br>and a line break</p>");
```

will display

```
this is a paragraph
and a line break
```

FORM_LOG

```
function form_log(text);
```

Outputs a log message to the browser console.

PARAMETERS

text

The message to write to the console.

RETURN VALUE

None.

EXAMPLE

```
form_log("message");
```

FORM_INFO

```
function form_info(text);
```

Outputs an info message to the browser console.

Parameters

text

The message to write to the console.

Return value

None.

Example

```
form_info("message");
```

FORM_WARN

```
function form_warn(text);
```

Outputs a warning message to the browser console.

PARAMETERS

text

The message to write to the console.

RETURN VALUE

None.

EXAMPLE

```
form_warn("message");
```

FORM_ERROR

```
function form_error(text);
```

Outputs an error message to the browser console.

Parameters

text

The message to write to the console.

Return value

None.

Example

```
form_error("message");
```

FORM_WAITSTART

```
function form_waitstart(inForm);
```

Affiche un logo rotatif « attendre » pour indiquer que l'application est occupée.

PARAMÈTRES

inForm

(Booléen) Indique si le logo rotatif d'attente doit bloquer le formulaire ou la vue entière. (`true` ou `false`)

VALEUR RETOURNÉE

Aucune

EXAMPLE

```
form_waitstart(true);
```

FORM_WAITSTOP

```
function form_waitstop(inForm, immediately)
```

Ferme le logo rotatif « wait » qui montre si l'application est occupée.

Paramètres

inForm

(Booléen) Indique si le logo rotatif d'attente doit bloquer le formulaire ou la vue entière. (true ou false)

immediately

(Optionnel) Indique si le déblocage doit être effectué immédiatement ou avec le délai d'attente prédéfini par le système. (type int)

Valeur retournée

Aucune

Example

```
form_waitstop(true);
```

form_formatDate

```
function form_formatDate(date);
```

Formats a date with the « dd/MM/yyyy » pattern.

Parameters

date

the date object to format

Return value

The formatted date.

Example

```
var date = new Date();
var formattedDate = form_formatDate(date); // something like "05/05/2024"
```

form_parseDate

```
function form_parseDate(date);
```

Parses a date formatted with the « dd/MM/yyyy » pattern and returns a valid date object.

Parameters

date

A string representation of a date in the « dd/MM/yyyy » pattern

Return value

A date object.

Example

```
var formattedDate = "05/05/2024";
var date = form_parseDate(formattedDate);
```

form_formatUserName

```
function form_formatUserName(userName, realName)
```

Formats the given user name and real name based on the siatel.config.user.name.display.format server option.

Parameters

userName

The user name

realName

The real name of the user

Return value

The formatted user name.

Example

```
var name = form_formatUserName("jeanw", "Jean Winters"); // can return something
like "jeanw (Jean Winters)"
```

Depending on the siatel.config.user.name.display.format server option the result will differ. In the example above the option is set to {{userName}} ({{realName}}).

FORM_HASDOCUMENTS

```
form_hasDocuments(inForm)
```

Indique si le formulaire possède au moins un document

PARAMÈTRES

inForm

Nom du formulaire où l'on veut l'information (string)

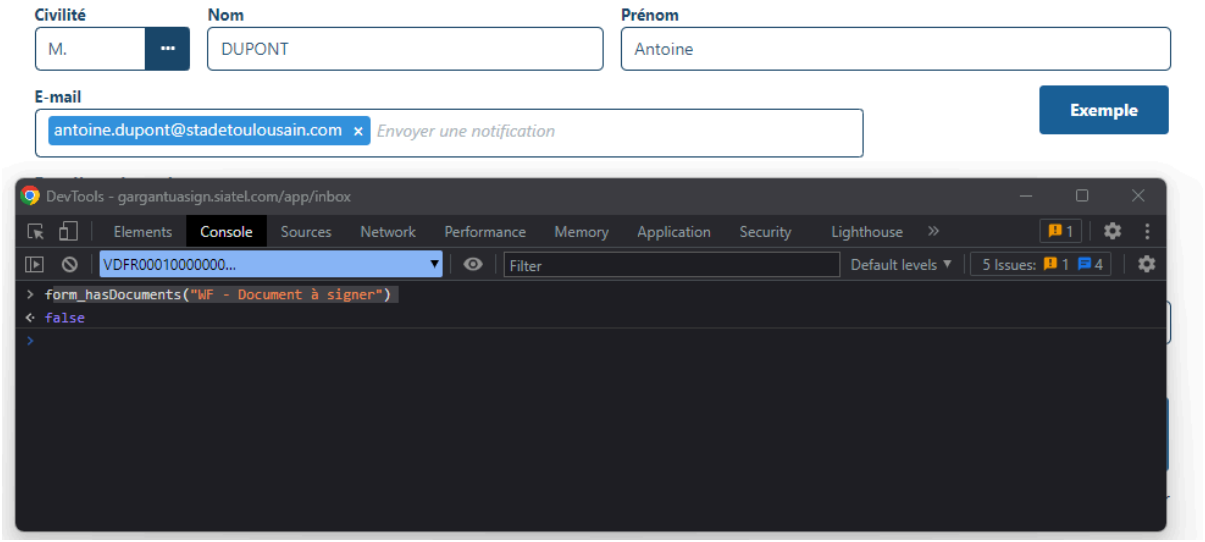
VALEUR RETOURNÉE

(booléen) `true` ou `false`

EXEMPLE

```
form_hasDocuments("WF - Document à signer")
```

renvoie `false` car il n'y a pas de documents indéxés par ce formulaire.



FORM_GETDOCUMENTS

```
form_getDocuments(inForm)
```

renvoie une liste des documents qui sont indexés par le formulaire.

PARAMÈTRES

inForm

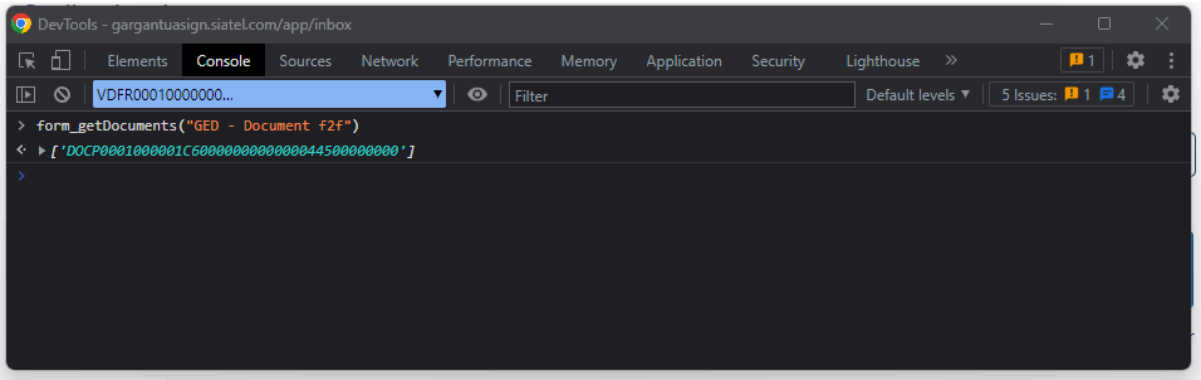
Nom du formulaire (string)

VALEUR RETOURNÉE

Un tableau vide si il n'y a pas de documents indéxés.

Un tableau, avec la liste des id des documents qui sont indexé par le formulaire

EXEMPLE



FORM_FILTERLIST

```
function form_filterList(id, arguments)
```

Filters the options of a list element.

PARAMETERS

id

The id of the list element

arguments

There are 3 possibilities:

- `regexp` - a regular expression either as object or string
- `callback` - a function callback that receives the current option and level as parameters and must return true if the option should be kept or false if it should be removed
- `searchOptions, contains` - an array of options and a boolean indicating whether to keep only the options contained in the searchOptions array (if `true`) or if to remove any of the options that are contained in the array (if `false`)

RETURN VALUE

None

EXAMPLE

```
form_filterList(id, /^test/); // keeps only options who's value starts with test and
removes all others
form_filterList(id, function (option, level) {
    return option.startsWith("test");
});
form_filterList(id, ["test element 1", "test element 2"], true); // keeps only the
options who's values appear in the search array
```

form_setDocTemplateOptions

```
function form_setDocTemplateOptions(options)
```

Specifies the options used by the document template selection dialog.

Parameters

options

an object that can contain the following options:

default

displays default templates; possible values: `0/1, true/false`

custom

displays custom templates(defined in administration console); possible values: `0/1, true/false`

plugin

displays plugin templates; possible values: `0/1, true/false`

tagNames

filter plugin templates by tag name; value: array of tag names

Return value

None.

Example

```
form_setDocTemplateOptions({ default: true, custom: 0, plugin: 1, tagNames: ['tag1',
'tag2'] });
```

flashElement

```
function flashElement(elementID, color, count)
```

Fait clignoter l'élément spécifié. À utiliser pour attirer l'attention d'un utilisateur.

Paramètres

elementID

ID de l'attribut à faire clignoter.

color

La couleur utilisée pour le clignotement.

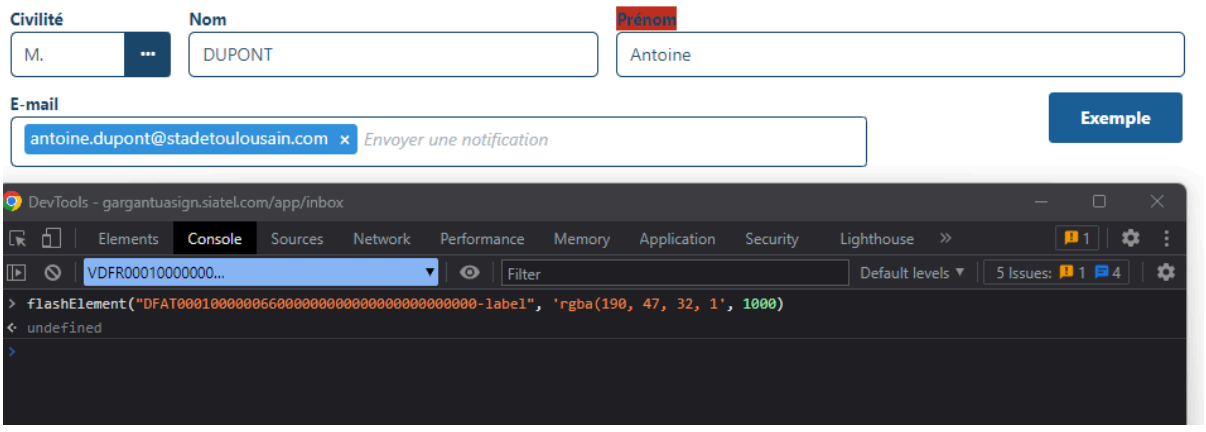
count

Nombre de clignotements.

Valeur retournée

Aucune

Exemple



Le fond de "Prénom" clignote en rouge 1000 fois.

form_displayButtonBar

```
function form_displayButtonBar(bDisplay)
```

Affiche/masque la barre des boutons. Utile si les formulaires contiennent des boutons d'actions personnalisés.

Paramètres

bDisplay

indication de visibilité de la barre des boutons. (booléen) :

`true` pour afficher la barre des boutons, `false` pour la cacher.

Valeur retournée

Aucune.

form_displayDocumentListToggle

```
function form_displayDocumentListToggle()
```

Affiche/masque le conteneur de la liste des documents du processus

Paramètres

Aucun

Valeur retournée

Aucune

form_displayDocumentList

```
function form_displayDocumentList(bDisplay)
```

Affiche/masque la liste des documents du processus.

Cette fonction est obsolète.

Paramètres

bDisplay

(booléen) Indicateur de visibilité de la liste des documents

`true` pour afficher, `false` pout cacher

Valeur retournée

Aucune

form_displaySelectorArea

```
function form_displaySelectorArea(bDisplay)
```

Affiche/masque la barre de sélection de formulaires. Utile dans le cas où on ne doit pas changer de formulaire.

Paramètres

bDisplay

(booléen) indication de visibilité de la barre de sélection de formulaire.

Valeur retournée

Aucune

form_displayDefaultButtonBar

```
function form_displayDefaultButtonBar(bDisplay)
```

Affiche/masque la barre par défaut

Paramètres

bDisplay

(booléen) Indicateur de visibilité de la barre par défaut

`true` pour afficher, `false` pout cacher

Valeur retournée

Aucune

FORM_SETCUSTOMBUTTONENABLED

```
function form_setCustomButtonEnabled(id, enabled)
```

Permet d'activer ou désactiver l'un des boutons personnalisés précédemment créés en appelant `form_displayCustomButtonBar`

PARAMÈTRES

id

L'ID du bouton à montrer ou à cacher (string)

enabled

(booléen) Statut du bouton.

`true` pour activer `false` pour désactiver

VALEUR RETOURNÉE

Aucune

form_displayCustomButtonBar

```
function form_displayCustomButtonBar(data)
```

Affiche une barre de boutons personnalisée sous le formulaire

Paramètres

data

Un objet JSON qui décrit la structure du bouton à créer

Valeur retournée

Aucune

FORM_SETCUSTOMBUTTONVISIBLE

```
function form_setCustomButtonVisible(id, visible)
```

Montre ou cache l'un des boutons personnalisés crées par `form_displayCustomButtonBar`

PARAMÈTRES

id

l'ID du bouton à montrer ou cacher

visible

(booléen) Le nouveau statut du bouton (visible ou non)

form_setCustomButtonHidden

```
function form_setCustomButtonHidden(id, visible)
```

Cette fonction est obsolète, elle existe uniquement pour des raisons de compatibilité et n'a aucun effet.

FORM_REFRESHDOCUMENTLIST

```
function form_refreshDocumentList()
```

Cette fonction est obsolète, elle existe uniquement pour des raisons de compatibilité et n'a aucun effet. L'interface utilisateur de GARGANTUA 8 n'affiche plus la liste des documents en même temps que le formulaire de tâche.

form_displayDocumentListOnReturn

```
function form_displayDocumentListOnReturn()
```

Cette fonction est obsolète, elle existe uniquement pour des raisons de compatibilité et n'a aucun effet. L'interface utilisateur de GARGANTUA 8 n'affiche plus la liste des documents en même temps que le formulaire de tâche.

FORM_PROCESSAREAVERTICALMAXIMIZE

```
function form_processAreaVerticalMaximize(bMaximize)
```

Cette fonction est obsolète, elle existe uniquement pour des raisons de compatibilité et n'a aucun effet. L'interface utilisateur de GARGANTUA 8 n'affiche plus la liste des documents en même temps que le formulaire de tâche.

form_processAreaResizeToggle

```
function form_processAreaResizeToggle()
```

Cette fonction est obsolète, elle existe uniquement pour des raisons de compatibilité et n'a aucun effet. L'interface utilisateur de GARGANTUA 8 n'affiche plus la liste des documents en même temps que le formulaire de tâche.

FORM_PROCESSAREARESIZE

```
function form_processAreaResize(bResize)
```

> ***Cette fonction est obsolète, elle existe uniquement pour des raisons de compatibilité et n'a aucun effet. L'interface utilisateur de GARGANTUA 8 n'affiche plus la liste des documents en même temps que le formulaire de tâche.***

```
form_processAreaResize
```

form_displayAlert

```
function form_displayAlert(data)
```

Affiche une fenêtre d'alerte

Paramètres

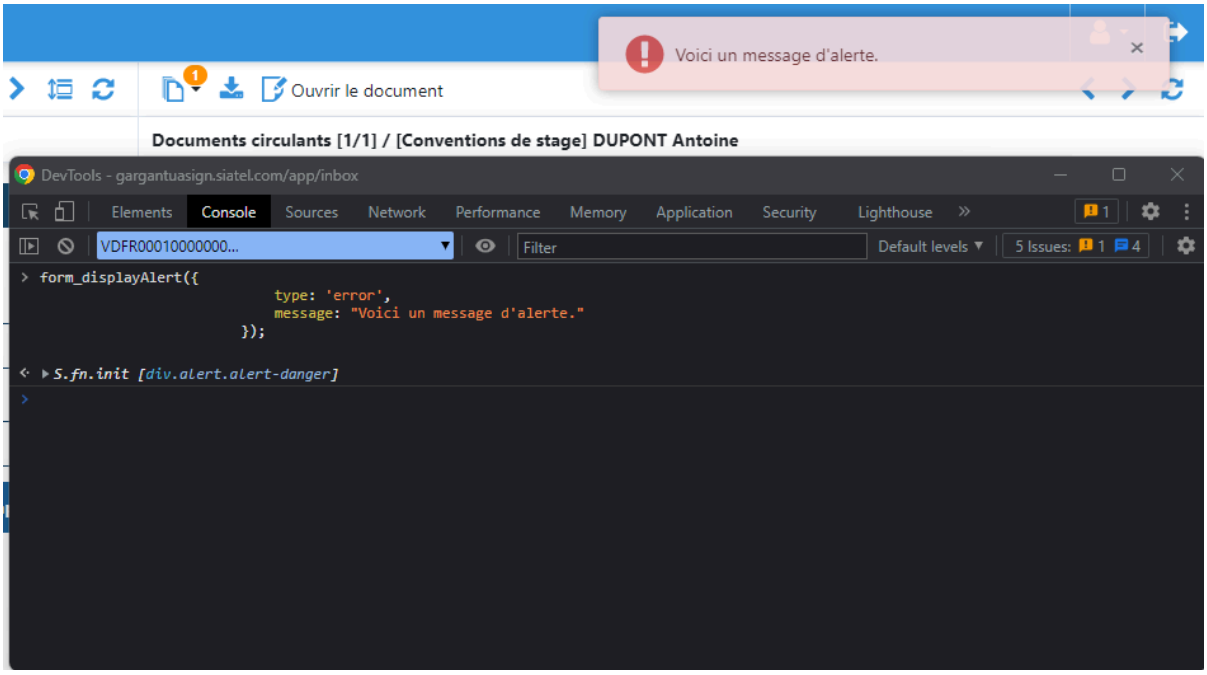
data

Un objet JSON qui décrit l'alerte à afficher.

Valeur retournée

Une alerte jQuery.

Exemple



On peut changer le type de l'alerte : **info** en bleu, **error** en rouge, **warning** en jaune, **success** en vert

form_closeAlert

```
function form_closeAlert($alert)
```

Ferme une alerte précédemment affichée par la fonction `form_displayAlert`

Paramètres

$alert

L'objet jQuery que représente l'alerte à fermer.

form_displayDialog

```
function form_displayDialog(data)
```

Affiche une boite de dialogue.

Paramètres

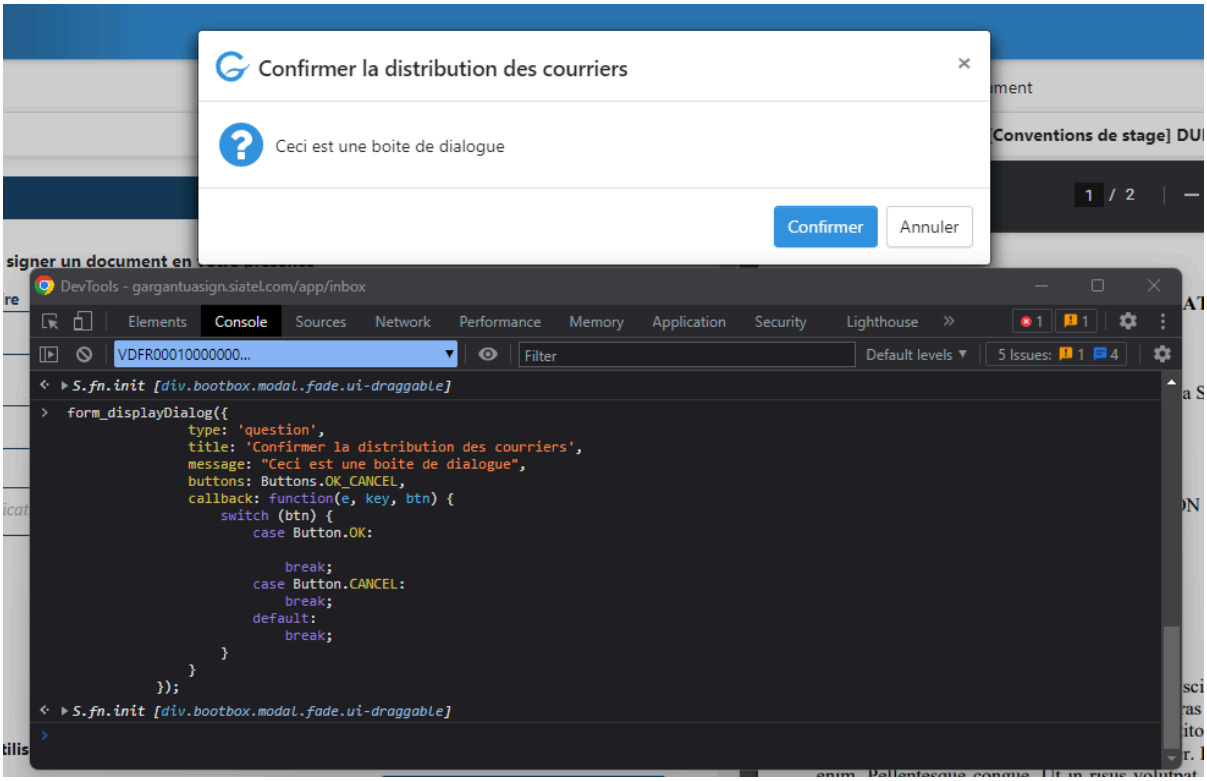
data

Un Object JSON qui décrit la boite de dialogue à afficher

Valeur retournée

Une boite de dialogue jQuery

Exemple



form_closeDialog

```
function form_closeDialog($dialog)
```

Ferme la boite de dialogue créée par la fonction `form_displayDialog`

Paramètres

$dialog

L'objet jQuery que représente la boite de dialogue à fermer.

Fonctions JavaScript pour déclencher des actions

resetElement

```
function resetElement(id)
```

Réinitialise la valeur de l'attribut.

Paramètres

id

L'ID de l'attribut que l'on veut réinitialiser.

runServerScript

```
function runServerScript(scriptName, parameters, callback)
```

Lance l'exécution d'un script au niveau serveur. Le script qui est appelé doit obligatoirement démarrer par la fonction « execute », point d'entrée du script :

```
function execute(params)
{
    // traitement
}
```

Paramètres et valeur retour :

scriptName

nom du script à exécuter.

parameters

paramètres du script. Ils doivent impérativement inclure le paramètre « databaseId », qui permet d'indiquer la base de données dans laquelle se trouve le script à exécuter.

Par exemple : « databaseId=DATA000800000000000000000000000000000000 ».

callback

Fonction qui doit être exécutée quand la réponse du serveur est reçue.

bAsync

Un booléen qui indique si la requête doit être synchrone ou asynchrone. Le mode asynchrone est préférable, cela permet une interface utilisateur plus responsive.

Valeur retournée

Réponse du serveur au format indiqué de la fonction invoquée. En cas d'appel asynchrone, la réponse sera reçue et traitée par la fonction de rappel.

Cette fonction API a été remplacée par la fonction form_runScript. Veuillez utiliser cette dernière autant que possible.

runServerScript2

```
function runServerScript2(scriptName, functionName, parameters, callback, bAsync)
```

Démarre l'exécution du script au niveau du serveur.

Paramètres et valeur retour :

scriptName

nom du script à exécuter.

functionName

nom du fonction à exécuter.

parameters

paramètres du script. Ils doivent impérativement inclure le paramètre « databaseId », qui permet d'indiquer la base de données dans laquelle se trouve le script à exécuter.

Par exemple : « databaseId=DATA0008000000000000000000000000000000000 ».

callback

Fonction qui doit être exécutée quand la réponse du serveur est reçue.

bAsync

Un booléen qui indique si la requête doit être synchrone ou asynchrone. Le mode asynchrone est préférable, cela permet une interface utilisateur plus responsive.

Valeur retournée

Réponse du serveur au format indiqué de la fonction invoquée. En cas d'appel asynchrone, la réponse sera reçue et traitée par la fonction de rappel.

Cette fonction API a été remplacée par la fonction form_runScript. Veuillez utiliser cette dernière autant que possible.

FORM_RUNSCRIPT

```
function form_runScript(options)
```

Démarre l'exécution du script au niveau du serveur.

Paramètres et valeur retour :

options

Objet clé-valeur spécifiant les options suivantes :

- `script` ou `scriptName` - nom du script à exécuter
- `function` ou `functionName` - nom de la fonction à exécuter
- `acync` - booléen indiquant si la requête doit être synchrone (false) ou asynchrone (true)
- `data` - objet contenant des données supplémentaires à envoyer
- `callback` - fonction appelée lorsque le résultat est disponible.

Valeur retournée

Aucune.

Example :

```
form_runScript({
    script: 'server.queries',
    'function': 'getValues',
    data: {
        docId: ids[0]
    },
    callback: function (result) {
        try {
           var json = JSON.parse(result);
           form_info(json);
           ....
        }
        catch (e) {
            form_error(e);
        }
    }
});
```

form_apiRequest

```
function form_apiRequest(options)
```

Utilisé pour effectuer des appels à des sources externes (et éviter les problèmes de requêtes cross-origin). L'appel transite par le serveur GARGANTUA, qui effectue la requête en votre nom.

Paramètres et valeur retour :

options

Objet clé-valeur contenant les options suivantes :

- `url` - l'URL à appeler
- `method` - a méthode HTTP (GET par défaut)
- `headers` - les en-têtes HTTP (sous forme d'objet clé-valeur)
- `params` - les paramètres HTTP (sous forme d'objet clé-valeur)
- `body` - le corps de la requête HTTP (chaîne de caractères ou objet)
- `contentType` - le type de contenu
- `async` - booléen indiquant si la requête doit être synchrone (false) ou asynchrone (true)
- `callback` - fonction à appeler lorsque le résultat est disponible

Valeur retournée

Aucune.

Example :

```
form_apiRequest({
    url :  'https://thequoteshub.com/api/random-quote',
    method : 'GET',
    callback : function(result) {
        form_info(JSON.parse(result));
    }
});
```

FORM_SCANDOCUMENTS

```
function form_scanDocuments(direct)
```

Lance l'opération de numérisation si le contexte d'exécution est correctement initialisé.

Paramètres et valeur retour :

direct

[deprecated] indicateur de numérisation directe.

Valeur retournée

Aucune.

form_scanDocumentsExt

> ***This function has been deprecated; it exists only for compatibility reasons. Please replace it with « `form_addDocuments` ». Currently, this function call defaults to « `form_addDocuments` »***

```
function form_scanDocumentsExt(direct, label)
```

Launches the scanning operation if the execution context is properly initialized.

Parameters

direct

Indicator for direct scanning. See `form_addDocuments` for more information.

label

The label to use for the newly scanned documents.

Return value

None.

Example

form_scanDocumentsExt2

This function has been deprecated; it exists only for compatibility reasons. Please replace it with « form_addDocuments ». Currently, this function call defaults to « form_addDocuments »

```
function form_scanDocumentsExt2(direct, form)
```

Launches the scanning operation if the execution context is properly initialized.

Parameters

direct

Indicator for direct scanning. See `form_addDocuments` for more information.

form

The Form to use for the newly scanned documents.

Return value

None.

Example

form_scanDocumentsExt3

> ***This function has been deprecated; it exists only for compatibility reasons. Please replace it with « form_addDocuments ». Currently, this function call defaults to « form_addDocuments »***

```
function form_scanDocumentsExt3(direct, form, label)
```

Launches the scanning operation if the execution context is properly initialized.

Parameters

direct

Indicator for direct scanning. See `form_addDocuments` for more information.

form

The Form to use for the newly scanned documents.

label

The label to use for the newly scanned documents.

Return value

None.

Example

form_createDocument

```
function form_createDocument(template, label, form, fulltext, markers)
```

Creates a new document based on a template and text replacements,

Parameters

template

The template ID

label

The label to use for the new document.

form

The form to use for the new document.

fulltext

Boolean to indicate whether the document has to be indexed for fulltext search

markers

A map of (<placeholder>, <value>) elements to replace in the new document

Return value

The new document ID.

Example

FORM_ADDDOCUMENTS

```
function form_addDocuments()
```

Lance l'importation des documents si le contexte d'exécution est correctement initialisé.

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

form_addDocumentsDirect

This function has been deprecated; it exists only for compatibility reasons. Please replace it with « `form_addDocuments` ». Currently, this function call defaults to « `form_addDocuments` »

```
function form_addDocumentsDirect(direct)
```

Launches the add document operation if the execution context is properly initialized.

Parameters

direct

Indicator for direct add (no form selector). See `form_addDocuments` for more information.

Return value

None.

Example

form_certifyDocument

```
function form_certifyDocument(params)
```

Shows the document certification user interface elements for the given document.

Parameters

params

An object describing the operation options.

id

The document id to display the certification UI for.

Return value

None.

Example

form_editDocument

```
function form_editDocument(id, create, callback)
```

Triggers the 'edit native' operation for a given document

Parameters

id

The document ID to edit

create

Indicates whether this is a new document or an existing document; for new documents, when the editing operation ends, there will be no dialog box asking for versioning information

callback

The callback used when the operation opens the native application.

Return value

None.

Example

FORM_VIEWDOCUMENT

```
function form_viewDocument(id, options)
```

Displays the view document user interface elements according to the global layout and settings; has the same effect as the user clicking on the document label or double-clicking on the document line or thumbnail.

PARAMETERS

id

The document ID to display

options

A JSON object specifying the operation options.

The options parameter is a JSON object that needs to follow the following structure; one or more members may be present.

```
{
    viewer : {
        toolbar : {
            close : false,
            zapper : false,
            pinZapper : false,
            editNative : false,
            closeOnEditNative : false,
            switchView : false
        },
        label : true,
        panel : {
            display : 'none',
            fragment : true,
            notes : true,
            ocr : true,
            links : false,
            actions : false,
            sysinfo : true
```

```
        }
    },
    getDisplayData : $.noop,
    onZap : $.noop,
    onClose : $.noop,
    onActionFinished : $.noop
}
```

viewer/toolbar

individual commands for the viewer panel : close button, zapper, zapper for pins, edit native, close viewer on editnative and switch view.

label

boolean to indicate if label should be displayed or not

panel

what the footer panel shoulr display : default display, fragment, notes, ocr, links, actions, system information

getDisplayData, onZap, onClose, onActionFinished

various callbacks for the viewer panel

Return value

None.

Example

form_signDocument

```
function form_signDocument(params)
```

Shows the document signing user interface elements for the given document.

Parameters

params

An object describing the operation options.

id

The document id to display the signing UI for.

Return value

None.

Example

form_viewDocuments

```
function form_viewDocuments()
```

Displays the view documents user interface elements according to the global layout and settings.

This API function call is used exclusively in inbox context.

Parameters

None.

Return value

None.

Example

...

form_downloadDocuments

```
function form_downloadDocuments(ids)
```

Function that downloads the documents identified by the ids array.

Parameters

ids

Array of document ids.

Return value

None

form_editObject

```
function form_editObject(id, options)
```

Function that opens the view/edit attributes dialog for the specified object.

Parameters

id

The ID of the object to edit.

options

Edit options.

Return value

None.

Example

...

form_save

```
function form_save()
```

Sauvegarde le formulaire du processus ou de la tâche affichée.

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

FORM_SAVEIMMEDIATE

```
function form_saveImmediate()
```

Sauvegarde immédiatement et silencieusement le formulaire du processus ou de la tâche affichée (aucun message de confirmation ne sera affiché).

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

FORM_SAVEDRAFT

```
function form_saveDraft()
```

Sauvegarde le formulaire de la tâche affichée.

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

This API function has been replaced by the form_save function; please use the new function whenever possible.

form_saveDraftImmediate

```
function form_saveDraftImmediate()
```

Saves all the form content (all attribute values) immediately and silently.

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

This API function has been replaced by the form_saveImmediate function; please use the new function whenever possible.

form_saveTaskImmediate

```
function form_saveTaskImmediate()
```

Saves all the form content (all attribute values) immediately and silently.

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

This API function has been replaced by the form_saveImmediate function; please use the new function whenever possible.

form_saveTask

```
function form_saveTask()
```

Sauvegarde le formulaire de la tâche affichée.

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

This API function has been replaced by the form_save function; please use the new function whenever possible.

FORM_CANCELDRAFT

```
function form_cancelDraft()
```

Annule le brouillon et par conséquent le démarrage du processus.

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

> ***This API function has been replaced by the form_cancel function; please use the new function whenever possible.***

FORM_CANCEL

```
function form_cancel()
```

Annule le brouillon du processus (et par conséquent le démarrage du processus) ou la tâche en cours (si le contexte d'exécution est correctement initialisé), selon le cas.

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

FORM_FINISH

```
function form_finish()
```

Démarre un nouveau processus si le processus en cours est un brouillon ou envoie la tâche en cours (valider la tâche - tâche effectuée) si le contexte d'exécution est correctement initialisé, selon le cas.

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

FORM_STARTPROCESS

```
function form_startProcess()
```

Démarre un nouveau processus si le processus est à l'état de brouillon et que le contexte d'exécution est correctement initialisé.

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

This API function has been replaced by the form_finish function; please use the new function whenever possible.

FORM_CANCELTASK

```
function form_cancelTask()
```

Envoie la tâche (tâche annulée) si le contexte d'exécution est correctement initialisé.

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

> ***This API function has been replaced by the form_cancel function; please use the new function whenever possible.***

FORM_ESCALATE

```
function form_escalate()
```

Escalates the task according to the rules set by the process definition.

Paramètres et valeur retour :

Aucun paramètre.

Valeur retournée

Aucune.

form_finishTask

```
function form_finishTask()
```

Envoie la tâche (valider la tâche - tâche effectuée) si le contexte d'exécution est correctement initialisé.

Paramètres et valeur retour :

Aucun paramètre.

Valeur retournée

Aucune.

This API function has been replaced by the form_finish function; please use the new function whenever possible.

FORM_EXECUTESEARCH

```
function form_executeSearch()
```

Lance l'exécution d'une recherche si le contexte d'exécution est correctement initialisé.

Paramètres et valeur retour :

Aucun paramètre

Valeur retournée

Aucune.

FORM_ESCALATETASK

```
function form_escalateTask()
```

Escalates the task according to the rules set by the process definition.

Paramètres et valeur retour :

Aucun paramètre.

Valeur retournée

Aucune.

This API function has been replaced by the form_escalate function; please use the new function whenever possible.

doValidateForm

```
function doValidateForm()
```

Effectue le test de validation des champs. Désactive le bouton envoyé et met en rouge le champ qui contient l'erreur.

Paramètres et valeur retour :

Aucune

Valeur retournée

Aucune.

form_switchPage

```
function form_switchPage(strPageName)
```

Afficher la page du formulaire dont le nom est indiqué en tant que paramètre.

Paramètres et valeur retour :

strPageName

nom de la page à afficher.

Valeur retournée

Aucune.

form_switchForm

```
function form_switchForm(form, page)
```

Switch to the given form and form page.

Parameters

form

the name of the form that contains the page to display.

page

the name of the page to display.

Return value

None.

Example

FORM_GOTOTASKS

```
function form_gotoTasks()
```

Returns to the task list from the form/document display.

Parameters

None

Return value

None

Exceptions

n/a

Example

```
form_gotoTasks();
```

form_invokePlugin

```
function form_invokePlugin(pluginId, command, params, format)
```

Executes the given command from a server plugin, using the provided parameters and output format.

Parameters

pluginId

the ID of the plugin to invoke.

command

the name of the plugin command to execute.

params

A JSON objects containing command-specific parameters.

format

A string specifying the expected return format; can be either xml or text.

Return value

The server response in the requested format, if the plugin can handle it; otherwise, it is a text response.

Example

ADVANCED JAVASCRIPT FUNCTIONS

form_newControl

```
function form_newControl(id, type, options)
```

Creates a new input control based on the given parameters.

Parameters

Return value

Example

form_destroyControl

```
function form_destroyControl(id)
```

Destroys a previously manually created control

Parameters

id

The form element control ID

Return value

None

Example

form_getControl

```
function form_getControl(id)
```

Retrieves the control for the given input ID

Parameters

id

The form element control ID

Return value

The new control object.

Example

form_getElement

```
function form_getElement()
```

Retrieves the jQuery object associated with the given element ID

Parameters

id

The page element ID

Return value

The jQuery object associated with the given element ID

Example

getElementKeyAndValue

```
function getElementKeyAndValue()
```

Retrieves the selected (key, value) pair from the autocomplete control as a JSON object.

Parameters

id

The form element control ID

Return value

A JSON object that represents the (key, value) pair for the given selector.

Example

setElementKeyAndValue

```
function setElementKeyAndValue(id, key, value)
```

Sets the key and value for the given autocomplete control (a list)

Parameters

id

The form element control ID

key

The key to be set

value

The value to be set

Return value

None

Example

form_saveGlobalGrids

```
function form_saveGlobalGrids()
```

Saves the values of all grids in the form.

This API function has been deprecated in Gargantua 8. You may safely remove it when porting existing forms from older Gargantua versions.

Parameters

None

Return value

None

Example

form_validateLatency

```
function form_validateLatency(timeout)
```

Modifies the form validation latency; this affects how the validation is performed on the form values.

Parameters

timeout

The validation timeout to be used; effective from the function call

Return value

None

Example

form_getValidateLatency

```
function form_getValidateLatency()
```

Retrieves the current form validation latency value.

Parameters

Return value

newMode

Example

form_validateLatencyOffset

```
function form_validateLatencyOffset(offset)
```

Modifies the form validation latency offset; this affects how the validation is performed on the form values.

Parameters

offset

The validation timeout offset to be used; effective from the function call

Return value

None

Example

form_validateMode

```
function form_validateMode(newMode)
```

Changes the validation mode for the form.

Parameters

newMode

The new validation mode for the form; it can be either `VALIDATE_ONSUBMIT` or `VALIDATE_CONTINUOUS`

Return value

None

Example

form_getValidateMode

```
function form_getValidateMode()
```

Retrieves the current validation mode for the form.

Parameters

None

Return value

The current validation mode. It can be either `VALIDATE_ONSUBMIT` or `VALIDATE_CONTINUOUS`

Example

FORM_GETVALIDATIONMESSAGE

```
function form_getValidationMessage(failedIds, failedReasons, failedCodes [, options])
```

Returns a standard validation message with the form errors.

Parameters

failedIds

An array with the failed attribute IDs.

failedReasons

An array with the failure reasons.

failedCodes

An array with the failure codes.

options

An object containing the following possible options:

- `multiline` parameter indicating if the different errors should be rendered on separate lines or not. By default this is `false`.

Return value

None.

Example

```
function fun_form_post_validatereponse(failedIds, failedReasons, failedCodes)
{
    var message = form_getValidationMessage(failedIds, failedReasons, failedCodes, {
multiline: true });
    if (message)
        form_displayAlert({ type : 'error', message: message });
}
```

FORM_SETSUBMITONENTER

```
function form_setSubmitOnEnter(bEnable)
```

Enables or disables the use of the **Enter** key for submitting the form.

By default, the « Enter » key is enabled for submitting the form.

Parameters

bEnable

Boolean to enable or disable the use of the **Enter** key for submitting the form..

Return value

None.

Example

```
form_setSubmitOnEnter(false);
```

Controls

General behavior

getValue

```
getValue()
```

Gets the value of the control as a string.

Parameters

None.

Return value

The current value of the control.

Example

```
var control = form_getControl(attrId);
var value = control.getValue();
```

Equivalent to:

```
var value = getElementValue(attrId);
```

setValue

```
setValue(value)
```

Sets the value of the control as a string.

Parameters

value

The value to set.

Return value

None

Example

```
var value = "...";
var control = form_getControl(attrId);
control.setValue(value);
```

Equivalent to:

```
setElementValue(attrId, value);
```

enable

```
enable()
```

Enables the control.

Parameters

None

Return value

None

Example

```
var control = form_getControl(attrId);
control.enable();
```

Equivalent to:

```
setElementEnabled(attrId, true);
```

disable

```
disable()
```

Disables the control.

Parameters

None

Return value

None

Example

```
var control = form_getControl(attrId);
control.disable();
```

Equivalent to:

```
setElementEnabled(attrId, false);
```

isEmpty

```
isEmpty()
```

Checks if the control value is empty.

Parameters

None

Return value

`true` if the value is empty, `false` otherwise

Example

```
var control = form_getControl(attrId);
if (control.isEmpty())
{ ... }
```

reset

```
reset()
```

Resets the value of the control to the initial value.

Parameters

None

Return value

None

Example

```
var control = form_getControl();
control.reset();
```

isEnabled

```
isEnabled()
```

Checks if the control is enabled.

Parameters

None

Return value

`true` if the control is enabled, `false` otherwise

Example

```
var control = form_getControl(attrId);
if (control.isEnabled())
{ ... }
```

Equivalent to:

```
isElementEnabled(attrId)
```

hasError

```
hasError()
```

Checks if the control value is correct or not (if it respects the control's pattern).

Parameters

None

Return value

`true` if the value is correct, `false` otherwise

Example

```
var control = form_getControl(attrId);
if (control.hasError())
{ ... }
```

SIMPLE INPUT FIELDS

Simple string input

To create a string control:

```
var control = form_newControl(attrId, 'string');
```

Numeric input

To create a numeric control:

```
var control = form_newControl(attrId, 'number', options);
```

The options include:

type

The type of number `[integer|float]`

negative

Boolean indicating if the number can be negative or not

min

The minimum value (any value below it will be considered invalid)

max

The maximum value (any value above it will be considered invalid)

maxUnits

The maximum number of units

maxSubunits

The maximum number of subunits

getNumber

```
getNumber()
```

Gets the value of the control as a number.

Parameters

None

Return value

The value as a number.

Example

```
var control = form_getControl(attrId);
var number = control.getNumber();
```

setNumber

```
setNumber(number)
```

Sets the value of the control as a number.

Parameters

number

The value as a number.

Return value

None

Example

```
var control = form_getControl(attrId);
control.setNumber(250);
```

Date input

To create a date control:

```
var control = form_newControl(attrId, 'date', options);
```

The options include:

mouseWheel

Boolean indicating if the mouse wheel should or not be used to alter the input's value

getDate

```
getDate()
```

Gets the value of the control as a date (without the hour, minutes, seconds and milliseconds).

Parameters

None

Return value

The value as a date.

Example

```
var control = form_getControl(attrId);
var date = control.getDate();
```

setDate

```
setDate(date)
```

Sets the value of the control as a date (without the hour, minutes, seconds and milliseconds).

Parameters

date

The value as a date.

Return value

None

Example

```
var control = form_getControl(attrId);
control.setDate(new Date());
```

Time input

To create a time control:

```
var control = form_newControl(attrId, 'time', options);
```

The options include:

mouseWheel

Boolean indicating if the mouse wheel should or not be used to alter the input's value

getTime

```
getTime()
```

Gets the value of the control as an array with 2 elements (hours and minutes).

Parameters

None

Return value

The value as an array with 2 elements (hours and minutes).

Example

```
var control = form_getControl(attrId);
var timeArray = control.getTime();
```

setTime

```
setTime(time)
```

Sets the value of the control as an array with 2 elements (hours and minutes).

Parameters

time

The value as an array with 2 elements (hours and minutes).

Return value

None

Example

```
var control = form_getControl(attrId);
control.setTime([10, 25]);
```

Date & time input

To create a date-time control:

```
var control = form_newControl(attrId, 'datetime', options);
```

The options include:

mouseWheel

Boolean indicating if the mouse wheel should or not be used to alter the input's value

getTimestamp

```
getTimestamp()
```

Gets the value of the control as a date.

Parameters

None

Return value

The value as a date.

Example

```
var control = form_getControl(attrId);
var date = control.getTimestamp();
```

setTimestamp

```
setTimestamp(date)
```

Sets the value of the control as a date.

Parameters

date

The value as a date.

Return value

None

Example

```
var control = form_getControl(attrId);
control.setTimestamp(new Date());
```

Time delay input

To create a time delay control:

```
var control = form_newControl(attrId, 'timedelay', options);
```

The options include:

mouseWheel

Boolean indicating if the mouse wheel should or not be used to alter the input's value

getDelay

```
getDelay()
```

Gets the value of the control as a number (in seconds).

Parameters

None

Return value

The value as a number.

Example

```
var control = form_getControl(attrId);
var delayInSeconds = control.getDelay();
```

setDelay

```
setDelay(delay)
```

Sets the value of the control as a number (in seconds).

Parameters

delay

The value as a date.

Return value

None

Example

```
var control = form_getControl(attrId);
control.setDelay(3600); // 60 minutes
```

Email input

To create an email control:

```
var control = form_newControl(attrId, 'email', options);
```

The options include:

multiple

Boolean indicating if the value can contain multiple emails

source

The source URL

Lists

Autocomplete

To create an autocomplete control:

```
var control = form_newControl(attrId, 'autocomplete', options);
```

The source of the autocomplete is an array of objects. Each object should have at least two properties: a unique value that identifies the object and a display value (for example an id and a name).

The options include:

multiple

Boolean indicating if the control allows multiple values

displayKey

The key identifying the value to display in the control element (for instance the name in the example above)

valueKey

The key that uniquely identifies the selected value (for instance the id in the example above)

freeInput

Boolean indicating if the autocomplete should allow the user to write values that do not exist in the source

source

The source of the autocomplete (can be a URL, an object or a function)

minLength

The minimum number of characters the user has to type before the autocomplete is triggered

prepareSuggestion

A callback function allowing you to customize the display of the autocomplete suggestions

Example

```
form_newControl(attrId, 'autocomplete', {
    displayKey: 'value',
    valueKey: 'key',
    source: {
        values : [
            { key : 'k1', value : 'New York' },
            { key : 'k2', value : 'Chicago' },
            { key : 'k3', value : 'Geneva' },
            { key : 'k4', value : 'London' },
            { key : 'k5', value : 'Helsinki' },
            { key : 'k6', value : 'Boston' },
            { key : 'k7', value : 'Paris (France)' },
```

```
            { key : 'k8', value : 'Paris (Texas)' },
            { key : 'k9', value : 'Memphis (Egypt)' },
            { key : 'k10', value : 'Memphis (Tennessee)' },
            { key : 'k11', value : 'Madrid' },
             { key : 'k12', value : 'Vienna' },
             { key : 'k13', value : 'Sydney (Australia)' },
             { key : 'k14', value : 'Sydney (Canada)' }
        ]
    }
});
```

The source should always look like the example above even if it's a function or an URL, the returned value must follow the same pattern.

getKey

```
getKey()
```

Gets the key of the control.

Parameters

None

Return value

The key of the control.

Example

```
var control = form_getControl(attrId);
var key = control.getKey();
var value = control.getValue();
```

For the example of cities above, if the user selected « Paris (France) », the key will be `k7` and the value will be `Paris (France)`.

setKey

```
setKey(key)
```

Sets the key of the control.

Parameters

key

The key to set.

Return value

None

Example

```
var control = form_getControl(attrId);
control.setKey('k7');
```

getKeyAndValue

```
getKeyAndValue()
```

Gets an object containing the key and value of the control (the property names will be those configured as displayKey and valueKey).

Parameters

None

Return value

An object containing the key and value of the control.

Example

```
var control = form_getControl(attrId);
var obj = control.getKeyAndValue();
```

For the example of cities above, if the user selected "Paris (France)", the object will be `{ 'key' : 'k7', 'value' : 'Paris (France)' }`.

Equivalent to:

```
getElementKeyAndValue(attrId);
```

setKeyAndValue

```
setKeyAndValue(object)
```

Sets the key and value of the control.

Parameters

object

The key and value to set.

Return value

None

Example

```
var control = form_getControl(attrId);
control.setKeyAndValue({ 'key' : 'k7', 'value' : 'Paris (France)' });
```

Equivalent to:

```
setElementKeyAndValue(attrId, { 'key' : 'k7', 'value' : 'Paris (France)' });
```

Simple list

To create a simple list control:

```
var control = form_newControl(attrId, 'list', options);
```

The options include:

flags

`[m|s|p|c]` The flags indicate the type of the list. `m` stands for multiple selection list, `s` stands for simple list, `p` stands for full path list, and `c` stands for category list

source

The source of the list (can be a URL, an object or a function)

closed

Boolean indicating if the list should not allow the user to write values that do not exist in the source

addEmptyOption

Boolean indicating if an empty option should be added to the list

prepareOption

A callback function allowing you to customize the display of the list options

linkId

The id of the master list (if you need to create linked lists; for multi level lists - each list will display it's own level)

Example

```
form_newControl(attrId, 'list', {
    source: {
        values : [
            { key : 'c1', value : 'America', values : [
                { key : 'k1', value : 'New York' },
                { key : 'k2', value : 'Chicago' },
                { key : 'k3', value : 'Boston' }
             ]},
             { key : 'c2', value : 'Europe', values : [
                { key : 'k4', value : 'London' },
                { key : 'k5', value : 'Paris' },
                { key : 'k6', value : 'Vienna' }
             ]}
        ]
    }
});
```

The example above will build a list with 2 levels.

If you add the `flags: 'c'` option the list will transform into a category list in which only the leaf values will be selectable.
By default the flags option is considered `'s'`, in which case all values are selectable. In the case of `'m'` flag you can select multiple values, and in the case of `'p'` flag the selected value will contain the full path (for instance 'Europe/Paris').

getKey

```
getKey()
```

Gets the key of the control.

Parameters

None

Return value

The key of the control.

Example

```
var control = form_getControl(attrId);
var key = control.getKey();
var value = control.getValue();
```

For the example of cities above, if the user selected « Paris (France) », the key will be `k7` and the value will be `Paris (France)`.

setKey

```
setKey(key)
```

Sets the key of the control.

Parameters

key

The key to set.

Return value

None

Example

```
var control = form_getControl(attrId);
control.setKey('k7');
```

getKeyAndValue

```
getKeyAndValue()
```

Gets an object containing the key and value of the control (the property names will be those configured as displayKey and valueKey).

Parameters

None

Return value

An object containing the key and value of the control.

Example

```
var control = form_getControl(attrId);
var obj = control.getKeyAndValue();
```

For the example of cities above, if the user selected "Paris (France)", the object will be `{ 'key' : 'k7', 'value' : 'Paris (France)' }`.

Equivalent to:

```
getElementKeyAndValue(attrId);
```

setKeyAndValue

```
setKeyAndValue(object)
```

Sets the key and value of the control.

Parameters

object

The key and value to set.

Return value

None

Example

```
var control = form_getControl(attrId);
control.setKeyAndValue({ 'key' : 'k7', 'value' : 'Paris (France)' });
```

Equivalent to:

```
setElementKeyAndValue(attrId, { 'key' : 'k7', 'value' : 'Paris (France)' });
```

Multi-line list

To create a multiline list control:

```
var control = form_newControl(attrId, 'multilinelist', options);
```

This will create a multiline select element.

The options include:

source

The source of the list (an object; see simple list example)

addEmptyOption

Boolean indicating if an empty option should be added to the list

Radio button list

To create a radio button list control:

```
var control = form_newControl(attrId, 'radiolist', options);
```

This will create a list of radio elements.

The options include:

source

The source of the list (an object; see simple list example)

Checkbox list

To create a checkbox list control:

```
var control = form_newControl(attrId, 'checklist', options);
```

This will create a list of checkbox elements.

The options include:

source

The source of the list (an object; see simple list example)

GRID

Definition and properties

A grid is a complex type of attribute. In order to be able to use grids in a GARGANTUA form you must first define a type of grid , then you have to create an attribute of this type which can be used in the form.

A grid presents the data (content) in a table format and it allows the data to be added, viewed, modified in each editable cell. More, the grid can be customized to contain a set of internal features (defined by embedded functions and events handlers) but also to interact externally with other attributes of the form through form scripting. An attribute of grid type can be built assisted by the Grid Designer tool or it is fully scriptable and it can be build in pure Javascript language in a form script.

The grid properties:

`Columns`

defines the grid's structure

`Header`

defines the grid heading properties and style

`Toolbar`

provides built-in and user-defined features (actions)

`Functions`

collection of user-defined functions (Javascript source code)

`Events`

your own event handlers for the events exposed by the grid (Javascript source code)

Columns

The column properties:

`id`

the column identification string, it is unique for all columns and must respect the W3C recommandation for HTML ID attribute definition

`width`

numeric, represent the column width in pixels

`type`

the column's type `[string|icon|button|link|checkbox|list|date|time|timestamp|timespan|integer|float|currency|email|`

`label`

the column's name

`tooltip`

the column's tooltip

`defaultValue`

a value used for a corresponding cell when a new grid row is created; for now only columns having type `icon` (use any valid URL for an image or local images such as »${baseURL}/images/controls/grid/icons/grid-xxx.gif » where xxx is any of the values specified for toolbar icon) or `checkbox` (use value '1' or 'true' to set a checkbox as checked) are supported;

`align`

the cell text horizontal alignment `[left|center|right]`

`readonly`
: the cell is not editable, but it can be activated and clickable

`editable`
: the cell is not editable, it can not be activated nor clickable

`onload`
: for list types only - callback function to provide the options values

`resetonload`
: for list types only - specify if `onload` shall be always invoked;

Header

The grid heading has the following properties:

`visible`
: toggles the visibility of the header.

`compact`
: specifies if the header is displayed on a single or multiple lines.

Toolbar

The grid toolbar has the following properties:

`visible`
: toggles the visibility of the toolbar.

`align`
: specifies the alignment of the toolbar. `[top|right|bottom|left]`

The toolbar contains three types of buttons: `[standard|custom|separator]`.

Standard buttons:

`gridButtonAddRow`
: add a new row

`gridButtonRemoveRow`
: remove the selected row

`gridButtonMoveRowUp`
: move the selected row one position up

`gridButtonMoveRowDown`
: move the selected row one position down

Button properties:

`type`
: `[standard|custom|separator]`

`id`
: the button identification string, it is unique for all buttons and must respect the W3C recommandation for HTML ID attribute definition

`visible`
: toggle the button visible status (default yes)

`icon`
: selected from a limited set of icons (»database », « delete », « disk », « down », « exclamation », « eye », « globe », « minus », « pencil », « plus », « question », « star », « tick-red », « tick », « up »)

`onclick`
: associate an existing function as click event handler (triggered when the button is clicked)

`title`
: tooltip text

`text`
: label text

There is a button specific `onclick` event handler and a global `toolbarbuttonclicked` event handler triggered for all toolbar buttons.

The buttons order inside toolbar can be changed with drag and drop in Grid Designer UI.

Functions

The functions can be used to implement an internal feature or to be used as an event handler (ex: `onclick` handler).

A function is callable from another function like this:

```
this.callFunction(functionName, paramsObject);
```

Returned value is optional and shall be specified like this:

```
params.value = '...';
```

Events

Each event handler has two variables already defined: `this` (the grid object) and `params` object (this object is used to provide important parameters depending on event such as `params.row, params.col, params.from, params.to, params.id`).

`loaded`

triggerred when `load` method (grid definition and data) has been finished;

no params defined

`dataloaded`

triggerred when `loadData` method has been finished;

no params defined

`changed`

triggerred when a cell has been modified;

params defined: `params.row` (number - cell's row), `params.col` (number - cell's column)

`focused`

triggerred when a cell has been focused;

params defined: `params.row` (number - cell's row), `params.col` (number - cell's column)

`clicked`

triggerred when a cell has been clicked;

params defined: `params.row` (number - cell's row), `params.col` (number - cell's column)

`rowmoved`

triggerred when a grid row has been moved up or down;

params defined: `params.from` (number - row initial index), `params.to` (number - row final index)

`toolbarbuttonclicked`

triggerred when a toolbar button has been clicked;

params defined: `params.id` (string - button's id)

Example of event handler for `toolbarbuttonclicked`:

```
var buttonId = params.id;

if (buttonId == "cmdShowValue")
{
```

```
        alert( this.getValue());
}
else if (buttonId == "gridButtonAddRow")
{
        var row = this.getRowsCount() - 1;
        var iLink = this.getColumnIndexById("cLink");
        var iIcon = this.getColumnIndexById("cIcon");
        var iButton = this.getColumnIndexById("cButton");

        this.setCellValue(row, iLink, "This is link-"+row);
        this.setCellValue(row, iButton, "button-"+row);
        var base = "${baseURL}/images/controls/grid/icons/grid-";
        var arrIcons = ["question.gif","exclamation.gif"];
        this.setCellValue(row, iIcon, base + arrIcons[row % 2]);
}
else if (buttonId == "cmdAutoSize")
{
        this.autoSize();
}
```

Scripting the grid

Creating a grid object:

```
var grid = form_newControl(attrId, 'grid', options);
```

The options include:

url
: the URL from which to load the grid definition (XML)

definition
: the grid definition (XML)

loaded
: callback function invoked after the grid is initialized

The url and definition options are exclusive.

Load grid data:

```
grid.load(dataURL, true);
```

Event binding:

```
// bind 2 anonymous functions to the same event

grid. bindEvent('loaded', function() { log('loaded listener-1'); });

grid. bindEvent('loaded', function() { log('loaded listener-2'); });
```

```
// bind changed and focused events

grid. bindEvent('changed', function(param) { log('item ('+ param.row + ',' + param.col
+') changed' ); });

grid. bindEvent('focused', function(param) { log('item ('+ param.row + ',' + param.col
+') focused' ); });
```

Functions

addColumn

```
addColumn(props)
```

Adds a column specified by provided props to the grid.

Parameters

props

An object with the column properties

Return value

None

Example

```
var props = {
     'id': 'productName',
     'type': 'string',
     'label': 'Product',
     'tooltip': 'Product name',
     'align': 'left',
     'width': '250px'
};
grid.addColumn(props);
```

addToolbarButton

```
addToolbarButton(props)
```

Adds a button specified by provided props to the grid toolbar.

Parameters

props

An object with the button properties

Return value

None

Example

```
var props = {
    'id': 'open',
    'type': 'custom',
    'title': 'Open document',
    'text': 'Open'
};
grid.addToolbarButton(props);
```

append

```
append(noEdit)
```

Append a row at the end of the grid. It uses defaultValues to fill up the cells.

Parameters

`noEdit`

boolean that specifies if the append operation should not trigger the edit of the first visible cell in the newly added row

Return value

The newly added `tr` DOM element.

Example

```
grid.append();
```

or

```
var tr = grid.append();
```

appendBefore

```
appendBefore(tr, noEdit)
```

Append a row before the given table row. It uses defaultValues to fill up the cells.

Parameters

`tr`

the `tr` DOM element before which to append the new row

`noEdit`

boolean that specifies if the append operation should not trigger the edit of the first visible cell in the newly added row

Return value

The newly added `tr` DOM element.

Example

```
var tr = grid.getRow(2);
var newTr = grid.appendBefore(tr);
```

bindEvent

```
bindEvent(eventname, callback)
```

Bind a callback function to an event.

Parameters

eventName

`loaded | dataloaded | changed | focused | clicked | rowmoved | toolbarbuttonclicked`

The event to bind

callback

callback function assigned to event

Return value

Returns true for success, false if fails.

Example

```
// bind focus event
grid.bindEvent('focused', function(param) { log('item ('+ param.row + ',' + param.col
+') focused' ); });
```

appendAfter

```
appendAfter(tr, noEdit)
```

Append a row after the given table row. It uses defaultValues to fill up the cells.

Parameters

`tr`

the `tr` DOM element after which to append the new row

`noEdit`

boolean that specifies if the append operation should not trigger the edit of the first visible cell in the newly added row

Return value

The newly added `tr` DOM element.

Example

```
var tr = grid.getRow(2);
var newTr = grid.appendAfter(tr);
```

callFunction

```
callFunction(name, params)
```

Call an internal function (created with the Grid Designer UI) specified by `name` and `param` parameters object.

Parameters

`name`

The name of the function to call

`params`

The parameters for the function to call

Return value

None.

Example

editCell

```
editCell(td, options)
```

Activates a cell editor for the specified `td` DOM element.

Parameters

td

The cell to edit

options

An object with the parameters; for now the only available option is `triggerClick` (boolean - if `true`, force an additional click event)

Return value

None

Example

```
grid.editCell(td, {
    'trigger_click': true
});
```

editCheckbox

```
editCheckbox(td)
```

Activates a checkbox type cell for the specified `td` DOM element.

Parameters

td

The cell to edit

Return value

None

Example

```
grid.editCheckbox(td);
```

forceEditCell

```
forceEdit(td)
```

More than editCell, if the grid is read only, it will force the display of the cell control (in read only mode).

Parameters

td

The cell to activate edit for.

Return value

None

Example

```
grid.forceEditCell(td);
```

getCell

```
getCell(row, column)
```

Returns the cell as `td` DOM element specified by `row` index and `column` index.

Parameters

row

The row

column

The column

Return value

Returns the cell as `td` DOM element specified by `row` index and `column` index.

Example

```
var td = grid.getCell(1, 1);
```

getCellData

```
getCellData(row, column, name)
```

Returns the data having `name` key associated to a cell specified by `row` and `column`.

Parameters

row

The row of the cell

column

The column of the cell

name

The name of the property to retrieve

Return value

Returns the data having `name` key associated to a cell specified by `row` and `column`.

Example

```
var value = grid.getCellData(1, 2, 'test');
```

getCellValue

```
getCellValue(row, column)
```

Returns (string) the cell value specified by `row` index and `column` index.

Parameters

row

The row of the cell

column

The column of the cell

Return value

The cell value specified by `row` index and `column` index.

Example

```
var value = grid.getCellValue(2, 3);
```

getColsCount

```
getColsCount()
```

Return (number) the columns count.

Parameters

None

Return value

The columns count.

Example

```
var columns = grid.getColsCount();
```

getColumnIndexById

```
getColumnIndexById(columnId)
```

Returns the numeric index of a column specified by `columnId`, or -1 if not found.

Parameters

columnId

The column identifier

Return value

The numeric index of a column specified by `columnId`.

Example

```
var column = grid.getColumnIndexById('col1');
```

getColumnProps

```
getColumnProps(column)
```

Get the properties of a grid column specified by `column` index.

Parameters

column

The column index.

Return value

The properties (object) of a grid column.

Example

```
var props = grid.getColumnProps(1);
```

getRow

```
getRow(row)
```

Returns the row as `tr` DOM element specified by `row` index.

Parameters

row

The row index to retrieve

Return value

The row as `tr` DOM element specified by `row` index.

Example

```
var tr = grid.getRow(2);
```

getRowIndex

```
getRowIndex(tr)
```

Returns the numeric row index.

Parameters

tr

The DOM element row to retrieve the index for.

Return value

The numeric row index.

Example

```
var row = grid.getRowIndex(tr);
```

getRowsCount

```
getRowsCount()
```

Return (number) the rows count.

Parameters

None

Return value

The rows count.

Example

```
var rows = grid.getRowsCount();
```

getSelectedRow

```
getSelectedRow()
```

Returns the selected row `tr` DOM element or `null` if not found.

Parameters

None

Return value

Returns the selected row `tr` DOM element or `null` if not found.

Example

```
var tr = grid.getSelectedRow();
```

getToolbarButtonProps

```
getToolbarButtonProps(id)
```

Get the properties of the toolbar button identified by the given `id`.

Parameters

`id`

The button identifier

Return value

The properties (object) of a grid toolbar button.

Example

```
var props = grid.getToolbarButtonProps('open');
```

getValue

```
getValue()
```

Returns the grid value as JSON string.

Parameters

None

Return value

The grid data as JSON string.

Example

```
var data = grid.getValue();
```

hasError

```
hasError()
```

Returns `true` if the grid has at least one row or one cell marked as invalid or if the grid is marked as required but has not rows.

Parameters

None

Return value

`true` if the grid has at least one row or one cell marked as invalid or is empty while being required.

Example

```
if (grid.hasError())
{
    ...
}
```

groupRowsByColumn

```
groupRowsByColumn(column)
```

Scan the cells of the grid column specified by `column` index from top to bottom and merge the cells having the same value.

Parameters

column

The column index to scan

Return value

None

Example

isCellEditable

```
isCellEditable(row, column)
```

Returns true if a specified cell is editable.

Parameters

row

The row that contains the cell

column

The column that contains the cell

Return value

`true` if the specified cell is editable.

Example

```
var isEditable = grid.isCellEditable(5, 4);
```

isCellReadOnly

```
isCellReadOnly(row, column)
```

Returns true if a specified cell is read only.

Parameters

row

The row that contains the cell

column

The column that contains the cell

Return value

`true` if the specified cell is read only.

Example

```
var isReadOnly = grid.isCellReadOnly(5, 4);
```

isFunction

```
isFunction(name)
```

Returns `true` if a function specified by `name` exists (created with the Grid Designer UI).

Parameters

name

The function name to check for

Return value

`true` if the function specified by `name` exists.

Example

```
grid.isFunction('internal_fxn');
```

hasRowErrors

```
hasRowErrors(row)
```

Returns `true` if the specified row has any invalid cell.

Parameters

row

The row to check

Return value

`true` if a specified row has any invalid cell.

Example

isCellInvalid

```
isCellInvalid(row, column)
```

Returns true if the cell specified by `row` and `column` is marked as invalid (has error).

Parameters

row

The row that contains the cell

column

The column that contains the cell

Return value

`true` if the cell specified by `row` and `column` is marked as invalid.

Example

isEmpty

```
isEmpty()
```

Returns `true` if grid is empty.

Parameters

None

Return value

`true` if grid is empty.

Example

isReadOnly

```
isReadOnly()
```

Returns `true` if the grid is read only.

Alias for `isEnabled()`.

Parameters

None

Return value

`true` if the grid is read only.

Example

```
if (!grid.isReadOnly())
{
   ...
}
```

layout

```
layout()
```

Refresh the grid layout, called after a UI option has been changed.

This is called internally after updating the header, toolbar or toolbar buttons.

Parameters

None

Return value

None

Example

```
grid.layout();
```

moveup

```
moveup(tr)
```

Move the row specified by `tr` DOM element one position up. If tr not specified, move the selected row.

Parameters

tr

The row to move

Return value

None

Example

```
var tr = grid.getSelectedRow();
grid.moveup(tr);
```

movedown

```
movedown(tr)
```

Move the row specified by `tr` DOM element one position down. If tr not specified, move the selected row.

Parameters

tr

The row to move

Return value

None

Example

```
var tr = grid.getSelectedRow();
grid.movedown(tr);
```

remove

```
remove(tr)
```

Delete the row specified by `tr`. If tr not specified, delete the selected row.

Parameters

tr

The row to remove

Return value

None

Example

```
var tr = grid.getSelectedRow();
grid.remove(tr);
```

loadData

```
loadData(url, data)
```

Loads the grid content from a specified url. It triggers `dataloaded` when finished.

Parameters

url

The endpoint for data

data

The parameters to pass with the request

Return value

None

Example

removeAll

```
removeAll()
```

Delete all rows.

Parameters

None

Return value

None

Example

```
grid.removeAll();
```

setCellColor

```
setCellColor(row, column, color)
```

Set a cell background color. The color is a CSS string.

Parameters

row

The row that contains the cell

column

The column of the cell

color

The color to set; the color is a CSS string.

Return value

None

Example

```
grid.setCellColor(0, 0, 'red');
grid.setCellColor(0, 0, '#888');
```

selectRow

```
selectRow(tr)
```

Selects the row specified by the `tr` DOM element.

Parameters

tr

The row to select

Return value

None

Example

```
grid.selectRow(grid.getRow(0)); // select first row!
```

setCellData

```
setCellData(row, column, name, value)
```

Set a data `value` of any type, having the `name` key to a cell specified by `row` and `column`.

Parameters

row

The row of the cell

column

The column of the cell

name

The name of the property to set

value

The value to set

Return value

None

Example

```
var pt = { x:100, y:200 };
grid.setCellData( 0, 0, 'point', pt);
grid.setCellData( 0, 0, 'hasPoint', true);
```

setCellEditable

```
setCellEditable(row, column, status)
```

Set/Unset cell editable.

Parameters

row

The row of the cell

column

The column of the cell

status

Boolean to indicate editable status

Return value

None

Example

```
grid.setCellEditable(0, 0, false);
```

setCellInvalid

```
setCellInvalid(row, column, value)
```

Mark a cell as invalid if `value` is true.

Parameters

row

The row of the cell

column

The column of the cell

value

Cell validity indicator

Return value

None

Example

```
grid.setInvalidCell(0, 0, true);  // invalid
grid.setInvalidCell(0, 1, false); // valid
```

setCellReadOnly

```
setCellReadOnly(row, column, status)
```

Set/Unset cell specified by `row` index, `column` index as read only.

Parameters

row

The row of the cell

column

The column of the cell

status

Boolean to indicate read-only status

Return value

None

Example

```
grid.setCellReadOnly(0, 0, true);
```

setCellValue

```
setCellValue(row, column, value)
```

Set the cell value.

Parameters

row

The row of the cell

column

The column of the cell

value

The value to set

Return value

None

Example

```
grid.getCellValue(2, 3, 'John Doe');
```

setColumnReadOnly

```
setColumnReadOnly(column, status)
```

Set/Unset cells in a column specified by column `index` as read only.

Parameters

column

The column of the cell

status

Boolean to indicate read-only status

Return value

None

Example

```
grid.setColumnReadOnly(1, true);
```

setReadOnly

```
setReadOnly(readonly)
```

Set/Unset grid readonly.

Aliases for `disable()`, `enable()`.

Parameters

readonly

The read-only status for the grid

Return value

None

Example

```
grid.setReadOnly(true);
```

setRowInvalid

```
setRowInvalid(row, value)
```

Mark a row as invalid.

Parameters

row

The row to set as invalid.

value

Row validity indicator

Return value

None

Example

```
grid.setRowInvalid(1, true);
```

setRowReadOnly

```
setRowReadOnly(row, status)
```

Set/Unset a grid row read only.

Parameters

row

The row to set.

value

Row read-only indicator

Return value

None

Example

```
grid.setRowReadOnly(1, true);
```

setValue

```
setValue(value)
```

Sets the value of the grid. Value must be a stringified JSON.

Parameters

value

The new grid content

Return value

None

Example

```
grid.setValue('{"col1":[1,2,3],"col2":["John","Maria","Alice"]}');
```

unbindEvent

```
unbindEvent(eventname)
```

Unbind all binded event handlers for an event.

Parameters

eventName

The name of the event to unbind handlers for

Return value

Returns `true` for success, `false` if fails.

Example

```
grid.unbindEvent('focused');
```

updateEditCell

```
updateEditCell(hide)
```

Saves the value from the editor into grid cell; if `hide` boolean is true then close the editing input.

Parameters

hide

Flag to signal that input field should be destroyed, too.

Return value

None

Example

```
grid.updateEditCell(true);
```

updateToolbarButton

```
updateToolbarButton(id, props)
```

Update the properties of the toolbar button identified by the given `id`.

Parameters

`id`

The id of the toolbar button

`props`

The button properties to change

Return value

None

Example

```
grid.updateToolbarButton('btn1', { 'visible': true, 'text': 'New button' });
```

updateToolbar

```
updateToolbar(props)
```

Update the properties of the grid toolbar.

Parameters

`props`

The toolbar properties to change

Return value

None

Example

```
grid.updateToolbar({ 'visible': true, 'align': 'top' });
```

updateHeader

```
updateHeader(props)
```

Update the properties of the grid header.

Parameters

props

The header properties to change

Return value

None

Example

```
grid.updateHeader({ 'visible': true, 'compact': false });
```

OTHER INPUT FIELDS

File input

To create a file control:

```
var control = form_newControl(attrId, 'file', options);
```

The options include:

dnd

Boolean indicating if the control should be rendered as a drag-and-drop area or as a simple input file

Phone input

To create a phone control:

```
var control = form_newControl(attrId, 'phone', options);
```

The options include:

initialCountry

The ISO code of the default country to select

placeholderNumberType

The number type to use for the placeholder `[MOBILE|FIXED_LINE|FIXED_LINE_OR_MOBILE]`

autoPlaceholder

Set the input's placeholder to an example number for the selected country, and update it if the country changes `[polite|aggressive]`

Fonctions JavaScript de rappel de formulaire

FUN_FORM_PRE_LOAD

```
fun_form_pre_load()
```

Callback function called before creating the form controls and filling the form values.

PARAMETERS

None.

RETURN VALUE

None.

EXAMPLE

```
function fun_form_pre_load() {
    try {
        // request necesary data
        var result = form_invokePlugin({
            pluginId: 'publicis',
            command: 'NomLists',
            data: {}
        });
        var json = JSON.parse(result);

        // cache data for later use
        // ...
   }
   catch (e) {
       form_error(e);
   }
}
```

FUN_FORM_LOAD

```
fun_form_load()
```

Callback function called after the form controls have been initialized and the form values have been set.

Parameters

None.

Return value

None.

Example

```
function fun_form_load() {
    // init form values with defaults
    setElementKey(gfxGetAttrIdByLabel('userId'), getElementValue('username'));


    // init custom buttons
    form_displayCustomButtonBar({ ... });
}
```

FUN_FORM_POST_LOAD

```
fun_form_post_load()
```

Callback function called at the very end of the form load sequence.

PARAMETERS

None.

RETURN VALUE

None.

EXAMPLE

FUN_FORM_DISPLAYCONTROLS

```
fun_form_displayControls(displayMode)
```

Callback function called after enabling/disabling the controls based on the display mode.

PARAMETERS

displayMode

The current display mode of the form, which can be one of the following values (both symbolic and numeric constants can be used):

```
0 = DISPLAY_NORMAL
1 = DISPLAY_READONLY
2 = DISPLAY_DISABLEDINPUTS
4 = DISPLAY_DISABLEDBUTTONS
```

RETURN VALUE

None.

EXAMPLE

FUN_FORM_PRE_LOADATTRS

```
fun_form_pre_loadattrs(data)
```

Callback function called before setting the attribute values inside the form.

PARAMETERS

data

An object containing two objects: attributes and rights. The attributes object contains the values to set (each object consists of an attributeId-attributeValue pairing). The rights object contains information on weather the user can view and/or modify the values.

RETURN VALUE

None.

EXAMPLE

FUN_FORM_POST_LOADATTRS

```
fun_form_post_loadattrs(data)
```

Callback function called after setting the attribute values inside the form.

Parameters

data

An object containing two objects: attributes and rights. The attributes object contains the values to set (each object consists of an attributeId-attributeValue pairing). The rights object contains information on weather the user can view and/or modify the values.

Return value

None.

Example

FUN_FORM_PENDING_VALIDATE

```
fun_form_pending_validate()
```

Callback function called before actually starting the validation process.

PARAMETERS

None.

RETURN VALUE

None.

EXAMPLE

FUN_FORM_PRE_VALIDATEATTRS

```
fun_form_pre_validateattrs(data)
```

Callback function called before starting the validation process.

Parameters

data

An object containing attributeId-attributeValue pairings that will be sent on the server for validation.

Return value

None.

Example

FUN_FORM_PRE_VALIDATEREPONSE

```
fun_form_pre_validatereponse(failedIds, failedReasons, failedCodes)
```

Callback function called immediately after receiving the validation response but before marking the failed values. Allows you to modify the attributes that will be marked as failed by adding or removing from the `failed*` arrays.

> ⚠ *The arrays must be kept in sync, so if you add something to the « `failedIds` » you have to also add to the other two arrays as well.*

PARAMETERS

failedIds

Array of attribute ids that have failed validation.

failedReasons

Array of reasons that describe the failure.

failedCodes

Array of failure codes.

RETURN VALUE

None.

EXAMPLE

```
function fun_form_pre_validatereponse(failedIds, failedReasons, failedCodes) {
            var date = gfxVal('date');
            var startDate = gfxVal('startDate');


            if (startDate.trim() !== '' && date.trim() !== '') {
                var sdate = form_parseDate(startDate);
                sdate.setHours(0);
                sdate.setMinutes(0);
                sdate.setSeconds(0);
                sdate.setMilliseconds(0);
```

```
                var adate = form_parseDate(date);
                adate.setHours(0);
                adate.setMinutes(0);
                adate.setSeconds(0);
                adate.setMilliseconds(0);


                if (sdate.getTime() < adate.getTime()) {
                    failedIds.push(gfxGetAttrIdByLabel('startDate'));
                    failedCodes.push('invalid');
                    failedReasons.push('Start date cannot be lower than the date.');
                }
            }
}
```

FUN_FORM_VALIDATE

```
fun_form_validate(result)
```

Callback function called immediately after after marking the failed values.

PARAMETERS

result

An object containing the validation outcome.

The object looks like this:

```
{
    validation : {
        enable : true | false,
        outcome : {
            labeled : true | false,
            anonymous : true | false,
            format : true | false, // true if there are no format failures
            mandatory : true | false, true if there are no mandatory failures
            regex : true | false, // true if there are no regex failures
            formatRegex : true | false, // true if there are no format and regex
failures
            validate : true | false, // true if validation succeeded
            label : true | false, // true if the label is filled
        },
        fulltext : true | false,
    }
}
```

RETURN VALUE

None.

EXAMPLE

FUN_FORM_POST_VALIDATEREPONSE

```
fun_form_post_validatereponse(failedIds, failedReasons, failedCodes)
```

Callback function called immediately after after marking the failed values. You might use it to enable/disable the form buttons based on the validation state.

Parameters

failedIds

Array of attribute ids that have failed validation.

failedReasons

Array of reasons that describe the failure.

failedCodes

Array of failure codes.

Return value

None.

Example

```
function fun_form_post_validatereponse(failedIds, failedReasons, failedCodes) {
      // disable buttons if form not valid
      form_setCustomButtonEnabled('submit', failedIds.length === 0);
}
```

FUN_FORM_GET_VALIDATIONMESSAGE

```
fun_form_get_validationmessage(id, label, reason, code)
```

Callback function that offers the possibility to customize the validation messages that appear as tooltip on invalid controls.

Parameters

id

The invalid attribute's ID.

label

The invalid attribute's label.

reason

The default validation message.

code

The validation code.

Return value

The custom message. If `null`, `empty` or `undefined` is returned then the default validation message will be used.

Example

```
function fun_form_get_validationmessage(id, label, reason, code)
{
  form_info(id + ", " + label);

  switch (code)
  {
    case 'mandatory': // This field is mandatory. You must fill it.
      return 'Custom mandatory message';

    case 'length': // The value entered in this field is too long.
      if (label === 'myattribute')
        return 'The value exceeds the accepted size.';
      else
        return null;
```

```
    // case 'format': // Please check the validity of the data.

    // case 'listsize': // You have selected too many elements from this multiple
selection list.

    // case 'regex': // The value does not match the regular expression specified
for validation.

    // case 'emailsize': // You have selected too many e-mails.

    // case 'mandatorycolumn': // This grid contains mandatory cells. You must fill
them.


    default:

      return null;

  }

}
```

FUN_FORM_PRE_SUBMITFORM

fun_form_pre_submitform(formId, parentId, objectId)

Callback function called before submitting the form.

PARAMETERS

formId

The id of the current form

parentId

The id of the parent object

objectId

The id of the current object

RETURN VALUE

The return value should be a boolean. If `false` then the form submit is stopped, otherwise the submit is allowed to continue.

EXAMPLE

```
function fun_form_pre_submitform(formid, parentid, objectid) {
    generateDocumentNumber();
    return true;
}
```

In the example above a function that generates a number is called before allowing the form to submit.

You can also make certain validations and allow or stop the form submit accordingly.

FUN_FORM_CONFIRM_SUBMITFORM

```
fun_form_confirm_submitform(options)
```

Callback function called after `fun_form_pre_submit` and before actually submitting the form. Allows the display of a confirmation message before submitting the form.

Parameters

options

An object containing the command name, the context and subcontext of the call.

Return value

An object containing the title, message and buttons to display in the confirmation dialog. You can also specify a callback function to be called when one of the buttons is clicked.

Example

```
function fun_form_confirm_submitform(options) {
    switch(options.cmd) {
        case 'save':
        case 'saveClose':
        case 'zap':
            return {
                title : 'Save',
                message : 'Are you sure you want to save this ?',
                callback : function(e, key, btn) {
                    console.log('pressed : ' + btn);

                    switch (btn)
                    {
                        case Button.YES:
                            // do something
                            break;

                        case Button.NO:
```

```
                            // do something
                            break;

                        default:
                            break;
                    }
                }
            };

         default:
            break;
     }
}
```

```
function fun_form_confirm_submitform(options) {
    return {
        title : 'Save',
        message : 'Are you sure you want to save this ?',
        buttons : {
               yes : {
                   label : 'Valider',
                   className : 'btn-default',
                   callback : function(e) {
                       console.log('validate');
                    }
               },
               no : {
                    label : 'Fermer'
               }
           }
    };
}
```

FUN_FORM_POST_SUBMITFORM

```
fun_form_post_submitform(formId, parentId, objectId, success)
```

Callback function called after submitting the form.

Parameters

formId

The id of the current form

parentId

The id of the parent object

objectId

The id of the current object

success

Boolean indicating if the submit was successful or not

Return value

None.

Example

```
function fun_form_post_submitform(formId, parentId, objectId, success) {
    form_displayAlert({
        type : success ? 'success' : 'error',
        message : success ? 'Success message ...' : 'Error message ...'
    });
}
```

FUN_FORM_PRE_SENDTASK

```
fun_form_pre_sendtask(formId, parentId, objectId)
```

Callback function called before finishing a task. Can stop finish task if returned value is `false`.

This callback is called only for forms displayed in inbox.

PARAMETERS

formId

The id of the current form

parentId

The id of the parent object

objectId

The id of the current object

RETURN VALUE

The return value should be a boolean. If `false` then the task finish is stopped, otherwise the finish operation is allowed to continue.

EXAMPLE

FUN_FORM_GET_EDITNATIVE_OPTIONS

```
fun_form_get_editnative_options(id)
```

Callback function called when an edit native operation is requested in order to get the overwrite options for it.

Parameters

id

The id of the document being edited.

Return value

Return a JSON object containing the version property with one of the following values: `overwrite | major | minor`.

Example

```
function fun_form_get_editnative_options(id) {
    // add check if the doc's parent id is the current process and compute edit native
options
    return {
        version : 'major' // minor or overwrite
    };
}
```

FUN_FORM_REFRESH_DOCS

```
fun_form_refresh_docs()
```

Callback function called when documents were added, edited, deleted, pinned, unpinned, pasted, rolled back or splitted.

PARAMETERS

None.

RETURN VALUE

None.

EXAMPLE

```
function fun_form_refresh_docs() {
    // check existence of a certain document using form_hasDocuments
or form_getDocuments methods
}
```

FUN_FORM_PRE_ESCALATETASK

```
fun_form_pre_escalatetask(formId, parentId, objectId)
```

Callback function called before escalating a task. Can stop escalate task if returned value is `false`.

This callback is called only for forms displayed in inbox.

Parameters

formId

The id of the current form

parentId

The id of the parent object

objectId

The id of the current object

Return value

The return value should be a boolean. If `false` then the task escalate is stopped, otherwise the escalate operation is allowed to continue.

Example

FUN_FORM_GET_OCR_ATTRIBUTES

```
fun_form_get_ocr_attributes()
```

Callback function called in order to retrieve the list of attribute ids that can be used when performing OCR.

PARAMETERS

None.

RETURN VALUE

The function should return an array of valid attribute ids.

EXAMPLE

```
var ocrAttrs = ['label', 'alfa', 'text']; // or ids


function fun_form_get_ocr_attributes() {
    var ids = [];
    for (var i = 0; i < ocrAttrs.length; i++)
        ids.push(gfxGetAttrIdByLabel(ocrAttrs[i]));


    return ids;
}
```

FUN_FORM_SET_OCR_ATTRIBUTES

```
fun_form_set_ocr_attributes(attributes)
```

Callback function called after an OCR text to attribute association (after setting the values that were retrieved through OCR).

Parameters

attributes

Array of objects containing attribute id to value pairs.

Return value

None.

Example

```
function fun_form_set_ocr_attributes(attributes) {
    form_info(attributes);
    // perform necessary actions post OCR value set
}
```

FUN_FORM_POST_SIGN_DOCUMENTS

```
fun_form_post_sign_documents(data)
```

Callback function called after a document sign performed in the sign viewer (according to the task configuration). This allows you to finish the current task after the user has signed all documents.

PARAMETERS

data

An object containing :

- `ids` - array of signed document ids
- `count` - number of documents still left to sign

RETURN VALUE

None.

EXAMPLE

```
function fun_form_post_sign_documents(data) {
    form_info('Signed : ');
    form_info(data.ids);
    form_info('No. documents left to sign : ' + data.count);

    if (data.count == 0)
        form_finish();
}
```

FUN_FORM_RESET

```
fun_form_reset()
```

Callback function called after a form rest was performed.

Parameters

None.

Return value

None.

Example

SERVICES DISPONIBLES À TRAVERS DES REQUÊTES AJAX

AJAX_SYNCSERVERREQUESTXML

```
function ajax_syncServerRequestXML(strServerUrl)
```

Invoque un service identifié par l'URL.

Paramètres et valeur retour :

strServerUrl

identificateur du service invoqué.

Valeur retournée

Réponse au format XML du service invoqué

/API/SEARCH/ATTR?REQUEST

Format de la requête

```
<Request> ::= <ConditionIds> & <Criteria>
<ConditionIds> ::= <IdAttr> "=" <valueAttr> | <IdAttr> "=" <valueAttr> "&"
<ConditionIds>
<Criteria> ::= <CriteriaObject> | <CriteriaObject> "&" <Criteria>
<CriteriaObject> ::= "processes=1" | "folders=1" | "documents=1" | "fiches=1" |
"pins=1"
<valueAttr> ::= valeur de l'attribut
<IdAttr> ::= Id d'attribut préfixe avec DFAT
```

Exemple :

```
var aReq = "DFAT00010000001000000000000000000000000=siatel";
aReq += "&";
aReq += "folders=1&documents=1&";
var xmlResponse = ajax_syncServerRequestXML( '/api/search/attr?&' + aReq );
```

Format de la réponse

Retourne les ID des objets (dossiers, documents, processus, etc.)

Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<search>
    <results>
       <result>
          <id>DOCF000100000005000000000000000100000000<id>
       </result>
       <result>
          <id> FLDR00010000000500000000000000000000000</id>
       </result>
    </results>
</search>
```

/API/GET?REQUEST

Format de la requête

```
<request> ::= <param1> & <param2>
<param1> ::= "id=" <idObj>
<param2> ::= "mode=" <valueMode>
<valueMode> ::= "simple" | "full"
<idObj> ::= identificateur d'objet préfixé par un préfixe d'objet
```

Exemple :

```
var aReq = "id=DOCF000100000005000000000000000100000000";
aReq += "&";
aReq += "mode=simple";
var xmlResponse = ajax_syncServerRequestXML( '/api/get?' + aReq );
```

Format de la réponse

Retourne les informations sur un objet.

> ***Actuellement le mode full n'est pas implémenté.***

Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<object>
    <objectid> DOCF000100000005000000000000000100000000
    </objectid>
    <objectlabel> document de test </objectlabel>
    <objectdesc></objectdesc>
    <objectattrs>
       <objectattr>
          <attrid>DFAT000100000010000000000000000000000000</attrid>
          <attrvalue>Exemple d'explication</attrvalue>
       </objectattr>
    </objectattrs>
</object>
```

Éléments d'information accessibles à partir du formulaire

Éléments d'information accessibles à partir du formulaire :

userid

Id de l'utilisateur actuellement connecté.

username

Le login de l'utilisateur actuellement connecté.

userrealname

Le nom réel de l'utilisateur actuellement connecté.

search

Indique si le formulaire est utilisé en mode recherche.

multi

Indique si le formulaire est utilisé en mode multiple.

ext_taskName

Nom de la tâche.

ext_processName

Nom de l'instance de processus.

ext_processDefName

Nom de la définition de processus.

Exemple

```
var taskName = document.getElementById( "ext_taskName" ).value;
```

SCRIPTS DE PROCESSUS

QU'EST-CE QU'UN SCRIPT DE PROCESSUS ?

Les processus GARGANTUA permettent d'automatiser et de standardiser une grande partie des traitements effectués dans le cadre de processus métier réels.

> ***Les processus GARGANTUA sont exécutés au niveau du serveur GARGANTUA et non sur les postes clients.***

Leur conception repose sur la **notation Business Process Modeling Notation** (BPMN), au travers de l'Outil d'administration GARGANTUA. Schématiquement, le serveur GARGANTUA contient un moteur de workflow qui exécute les tâches et évalue les conditions pour déterminer le chemin du processus (tâche suivante).

Le moteur GARGANTUA exécute des actions prédéfinies (envoi d'e-mail, ajout dans la GED, etc.) et des actions personnalisées. Ce sont, entre autres, ces dernières qui contiennent des scripts.

Ce document **s'adresse à des développeurs compétents** à la fois en JavaScript et en Java, et ayant préalablement terminé leur formation générale sur les processus.

OÙ ET POURQUOI INSÉRER DES SCRIPTS DANS UN PROCESSUS ?

La plupart du temps, les scripts sont définis dans le corps des tâches automatiques (actions personnalisées). Cependant, ils sont également utilisés dans les tâches de démarrage d'un sous-processus et dans les événements personnalisés.

DANS UNE TÂCHE AUTOMATIQUE

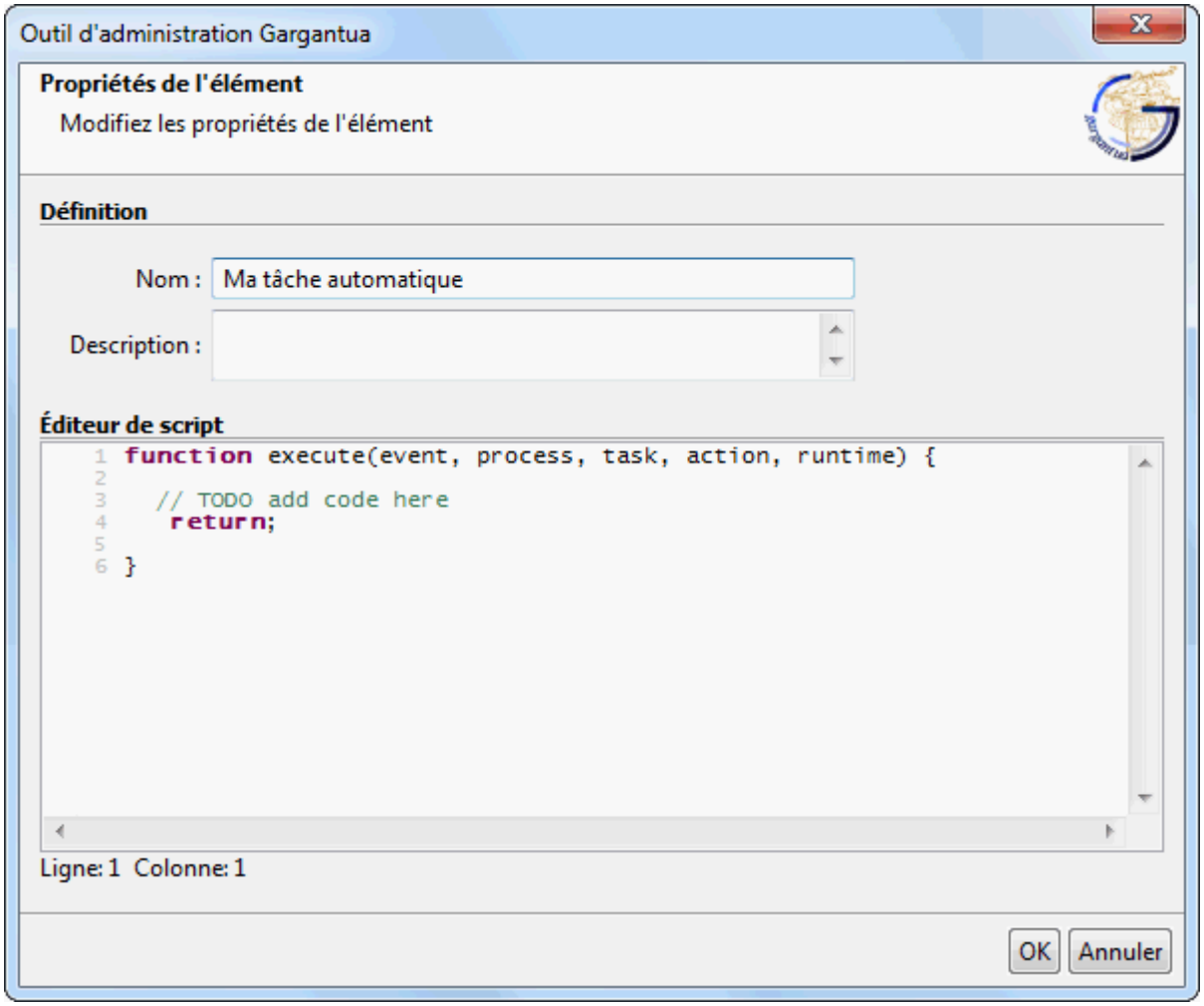
Lorsque le moteur de workflow arrive à une tâche automatique, il exécute son script. Les scripts de ces tâches servent, par exemple, à **générer des valeurs** (identifiants séquentiels), des documents, ou à communiquer avec des **systèmes informatiques externes** à GARGANTUA.

Pour insérer un script dans une tâche automatique, procédez comme suit :

1. Dans le schéma de définition d'un processus, insérez une tâche de type **Tâche automatique** à partir de la palette des éléments disponibles pour le workflow :

2. Double-cliquez sur la tâche automatique.

La fenêtre de propriétés de la tâche s'affiche. La zone **Éditeur de script** permet de modifier le script associé à cette tâche :



Dans une tâche de démarrage d'un sous-processus

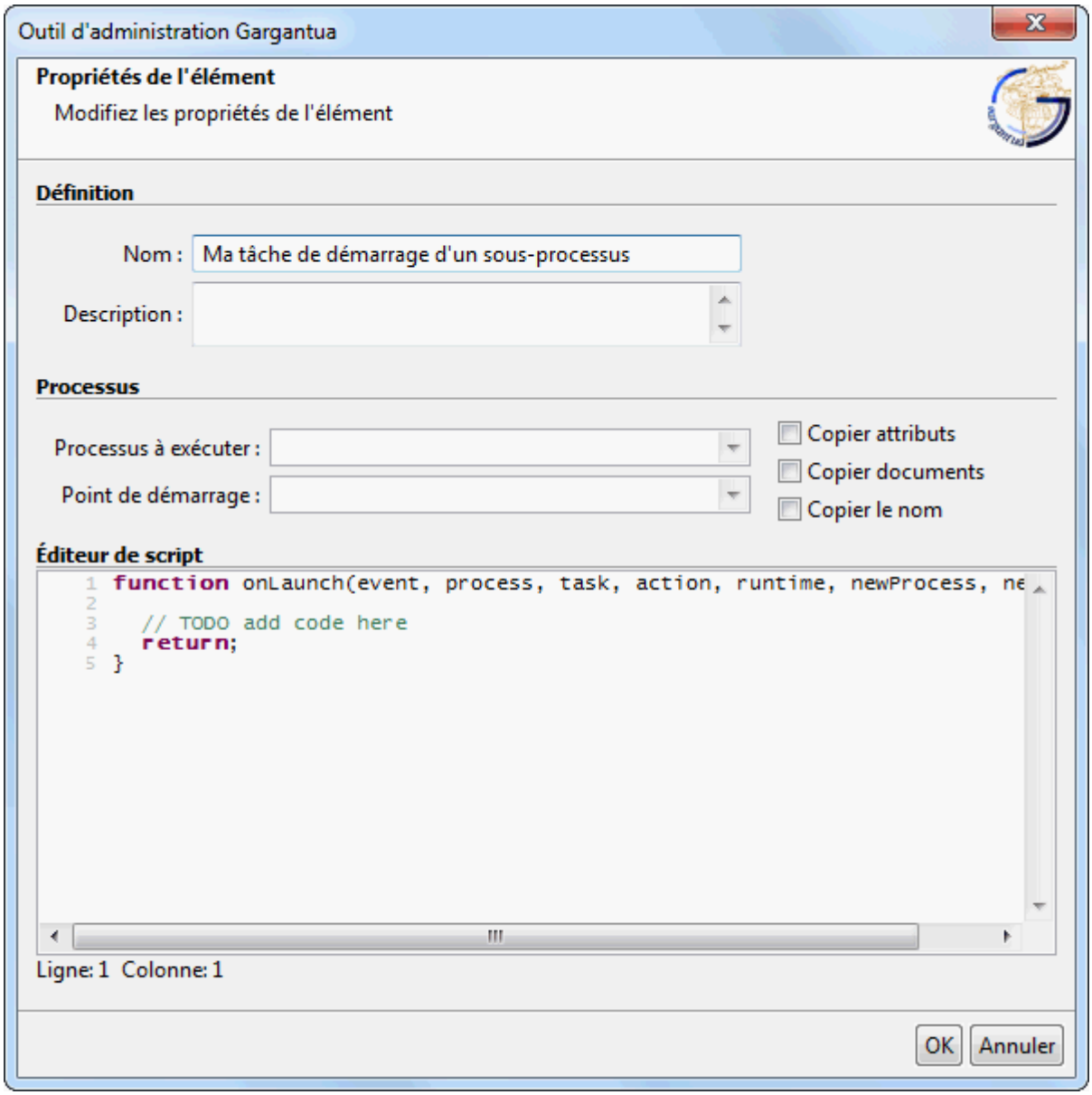
Il est également possible, par le biais d'un script, de personnaliser la façon dont un processus lance un autre processus. Ce type de script permet de **contrôler avec précision les données transmises** par le processus appelant au processus appelé.

Pour insérer un script dans une tâche de démarrage d'un sous-processus, procédez comme suit :

1. Dans le schéma de définition d'un processus, insérez une tâche de type **Démarrer un processus** à partir de la palette des éléments disponibles pour le workflow :

2. Double-cliquez sur la tâche de démarrage d'un sous-processus.

   La fenêtre de propriétés de la tâche s'affiche. La zone **Éditeur de script** permet de modifier le script associé à cette tâche :



DANS UN ÉVÉNEMENT PERSONNALISÉ

Les scripts peuvent être utilisés dans le cadre des événements personnalisés. Les événements personnalisés sont des points du processus où le moteur de workflow met temporairement en pause l'exécution du workflow jusqu'à ce qu'un événement se produise. Pour que ce soit possible, le moteur de workflow exécute un script à intervalle régulier. Ce dernier est chargé d'effectuer les traitements qui **détectent si l'événement est survenu**, puis de retourner un booléen pour informer le moteur que l'événement s'est produit (`true`) ou non (`false`).

Pour insérer un script dans un événement personnalisé, procédez comme suit :

1. Dans le schéma de définition d'un processus, insérez une tâche de type **Événement personnalisé** à partir de la palette des éléments disponibles pour le workflow :

2. Double-cliquez sur l'événement.

   La fenêtre de propriétés de l'événement s'affiche. La zone **Éditeur de script** permet de modifier le script associé à cet événement :

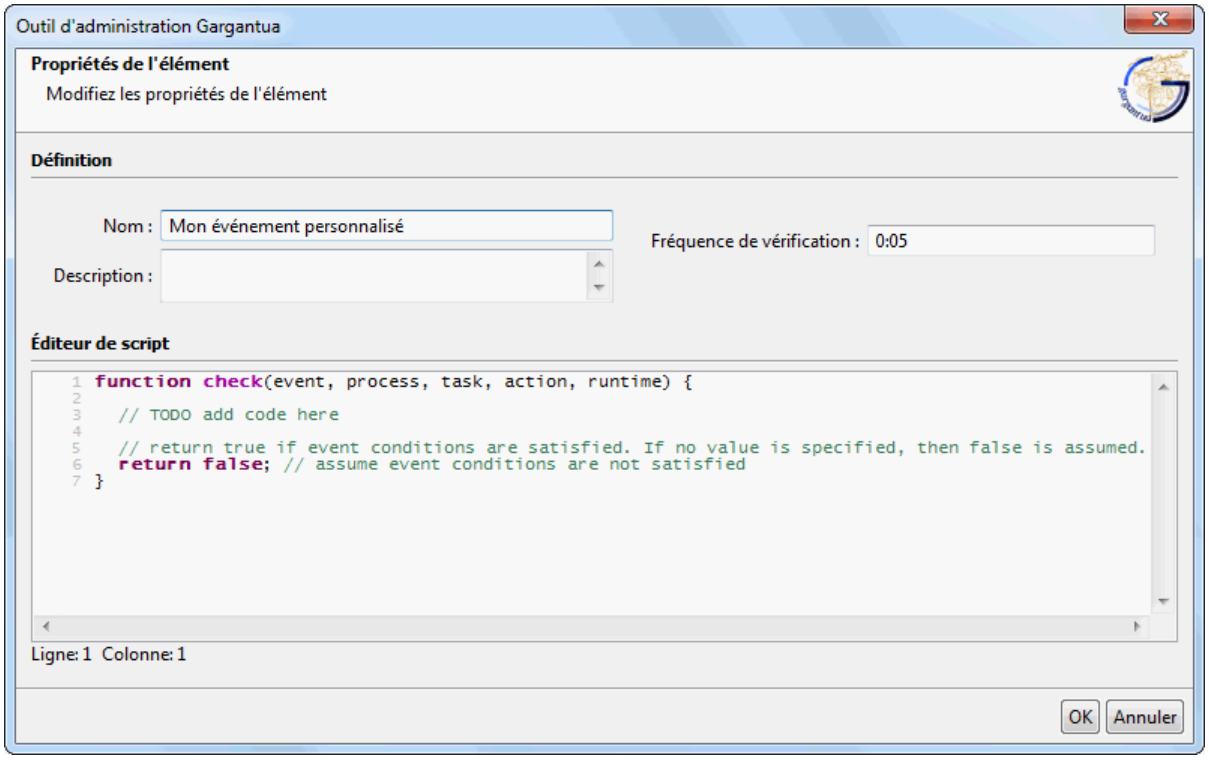


Tableau récapitulatif

Le tableau ci-dessous récapitule les différents points d'entrée et les principales fonctions des scripts associés :

Point d'insertion	Rôles
 Tâche automatique	Automatiser tout ce qui peut l'être dans le cadre d'un processus, en particulier : • La génération automatiques de valeurs (ex. : identifiants) ; • La génération automatiques de documents ; • La communication avec des composants logiciels externes.

 Tâche de démarrage d'un sous-processus	Configurer précisément la façon dont le sous-processus est lancé et les infomations qui lui sont transmises
 Événement personnalisé	Détecter qu'un événement se produit, en particulier pour savoir si l'exécution d'un sous-processus est terminée

L'éditeur de script est toujours inclus dans la fenêtre qui s'ouvre lorsqu'on double-clique sur l'élément correspondant du schéma.

SYNTAXE DES SCRIPTS DE PROCESSUS ET CLASSES DISPONIBLES

Pour simplifier, on peut dire que la syntaxe des scripts de processus GARGANTUA est celle de JavaScript, et que la bibliothèque de classes et de fonctions est celle de Java, étendue par les classes de l'API GARGANTUA.

RAPPELS DE JAVASCRIPT VALABLES POUR LES SCRIPTS DE PROCESSUS

Le langage JavaScript offre de nombreuses possibilités rarement exploitées. Nous commencerons par deux exemples pour présenter ensuite une fonctionnalité plus avancée du langage, mais qui est pertinente dans le cadre des scripts de processus.

DÉCLARER DES TABLEAUX ASSOCIATIFS

En JavaScript, vous pouvez déclarer des tableaux associatifs (équivalent des Map en Java) comme suit :

```
// déclaration d'un tableau associatif
var paul = {
   prenom: "Paul",
   age: 35
};
```

On peut imbriquer des tableaux associatifs les uns dans les autres. On accède alors aux éléments par le biais de l'opérateur . (point) :

```
paul.prenom
```

Déclarer une variable

Pour déclarer une variable, écrivez le mot-clé `var`, suivi du nom de la variable :

```
var unNomDeVariable;
```

Déclarer une fonction

La façon la plus simple de déclarer une fonction consiste à utiliser le mot-clé `function`, suivi du nom de la fonction, de ses paramètres entre parenthèses, puis des instructions à exécuter dans un bloc `{ ... }` :

```
function direBonjour(prenom) {
    return "Bonjour, " + prenom;
}
```

En JavaScript, une fonction est une valeur comme une autre et peut être stockée dans une variable. On peut donc aussi écrire (ce qui est rigoureusement identique) :

```
var direBonjour = function(prenom ) {
    return "Bonjour, " + prenom;
};
```

JavaScript et frameworks Java et Gargantua

Dans le cadre des scripts de processus Gargantua, vous profitez de la souplesse du langage JavaScript combinée à la richesse du framework Java et de l'API Gargantua :

- Les classes Java sont disponibles au travers de **tableaux associatifs prédéfinis**. Vous pouvez donc les instancier par leur nom Java.

  Par exemple, dans les scripts de processus, vous pouvez écrire :

  ```
  var file = new java.io.File("monfichier.txt");
  ```

  ou :

  ```
  var buffer = new java.lang.StringBuilder();
  buffer.append("Hello");
  System.out.println(buffer.toString());
  ```

- Dans les processus Gargantua, le code JavaScript des scripts accède également à **toutes les classes du framework Gargantua** (voir la documentation sur l'API Gargantua). Un aperçu des possibilités offertes par le framework Gargantua sera donné dans les rubriques suivantes et dans certains exercices pratiques de ce document.

Résumé

Quand vous écrivez des scripts de processus, vous devez utiliser la syntaxe du langage JavaScript, mais vous pouvez manipuler des classes et des objets du framework Java en utilisant leur nom complet (avec le chemin).

Points d'entrée (fonctions appelées par le moteur de workflow)

Il existe trois types d'éléments du workflow qui reposent sur des scripts : les tâches automatiques, les tâches de lancement de sous-processus et les événements personnalisés. Pour chacun de ces types d'éléments, le script associé doit définir un point d'entrée sous forme d'une fonction JavaScript.

Il n'est jamais nécessaire de créer ces fonctions à la main car elles sont générées automatiquement dans le corps des scripts correspondants aux tâches et aux événements personnalisés.

Dans les tâches de démarrage de sous-processus : fonction « onLaunch »

La fonction `onLaunch()`, exécutée lorsqu'un sous-processus est appelé, admet deux paramètres de plus que la fonction `execute()` :

```
function onLaunch(event, process, task, action, runtime, newProcess, newRuntime)
```

Alors que les paramètres `process` et `runtime` concernent le processus appelant, les paramètres `newProcess` et `newRuntime` font référence au processus appelé. Il est ainsi possible de passer des données de l'un à l'autre en utilisant ces quatre variables.

Pour cette fonction, il n'est pas nécessaire de préciser une valeur de retour.

Dans les scripts de tâches automatiques : fonction « execute »

Lors de la création d'une tâche automatique, un script par défaut est défini. Celui-ci contient déjà la fonction `execute` qui sera le point d'entrée. Sa définition est la suivante :

```
function execute(event, process, task, action, runtime)
```

Paramètres et valeur retournée :

event

Objet de type « Event » correspondant à l'événement qui a déclenché l'exécution du script.

process

Objet de type « Process » contenant des informations sur le processus en cours.

task

Tâche associée au script.

action

Ce paramètre n'est pas utile pour le moment. Il aura peut-être du sens dans une version prochaine de GARGANTUA.

runtime

Données d'exécution du processus. Il s'agit d'un objet de type « Map » servant de contexte commun à toutes les tâches d'un même workflow. On peut donc y placer des données que l'on souhaite utiliser dans une autre tâche.

Valeur retournée

Cette fonction ne retourne pas de valeur.

DANS LES ÉVÉNEMENTS PERSONNALISÉS : FONCTION « CHECK »

La fonction `check` doit être implémentée dans les événements personnalisés :

```
function check(event, process, task, action, runtime)
```

Les paramètres sont les mêmes que ceux de la fonction `execute()` lors de l'exécution d'une tâche automatique, mais la fonction `check()` doit retourner un booléen :

- `true` signifie que l'événement s'est produit. Le workflow va passer à l'action suivante et la fonction `check()` ne sera plus rappelée.
- `false` signifie que l'événement ne s'est pas produit. L'attente se prolonge et la fonction `check()` sera rappelée plus tard pour évaluer à nouveau si l'événement s'est produit.

GÉRER LES SCRIPTS DE PROCESSUS

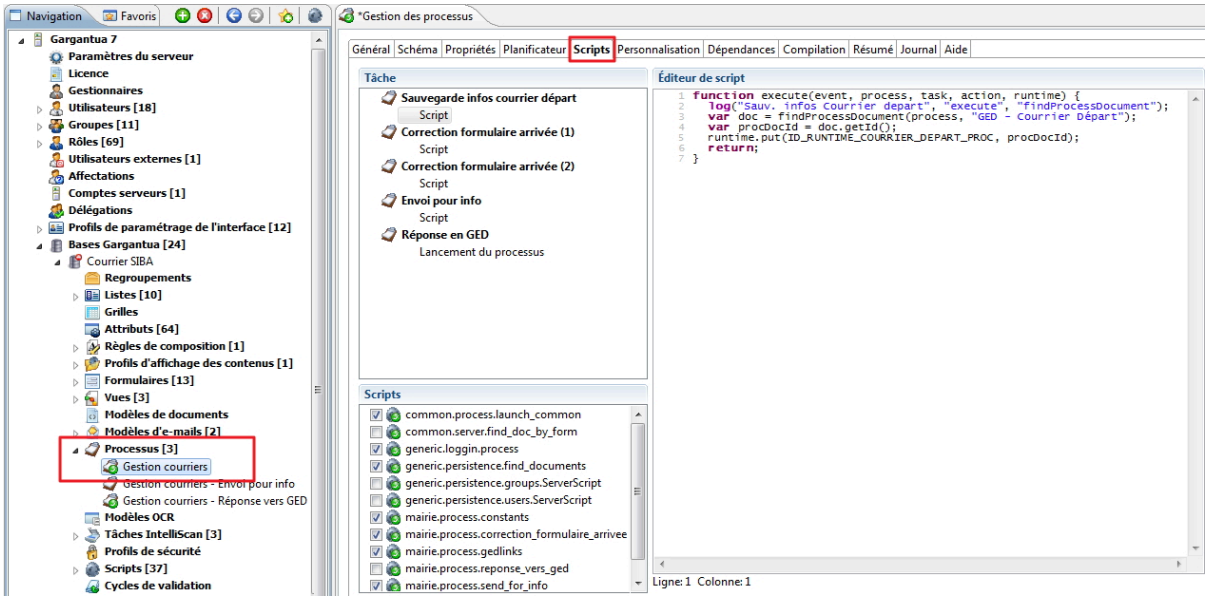
AU NIVEAU D'UNE BASE GARGANTUA

Définir la majorité des scripts au niveau d'une base GARGANTUA est une bonne pratique (voir Créer un script au niveau d'une base GARGANTUA).

Tous les scripts de type **Workflow** de la base GARGANTUA sont disponibles dans l'onglet **Scripts** de l'écran de gestion d'un processus. Vous pouvez cocher la case correspondant au script pour l'inclure dans un élément du processus.

AU NIVEAU D'UN PROCESSUS

Au lieu de double-cliquer sur chaque tâche ou événement contenant un script, vous pouvez consulter tous les scripts d'un même processus depuis l'onglet **Scripts** de l'écran de gestion d'un processus :



Le cadre **Scripts** situé en bas à gauche du volet droit permet d'inclure des scripts définis au niveau de la base GARGANTUA.

EXERCICE : ÉCRIRE UN PREMIER SCRIPT

CONSIGNE

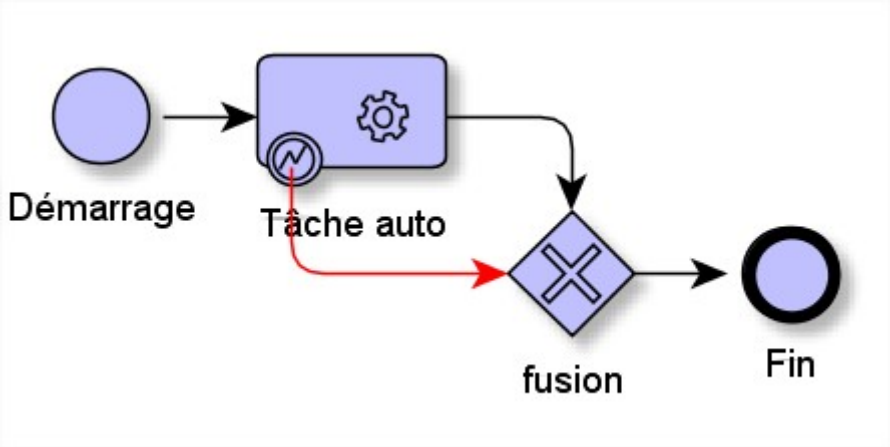
Créez le worfklow le plus simple possible, dans lequel on peut exécuter un script par le biais d'une tâche automatique. Dans ce script, affichez le message « Hello world ».

Aide : il faut utiliser `System.out.println`.

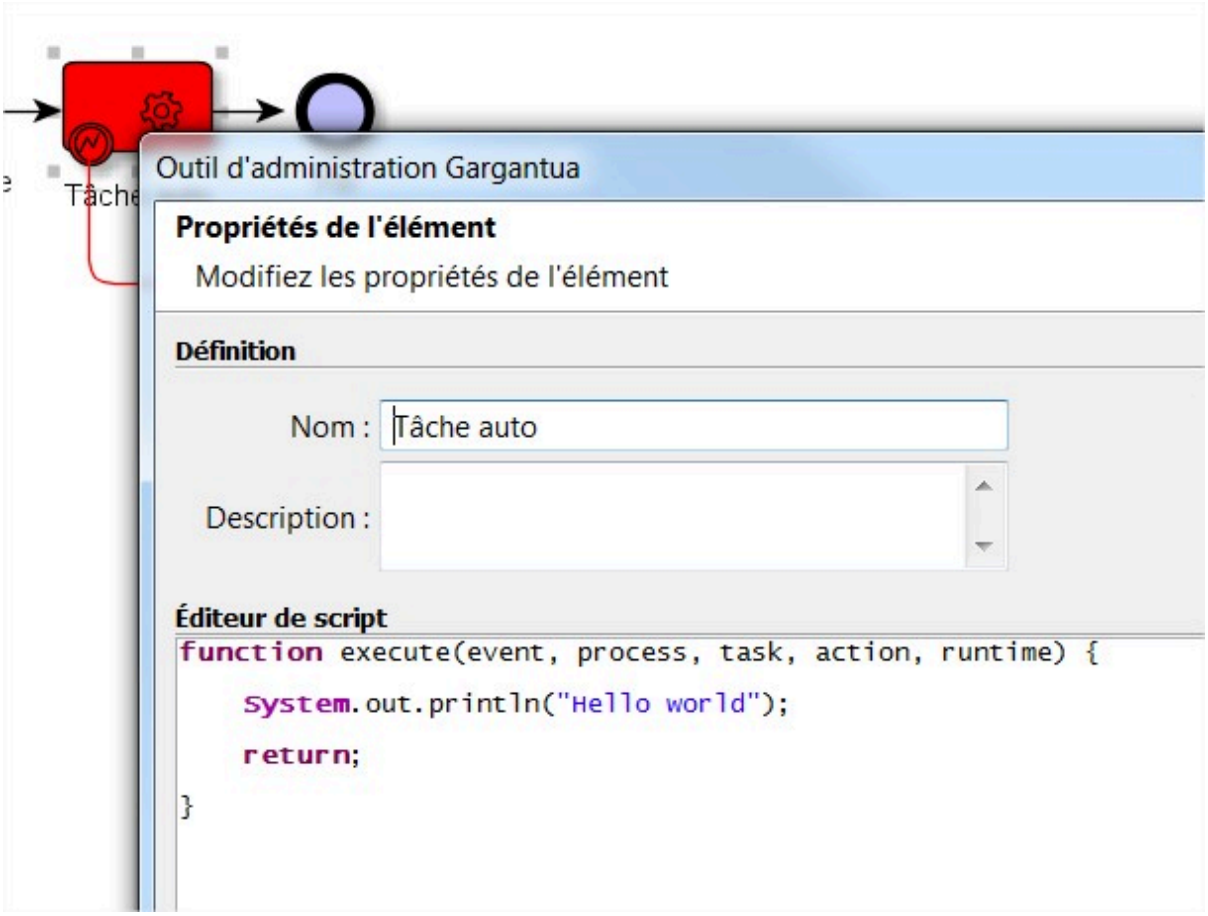
Question supplémentaire : où s'affiche ce message ?

CORRECTION

Voici le workflow le plus simple qui permet d'exécuter un script :



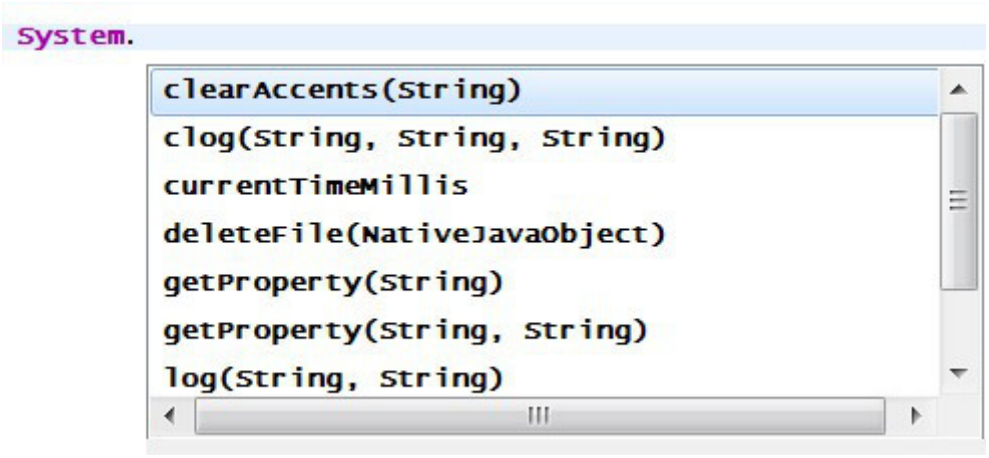
Voici le script :



Le message « Hello world » sera ajouté aux logs du serveur, ce qui n'est vraiment pas pratique quand on met au point un script. D'autres techniques permettent de récupérer des informations sur le déroulement d'un script.

ASTUCE : AFFICHER LA LISTE DES CHOIX CONTEXTUELS

Lorsque vous saisissez un script, il est possible qu'une liste de choix contextuels s'affiche.



Si cette liste ne s'affiche pas automatiquement, vous pouvez utiliser la combinaison de touches Ctrl + espace pour l'afficher.

DÉBOGUER

Il existe deux techniques pour déboguer : l'inclusion de fichiers d'erreurs et le débogueur.

INCLURE DES FICHIERS D'ERREURS DANS LE WORKFLOW

En ajoutant une directive particulière dans un script, les exceptions qu'il génère sont consignées dans un fichier qui est attaché à la tâche suivante.

Pour activer ce mode de dégogage, vous devez ajouter la directive suivante au début du script :

```
//@debug.mode.attachErrors
```

Il est important de ne pas ajouter d'espaces.

UTILISER LE DÉBOGUEUR

Utiliser le débogueur est la solution la plus efficace, mais elle implique de travailler directement sur le serveur GARGANTUA. Après avoir configuré le serveur, vous devez ajouter la directive qui convient aux scripts que vous voulez déboguer. Une fenêtre de débogage complète s'ouvre alors lorsqu'un de ces scripts est sur le point d'être exécuté.

Démarrer le serveur en mode commande

Une fois le serveur configuré, il doit être démarré en mode commande.

Procédez comme suit :

1. Arrêtez le service **Gargantua 7 Server**.
2. Accédez au dossier **apache-tomcat\bin** du dossier d'installation de votre serveur GARGANTUA (par défaut, **C:\siatel\Gargantua7server**).
3. Exécutez le fichier **startup.bat**.

   Le débogueur est prêt à être utilisé.

Configurer le serveur pour utiliser le débogueur

Utiliser le débogueur nécessite une configuration particulière du serveur.

Procédez comme suit :

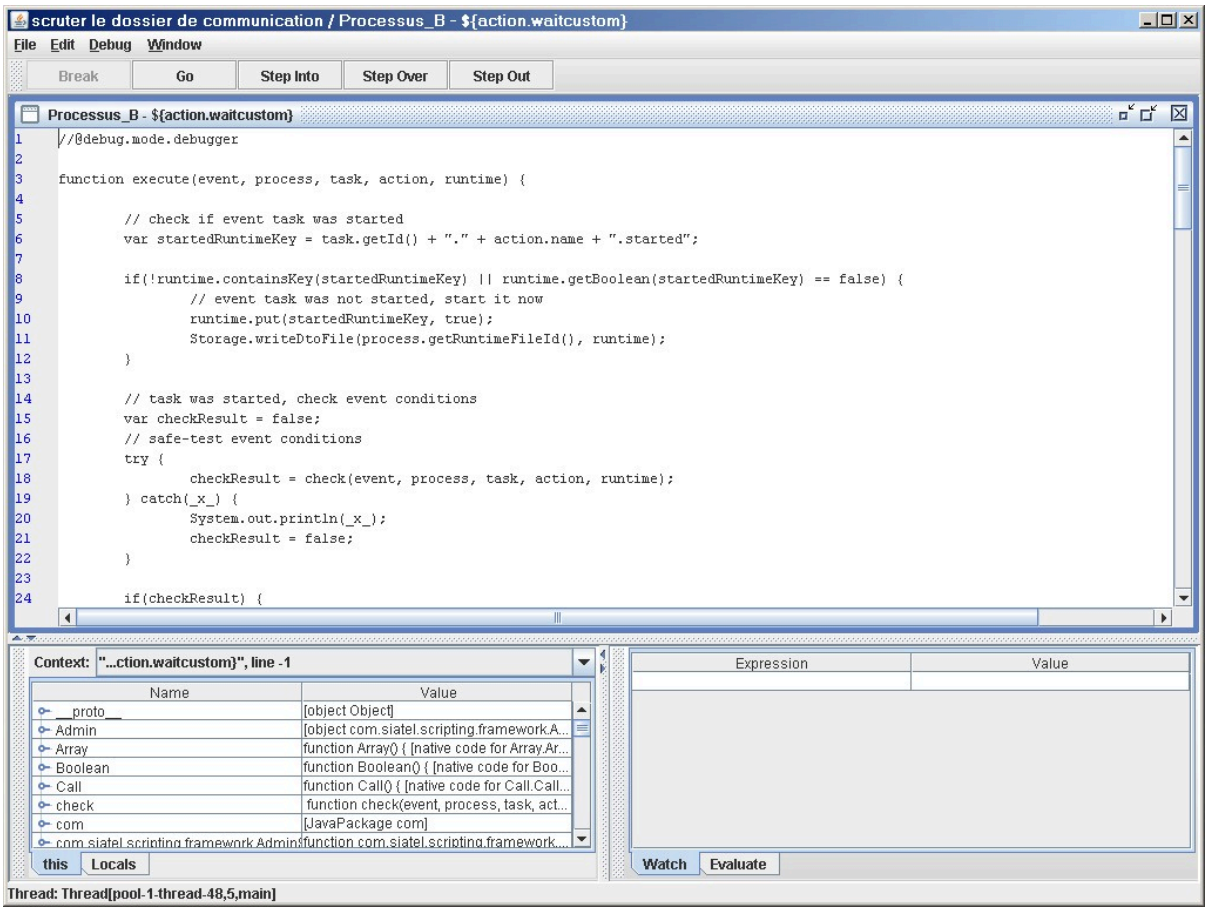
1. Accédez au dossier **apache-tomcat\bin** du dossier d'installation de votre serveur GARGANTUA (par défaut, **C:\siatel\Gargantua7server**).
2. Ouvrez le fichier **catalina.bat** dans un éditeur de texte.
3. Recherchez la ligne commençant par « set JAVA_OPTS ».
4. Au-dessous de cette ligne, ajoutez une nouvelle ligne avec le texte suivant : set JAVA_OPTS=%JAVA_OPTS% -Djs.debugging=enabled
5. Sauvegardez les modifications.

Ajouter une directive en haut des scripts

La directive à ajouter en haut des scripts est la suivante :

```
//@debug.mode.debugger
```

La fenêtre du débogueur se présente comme suit :



Exercice

Testez les deux techniques de débogage expliquées ci-dessus sur le script du workflow créé précédemment.

Analyse de cas : mise à jour du workflow à partir d'un fichier « .properties »

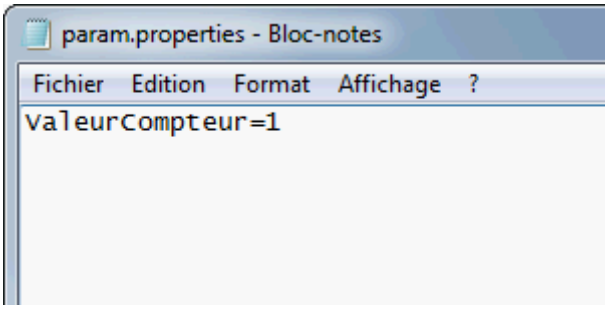
Problème

L'objectif est de stocker une valeur numérique dans un fichier externe (fichier **.properties**) pour qu'elle serve de compteur automatique. À l'aide d'un script, il faut donc :

1. Lire le fichier **.properties** et récupérer la valeur ;
2. Mettre à jour le champ du formulaire ;
3. Incrémenter la valeur ;
4. Mettre à jour le fichier **.properties**.

Pour y parvenir, vous disposez d'une part du framework Java et d'autre part de l'API publique GARGANTUA.

Le fichier **.properties** se présente comme suit :



SCRIPT CORRESPONDANT

Le script suivant répond au problème posé :

```
//@debug.mode.attachErrors

// Chemin du fichier de propriétés
var cheminProprietes = "c:\\work\\test\\param.properties";
var proprietes;

/*
    Point d'entrée
*/
function execute(event, process, task, action, runtime) {

    var numero = chargeNumero();
    majProcessus(process, numero);
    sauvegardeNumero( java.lang.Integer.parseInt(numero) + 1 );

    return;
}


/*
    Chargement du numéro depuis le fichier properties
*/
function chargeNumero() {
    var file = new java.io.File(cheminProprietes);
    var inputStream = new java.io.FileInputStream(file);

    proprietes = new java.util.Properties();
    proprietes.load(inputStream);

    return proprietes.getProperty("valeurCompteur","1");
}

/*
    Mise à jour du processus
*/
function majProcessus(process, numero) {

    // Récupérer l'ID de la base Gargantua
    var databaseId = IdFormat.toString(Prefix.DATABASE, process.getDbId());

    // Récupérer l'ID de l'attribut
    var p1AttrId = Design.idOfAttr("p1", databaseId);

    // Mise à jour de l'attribut du formulaire
    process.setAttributeValue(p1AttrId, numero);

    // Enregistrement des changements
    Flow.updateProcess(process, new java.lang.Long(Flag.ATTRIBUTES));
}

/*
    Enregistrement du numéro
*/
function sauvegardeNumero(numero) {
    proprietes.setProperty("valeurCompteur", numero);
    proprietes.store(new java.io.FileOutputStream(cheminProprietes), new java.util.Date().toString());

}
```

L'attribut du formulaire que nous mettons à jour s'appelle « p1 ».

EXPLICATIONS

Les méthodes `chargeNumero()` et `sauvegardeNumero()` ne contiennent que des utilisations du framework Java standard. Elles ne sont donc pas expliquées ici.

Cependant, la méthode `majProcessus()` fait appel à l'API GARGANTUA :

Note importante sur les types de données

Le script ci-dessus contient le code suivant :

```
new java.lang.Long(Flag.ATTRIBUTES)
```

et le code suivant dans la fonction `execute` :

```
java.lang.Integer.parseInt(numero) + 1
```

Dans les scripts de processus, même si les variables acceptent n'importe quel type de valeur, les opérateurs peuvent donner des résultats différents selon les types manipulés.

Exemple :

- *`"1" + "1"` donnera `"11"`.*
- *`"1" + 1` donnera également `"11"`.*
- *Cependant, `1 + 1` donnera `2`.*

De plus, les types Java ne sont pas traités comme des types du langage. Par exemple, `java.lang.Integer` n'est pas un type numérique mais un objet.

```
new java.lang.Integer(1) + 1 donnera "11" également.
```

En effet, en arrière plan, c'est la méthode `toString()` de `java.lang.Integer` qui est appelée.

Pourtant, certaines méthodes du framework Java ou de l'API Gargantua attendent un type précis d'objet Java. C'est par exemple le cas de `Flow.updateProcess` dont le deuxième paramètre doit être un `java.lang.Long`. Lui passer directement un type numérique JavaScript ne fonctionne pas. Il faudra alors créer un objet de type `Long` avant d'appeler la méthode.

Les types manipulés par le langage des scripts doivent être distingués des types de l'API utilisée et les conversions de l'un vers l'autre doivent être faites sous peine d'obtenir des résultats incohérents ou des erreurs.

Utiliser l'API Gargantua pour mettre à jour les données du processus (méthode « majProcessus »)

La méthode `majProcessus` permet de mettre à jour les données d'un processus :

- ```
  var databaseId = IdFormat.toString(Prefix.DATABASE, process.getDbId());
  ```

  Ce code source récupère l'identifiant de la base de données au format Gargantua (40 caractères). On peut en effet récupérer l'ID numérique de la base de donnée du processus actuel grâce à la méthode `getDbId()`. Mais ce n'est pas cet ID numérique qui est attendu par la plupart des méthodes
  ```

du framework de l'API GARGANTUA. Il est donc nécessaire d'utiliser `IdFormat.toString()` pour le convertir en identifiant GARGANTUA standard sur 40 caractères. Le paramètre `Prefix.DATABASE` précise que l'identifiant souhaité est un identifiant de base de données.

- ```
  var p1AttrId = Design.idOfAttr("p1", databaseId);
  ```

`Design.idOfAttr` donne l'identifiant GARGANTUA d'un attribut en fonction de son nom et de l'identifiant GARGANTUA de la base de données GARGANTUA dans laquelle il se trouve.

- ```
  process.setAttributeValue(p1AttrId, numero);
  ```

A ce stade, la variable `process` contient une copie des informations du processus (et non le processus original). La méthode `setAttributeValue` met donc à jour la valeur correspondant à l'identifiant d'attribut contenu dans `p1AttrId` de cette copie en mémoire.

La variable a besoin de deux paramètres : l'identifiant de l'attribut à modifier et la nouvelle valeur.

- ```
  Flow.updateProcess(process, new java.lang.Long(Flag.ATTRIBUTES));
  ```

`Flow.updateProcess` met à jour les données du processus GARGANTUA réel, à partir de la copie du processus stockée en mémoire (premier paramètre). Le second paramètre sert à choisir ce qu'il faut synchroniser.

> ***N'oubliez pas, quand vous travaillez avec l'API GARGANTUA, qu'il existe deux types d'identifiants pour les mêmes objets : les identifiants numériques et les identifiants GARGANTUA, qui sont composés de 40 caractères alphanumériques. Le passage de l'un à l'autre se fait grâce à « IdFormat ».***

UTILISER DES BIBLIOTHÈQUES JAVA

Utiliser une bibliothèque Java permet d'étendre les possibilités des scripts ou simplement d'écrire moins de code. C'est particulièrement pertinent lorsque GARGANTUA doit communiquer avec une application métier externe qui propose une API publique en Java.

INSTALLER UNE BIBLIOTHÈQUE JAVA

Pour installer une bibliothèque Java, procédez comme suit :

1. Accédez au dossier **apache-tomcat\lib** du dossier d'installation de votre serveur GARGANTUA (par défaut, **C:\siatel\Gargantua7server**).
2. Collez le fichier .jar correspondant à la bibliothèque Java.
3. Redémarrez le service **Gargantua 7 Server**.

La bibliothèque Java est maintenant disponible dans les scripts de processus.
  ```

Exercice : Se connecter à un système externe avec une API Java

Dans cet exercice, vous appliquerez tout ce que vous avez appris jusqu'ici dans un cas concret fréquemment utile : la connexion à une base de données à partir d'un script.

MySQL est un SGBDR disponible sous license GPL particulièrement utilisé pour réaliser des sites internet. Pour installer MySQL sous Windows, vous pouvez utiliser un package comme EasyPHP ou WAMP.

MySQL sera le système externe avec lequel nous allons communiquer. Une API Java est fournie par son connecteur JDBC.

Consigne

Créez un processus permettant d'obtenir le résultat suivant :

1. Un agent d'accueil renseigne le nom d'un demandeur et valide le formulaire.
2. La table « demandeur » est mise à jour par script.
3. Le formulaire est retourné à l'agent d'accueil avec l'ID généré par MySQL.

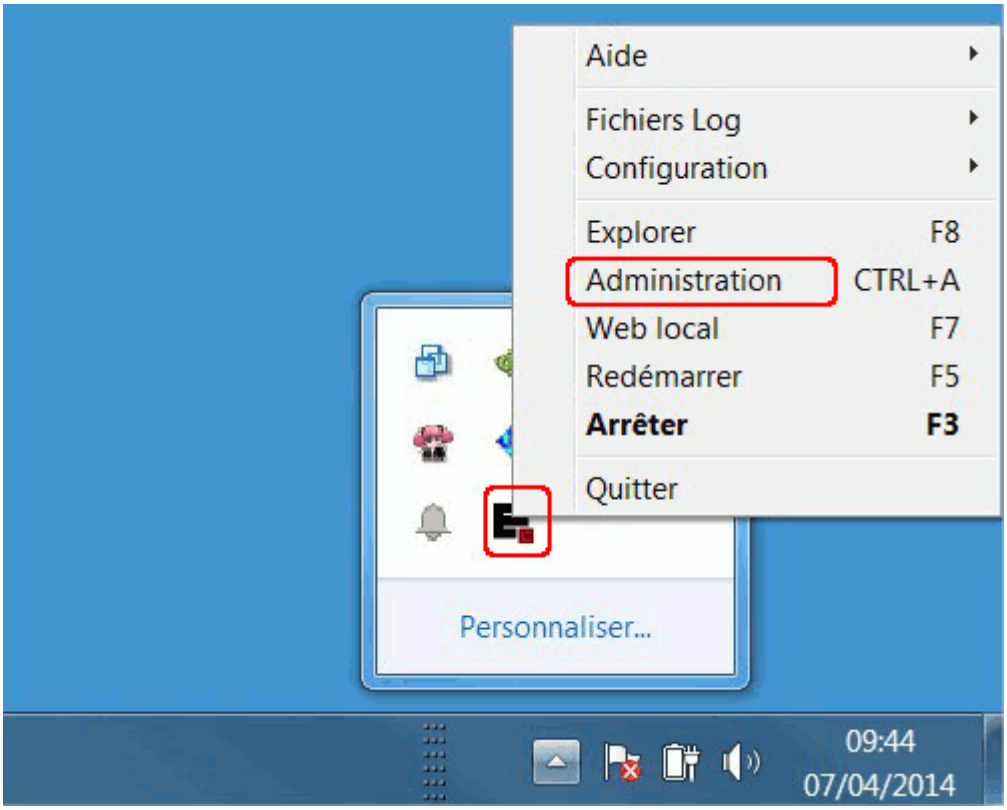
Préparation

Pour effectuer cet exercice, les étapes de préparations suivantes sont nécessaires :

1. Téléchargez et installez EasyPHP.

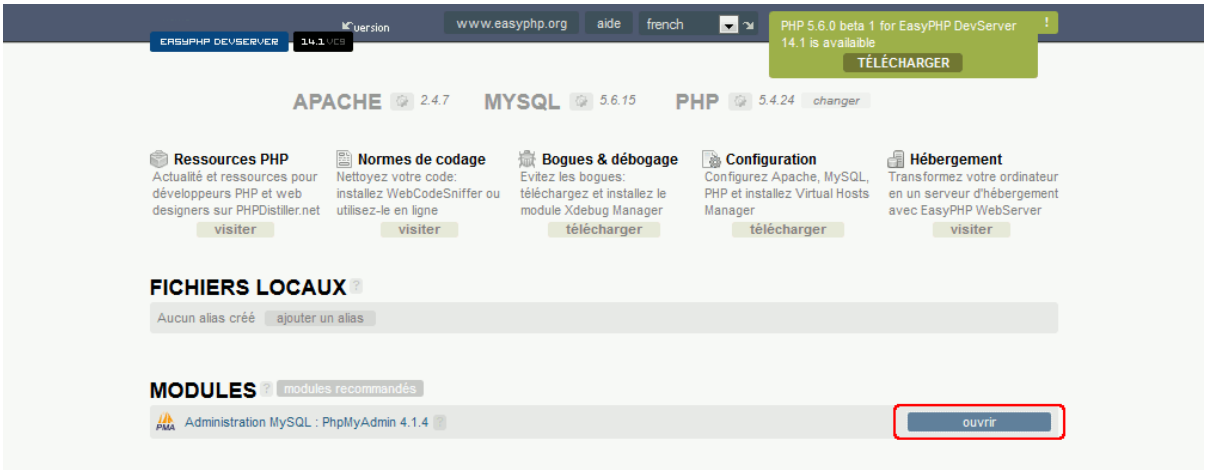
   Un utilisateur MySQL par défaut est créé automatiquement avec tous les droits. Son nom est « root » et son mot de passe une chaîne de caractères vide « ».

2. Faites un clic droit sur l'icône EasyPHP disponible dans la zone de notification (partie droite de la barre des tâches), et sélectionnez **Administration**:



L'écran d'administration d'EasyPHP s'ouvre dans le navigateur par défaut.

3. Dans la rubrique **MODULES**, cliquez sur le bouton **ouvrir** :



phpMyAdmin, l'outil d'administration de bases de données fourni dans le package EasyPHP s'ouvre dans le navigateur par défaut.

4. Cliquez sur l'onglet **Bases de données**.
5. Sous **Créer une base de données**, saisissez « siatel ».

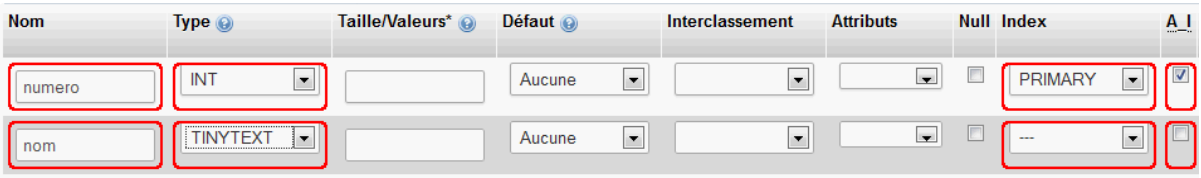
6. Cliquez sur le bouton **Créer** :

Un message indiquant que la base de données « siatel » a bien été créée s'affiche.

7. Dans l'arborescence située à gauche de l'écran, cliquez sur le lien **siatel**.
8. Sous **Nouvelle table**, dans le champ **Nom**, saisissez « demandeurs ».
9. Dans le champ **Nombre de colonnes**, saisissez « 2 ».
10. Cliquez sur le bouton **Exécuter** :

La table « demandeurs » est créée.

11. Pour terminer de définir la table, saisissez les caractéristiques des deux colonnes comme ci-dessous :



12. Cliquez sur le bouton **Sauvegarder**.

Le système externe est maintenant prêt : l'objectif de l'exercice est d'insérer des données dans la table « demandeurs » et de récupérer les numéros créés automatiquement par MySQL.

13. Téléchargez le fichier **.jar** du connecteur MySQL (mysql-connector-java).

AIDE : EXEMPLE DE CONNEXION PAR JDBC À MYSQL EN JAVA

Les informations ci-dessous ont pour but de vous économiser, si vous le souhaitez, la recherche de la syntaxe de connexion à MySQL par JDBC.

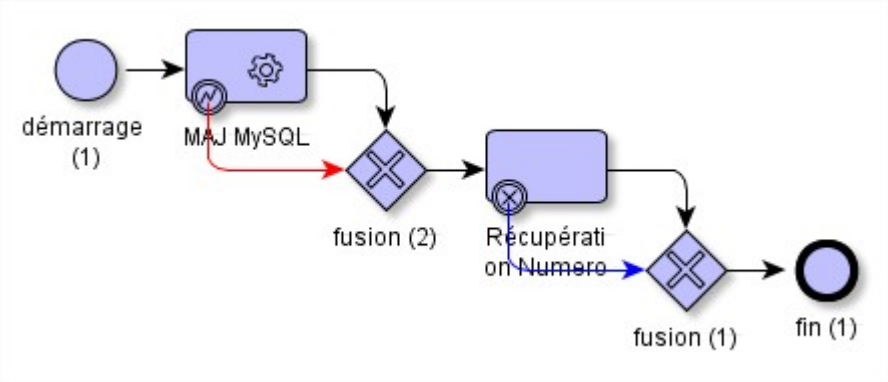
Le code source Java suivant insère un nom dans la table et récupère l'identifiant généré :

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
                                    ...
final String url = "jdbc:mysql://localhost/siatel";
final String user = "root";
final String psswd = "";

Class.forName("com.mysql.jdbc.Driver");

Connection connection = DriverManager.getConnection(
        url,
        user,
        psswd
);

final Statement statement = connection.createStatement();
statement.executeUpdate(
        "INSERT INTO siatel.demandeurs (`nom`) VALUES ('Patrick')"
);

ResultSet generatedKeys = statement.getGeneratedKeys();
if (generatedKeys.next()) {
    System.out.println(generatedKeys.getLong(1));
}

statement.close();
connection.close();
```

CORRECTION

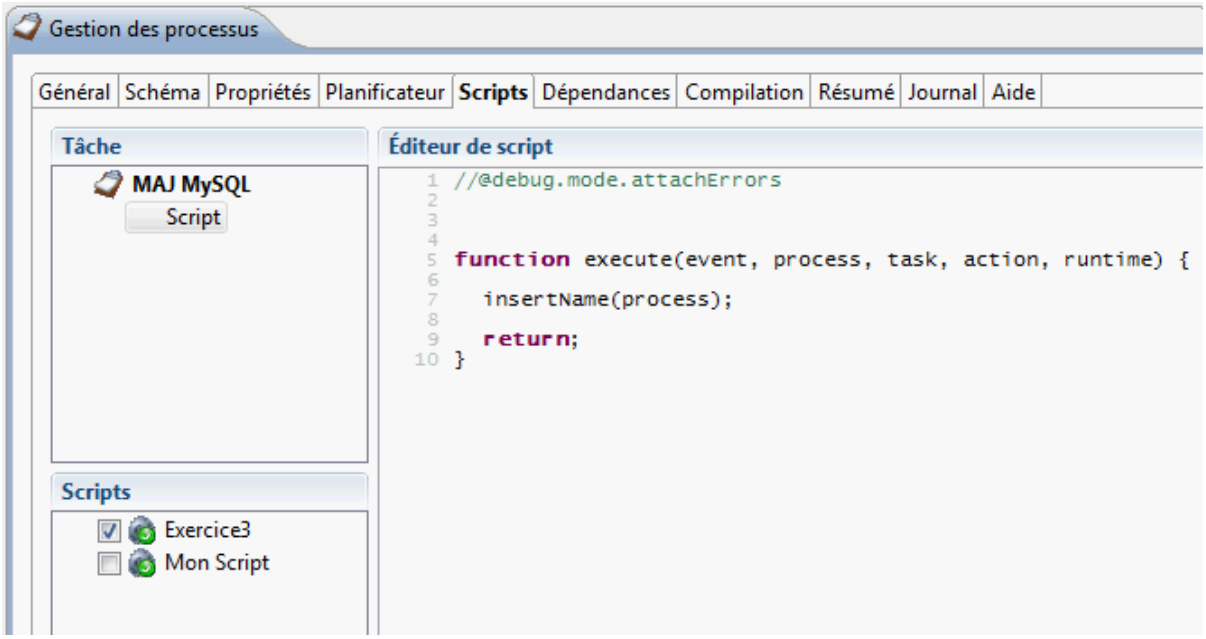
La procédure suivante permet de résoudre l'exercice :

1. Créez deux attributs :
    - « Numéro MySQL », de type nombre entier ;
    - « Nom & prénom » de type alphanumérique.
2. Créez un formulaire, nommé par exemple « Exercice 3 », contenant ces deux attributs.
3. Mettez au point le processus suivant :



- Au démarrage, l'agent d'accueil renseigne le nom et le prénom.
- La tâche automatique « MAJ MySQL » contiendra le script.
- Ses deux chemins sortants sont fusionnés.
- Le chemin résultant redirige vers la tâche utilisateur « Récupération Numéro » qui permet à l'agent de visualiser le numéro généré par MySQL dans le formulaire. Pendant le développement du script, cette tâche permet également de voir les exceptions.
- Quelle que soit l'issue de cette étape (annulation ou validation), le workflow s'arrête (fusion et fin).

4. Accédez au dossier **apache-tomcat\lib** du dossier d'installation de votre serveur GARGANTUA (par défaut, **C:\siatel\Gargantua7server**).
5. Collez dans ce dossier le fichier du connecteur MySQL, **mysql-connector-java-*.*.*-bin.jar**, téléchargé lors de la préparation de l'exercice.
6. Redémarrez le service **Gargantua 7 Server**.
7. Ajoutez un script, nommé par exemple « Exercice 3 », de type **Workflow**, au niveau de la base GARGANTUA et insérez-le dans le script de la tâche automatique « MAJ MySQL » :



La méthode « `insertName` » définie dans le script « Exercice3 » de la base GARGANTUA est appelée ici.

8. En vous aidant du script Java fourni précedemment, créez le script suivant dans la branche **Scripts** de la base GARGANTUA :

```
var url = "jdbc:mysql://localhost/siatel";
var user = "root";
var psswd = "";

function insertName(process) {

    var dbId = IdFormat.toString(Prefix.DATABASE, process.getDbId());

    var nom = process.getAttributeValue(Design.idOfAttr("Nom & prénom", dbId));


    java.lang.Class.forName("com.mysql.jdbc.Driver");

    var connection = java.sql.DriverManager.getConnection(
            url,
            user,
            psswd
            );

    var statement = connection.createStatement();
    statement.executeUpdate("INSERT INTO siatel.demandeurs (`nom`) VALUES ('"
            + nom + "')");

    var generatedKeys = statement.getGeneratedKeys();
    if (generatedKeys.next()) {
        process.setAttributeValue(Design.idOfAttr("Numéro mysql", dbId),
                new java.lang.Integer(generatedKeys.getLong(1)));
        Flow.updateProcess(process, new java.lang.Long(Flag.PROCESS_ALL));
    }

    statement.close();
    connection.close();

}
```

Ce script accède aux attributs du formulaire de processus de la même façon que dans l'exemple de la rubrique Analyse de cas.

Scripts pour les modules complémentaires

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.

Type de numéro de téléphone	Valeur	Description
`TELEPHONE_TYPE_PHONE_2`	4	Représente un second numéro de téléphone fixe. 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