Définitions et services

API

Table des matières

Introduction

Java API Reference

Java API Examples

This chapter demonstrates how to use the `SiatelClient` object. Along this chapter we will create a helper object named `Toolbox` that will encapsulate all functionality needed for communicating with a Gargantua server and accomplish basic tasks.

The Toolbox object

```
package utils.connecteursJava

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import utils.common.ConnectionInfos;
import utils.common.Utils;
import utils.connecteursJava.Toolbox;

import com.siatel.error.DeniedSecurityException;
import com.siatel.error.NotFoundPersistenceException;
import com.siatel.error.SiatelException;
import com.siatel.framework.web.AuthContext;
import com.siatel.proxy.http.SiatelClient;
import com.siatel.defs.Identified;
import com.siatel.domain.*;
import com.siatel.dto.DTO;
import com.siatel.dto.DTOFactory;

public final class Toolbox {
    private static volatile Toolbox instance = null;
    private SiatelClient client = null;

    // keep a connections cache
    private Map<String, ConnectionInfos>
        connectionsMap = Collections.synchronizedMap(
            new HashMap<String, ConnectionInfos>());

    private Toolbox(String urlServer) throws Exception{
        try {
            this.client = new SiatelClient(urlServer);
        } catch (Exception e) {
```

```
            StringBuffer sb = new StringBuffer();
            sb.append("<ERROR> The SIATEL client connector is not initialized.")
            .append("[url : ").append(urlServer).append(" ] ")
            .append(e.getMessage());
            Utils.log(sb.toString());
            throw new Exception(sb.toString());
        }
    }

    public static void initToolbox(String urlServer)throws Exception{
        Utils.log("------ Gargantua Java connector demo ------");
        if(Toolbox.instance == null){
            synchronized(Toolbox.class) {
                if (Toolbox.instance == null) {
                    Toolbox.instance = new Toolbox(urlServer);
                }
            }
        }
    }

    // ---------------------------------------------------------
    // code from the following sections will be inserted here
    // ---------------------------------------------------------
}
```

Managing the connection cache

Add the following code to the `Toolbox` object to handle the connection cache.

```
private void addToCache(String conn, String url, String userName){
    if(!instance.connectionsMap.containsKey(conn)){
        ConnectionInfos infos = new ConnectionInfos(url, userName);
        instance.connectionsMap.put(conn, infos);
    }
}

private ConnectionInfos getFromCache(String conn) throws Exception{
    if(instance.connectionsMap.containsKey(conn)){
        return instance.connectionsMap.get(conn);
    }else{
        throw new Exception("The connection cache does not " +
            "hold the specified connection (" + conn + ")");
    }
}
```

```
private String getUserNameFromCache(){
    String conn = instance.client.getConn();
    String userName = "";
    ConnectionInfos infos;
    try {
        infos = instance.getFromCache(conn);
        userName = infos.getUserName();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return userName;
}

private void removeFromCache(String conn){
    if(instance.connectionsMap.containsKey(conn)){
        instance.connectionsMap.remove(conn);
    }
}
```

Getting server information

The `aboutServer` method retrieves server information. The `Toolbox` objects needs to be valid and the connection already opened to the server.

```
// --- Informations about server
public static void aboutServer() throws Exception {
    try {
        StringBuffer sb = new StringBuffer();
        DTO dtoIn = DTOFactory.newDTO();

        dtoIn.put(Constants.SERVER_VERSION_MAJOR, "");
        dtoIn.put(Constants.SERVER_VERSION_MINOR, "");
        dtoIn.put(Constants.SERVER_VERSION_REVISION, "");

        DTO dtoOut = instance.client.getServerInfo(dtoIn);

        // server url
        sb.append("\n").append("Server URL = ").append(instance.client.getServerUrl());

        // server version
        sb
            .append("\n").append(" -> Major version = ")
            .append(dtoOut.get(Constants.SERVER_VERSION_MAJOR))
            .append("\n").append(" -> Minor version = ")
```

```
            .append(dtoOut.get(Constants.SERVER_VERSION_MINOR))
            .append("\n").append(" -> Revision = ")
            .append(dtoOut.get(Constants.SERVER_VERSION_REVISION));
        Utils.log(sb.toString());
    }
    catch (SiatelException e) {
        throw new Exception("Unable to retrieve server " +
        "information.\n"+e.getMessage());
    }
}
```

Opening a user connection to the server

The code below demonstrates how to login to the Gargantua server using user credentials.

```
// --- Open a user connection to the server
public static void doLogin(String username, String password) throws Exception {
    String conn = "";
    String urlServer = instance.client.getServerUrl();
    StringBuffer sb = new StringBuffer();

    try {
        instance.client.login(username, password, AuthContext.RUNTIME);
        conn = instance.client.getConn();
        instance.addToCash(conn, urlServer, username);

        sb.append("Login / ")
            .append(conn)
            .append(" / server [")
            .append(urlServer)
            .append("] / user [")
            .append(username).append("]");
        Utils.log(sb.toString());
    }
    catch (Exception e) {
        sb.append("<ERROR> Login failed ")
            .append(" / server [")
            .append(urlServer)
            .append("] / user [")
            .append(username).append("]");
        Utils.log(sb.toString());
        sb.append(" - e.getMessage : ").append(e.getMessage());
        throw new Exception(sb.toString());
    }
```

```
}
```

Closing a user connection

The code below demonstrates how to close a connection to the server.

```
// --- Close a user connection to the server
public static void doLogout() throws Exception {
    String urlserver =  instance.client.getServerUrl();
    String username = "";
    String conn = instance.client.getConn();
    StringBuffer sb = new StringBuffer();

    try {
        instance.client.logout();

        ConnectionInfos infos = instance.getFromCache(conn);
        username = infos.getUserName();
        sb.append("Logout / ").append(conn).append(" / server
[").append(urlserver).append("] / user [").append(username).append("]");
        Utils.log(sb.toString());

        instance.removeFromCash(conn);

    }
    catch (SiatelException e) {
        sb.append("<ERROR> Logout failed / ").append(conn).append(" / server
[").append(urlserver).append("] / user [").append(username).append("]");
        Utils.log(sb.toString());
        sb.append(" - e.getMessage : ").append(e.getMessage());
        throw new Exception(sb.toString());
    }
}
```

Reading user information by label

The code below demonstrates how to read a `User` object from the server given it's name. In general the same technique applies to all objects that be requested by label.

```
// read a user identified by it's name
public static User doReadUser(String label) throws Exception {
```

```
    User aUser = (User) SiatelFactory.newObject(Prefix.USER);
    aUser.setLabel(label);

    try {
        instance.client.readUser(aUser, Flag.USER_ALL);
        Utils.log(aUser.toString());
        return aUser;
    }
    catch(NotFoundPersistenceException nfp) {
        StringBuffer sb = new StringBuffer();
        sb.append("The user identified by label [").append(label).append("] does not
exist.");
        Utils.log(sb.toString());
        throw new Exception(sb.toString());
    }
    catch (SiatelException e) {
        StringBuffer sb = new StringBuffer();
        sb.append("<ERROR> Read user [").append(label).append("] command
failed.").append(" - e.getMessage() : ").append(e.getMessage());
        Utils.log(sb.toString());
        throw new Exception(sb.toString());
    }
}
```

Update a user

The code below demonstrates how to update a `User` object on the server. In general the same technique applies to all objects that can be updated.

```
// update a user object
public static void doUpdateUser(User aUser) throws Exception {
    try {
        instance.client.updateUser(aUser, Flag.USER_UPDATE_ALL, true);
        Utils.log(aUser.toString());
    }
    catch(NotFoundPersistenceException nfp) {
        StringBuffer sb = new StringBuffer();
        sb.append(The user [").append(aUser.getLabel()).append("] does not exist.");
        Utils.log(sb.toString());
        throw new Exception(sb.toString());
    }
    catch (SiatelException e) {
        StringBuffer sb = new StringBuffer();
        sb.append("<ERROR> Update user [").append(aUser.getLabel()).append("] command
failed.").append(" - e.getMessage() : ").append(e.getMessage());
```

```
        Utils.log(sb.toString());
        throw new Exception(sb.toString());
    }
}
```

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("DFAT0007000006600000000000000000000000", "#ff0000");
```

After executing the previous line, the following line

```
window.alert(getElementColor("DFAT0007000006600000000000000000000000"));
```

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("DFAT0007000006600000000000000000000000", "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("DFAT0007000000660000000000000000000000000", 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('DFAT000700000001000000000000000000000000', 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('DFAT0007000000660000000000000000000000000'));
```

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('DFAT00070000001000000000000000000000000');
```

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('DFAT0007000000010000000000000000000000000');
```

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)

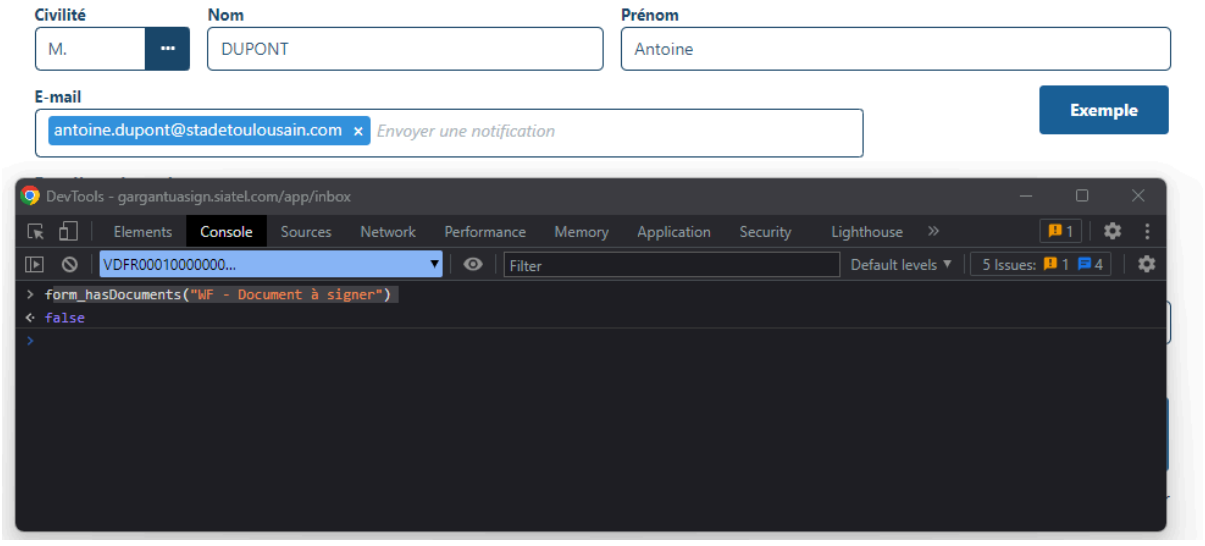
Valeur retournée

(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

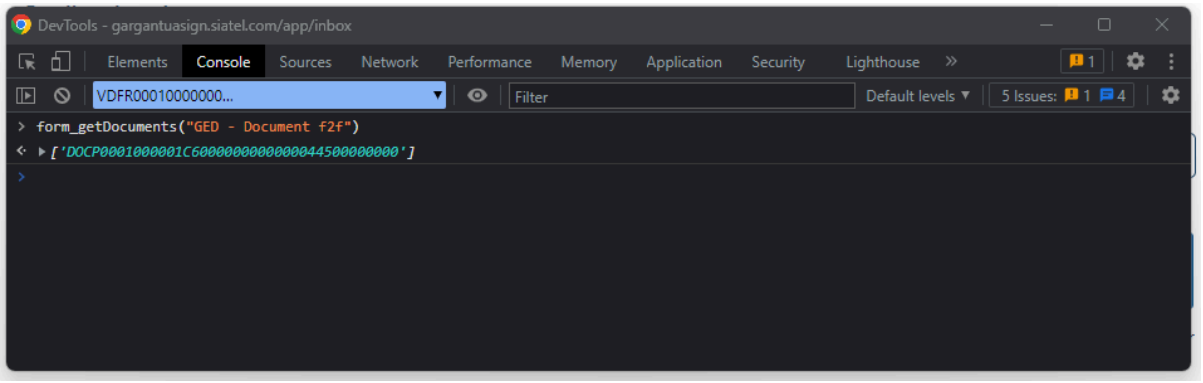
Nom du formulaire (string)

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.

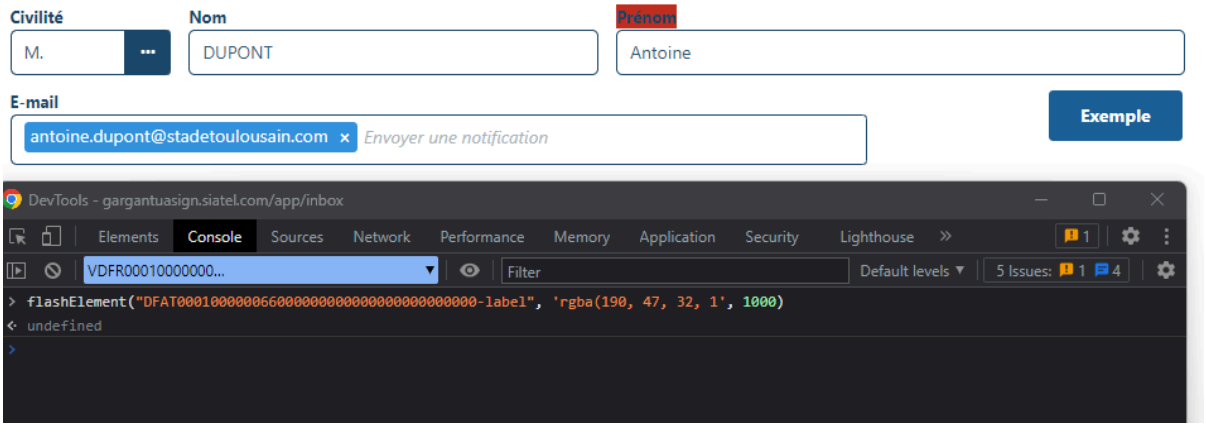
count

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

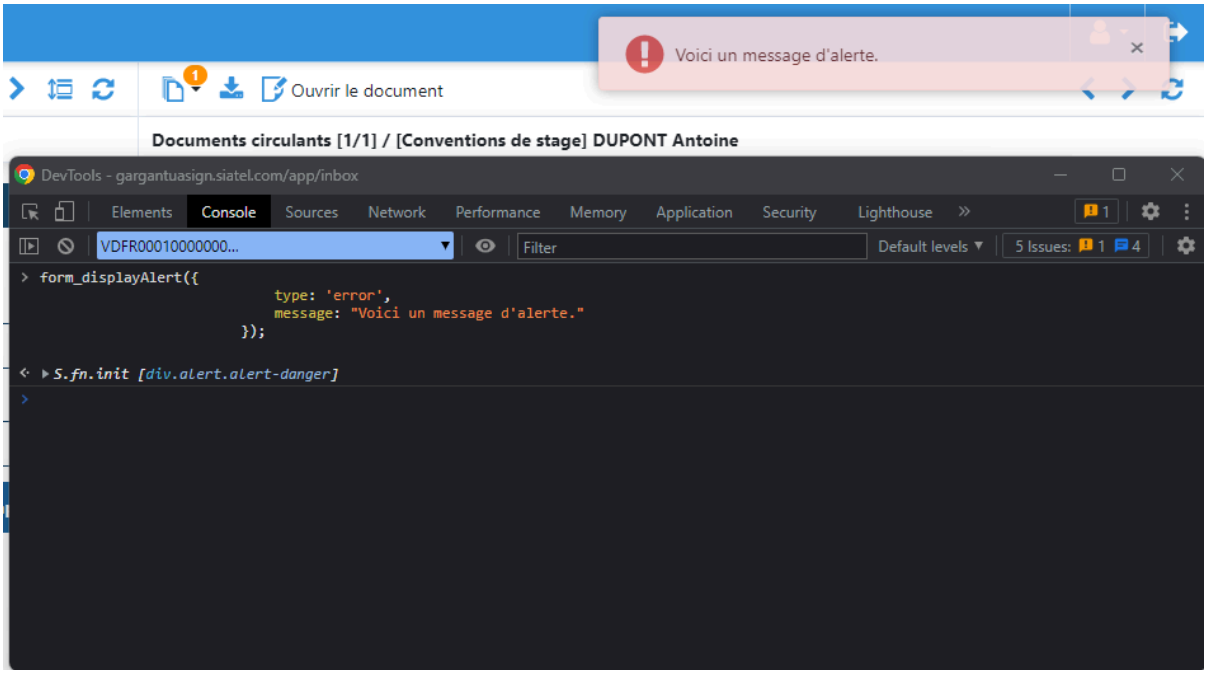
data

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

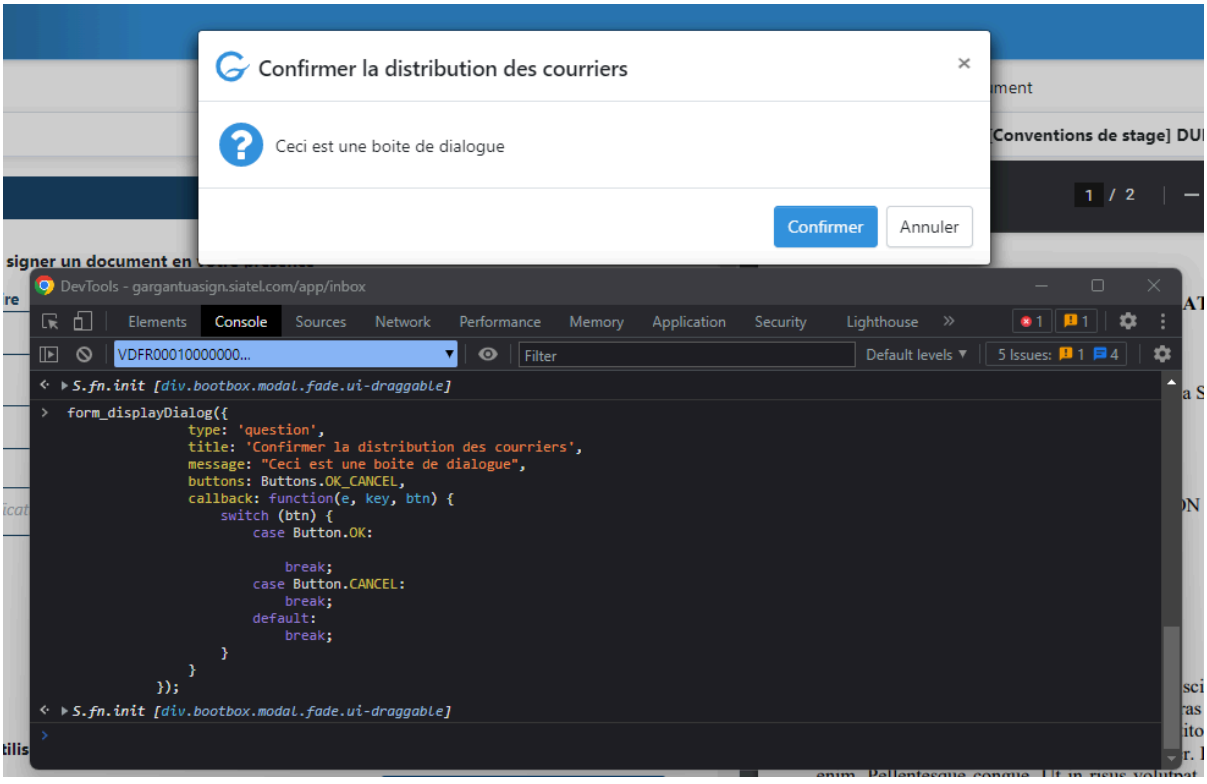
data

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=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.

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> FLDR000100000005000000000000000000000000</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;
```