JavaScript Form Generator

Disclaimer

This SOFTWARE PRODUCT is provided by El Condor "as is" and "with all faults." El Condor makes no representations or warranties of any kind concerning the safety, suitability, lack of viruses, inaccuracies, typographical errors, or other harmful components of this SOFTWARE PRODUCT. There are inherent dangers in the use of any software, and you are solely responsible for determining whether this SOFTWARE PRODUCT is compatible with your equipment and other software installed on your equipment. You are also solely responsible for the protection of your equipment and backup of your data, and El Condor will not be liable for any damages you may suffer in connection with using, modifying, or distributing this SOFTWARE PRODUCT.

You can use this **SOFTWARE PRODUCT freely, if you would you can credit me in program comment:**

El Condor – CONDOR INFORMATIQUE – Turin

Comments, suggestions and criticisms are welcomed: mail to rossati@libero.it

Conventions

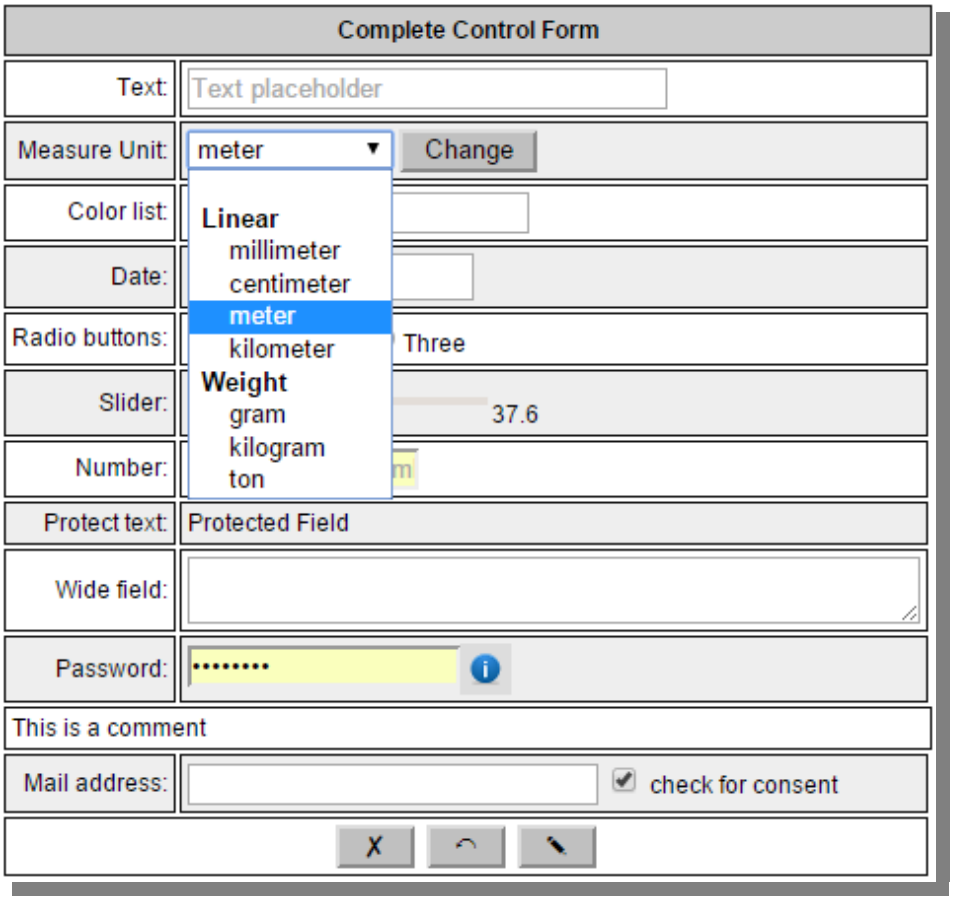
Commands syntax, instructions in programming language and examples are with font `COURIER NEW`. The optional parties of syntactic explanation are contained between `[square parentheses]`, alternatives are separated by `|` and the variable parties are in *`italics`*.

Contents table

1 Form generator

Form generator, briefly *FormGen*, is a JavaScript source which allows build and handle forms; *FormGen* is sufficiently generalized for create a wide set of useful forms from simple message box to complex input forms, based on a list of input controls (some text type, buttons, check boxes, lists, radio buttons...).

The form can be submitted or sent by Ajax technology or managed locally.



1.1 Using the form generator

The form builder is contained in `formgen.js` script, which contains the object function `formGen`.

This function can be invoked for create a new object:

```
fGenObject = new formGen(idContainer[,control_list][,callBackEvent])
```

☞ the function can also be invoked directly but in this case some functionalities aren't disposable:

```
formGen(idContainer[,control_list][,callBackEvent])
```

`idContainer` is a `div` tag (it can also be a `span` or a `td` tag) which will contain the created form.

☞ In the case of calling the function with only the `id` of the container, this must contains the list of controls, which is replaced, obviously, by the form generated (☞ Caveat! If the browser has a translation active the list can be altered with unknown results.):

```
...
<script type='text/javascript' src='js/formgen.js'></script>
...
<body bgcolor=teal onLoad='formGen("form")'>
...
<div id='form'>
```

```
parameter list
</div>
...
```

☞ If the `id` is not present, it is created a `<div>` tag as PopUp; if the `id` is not specified, the `<div>` tag is create with `id fg_PopUp` and `class fg_PopUp` (see Example 6: Internationalization and popup movable).

The second possible parameters is a string containing the list of controls (widgets).

☞ **Experimental!** The third possible parameters is a function called on some form events, see par. 1.5.

1.1.1 Compatibility

Internet Explorer > 9; some controls or properties presuppose HTML5.

1.2 Data description

Every control (or widget) is characterized by a list of attributes, separated by comma, in this order: *`Type`*, *`Field Name`*, *`Field Label`*, *`Length`*, and *`Extra field(s)`*. Controls are separated by semicolons.

In addition to the controls we can have some others information (*Pseudo types*) with different semantics that will be detailed in the paragraphs dedicated to them.

If *`Type`* starts with `//` it is a comment which ☞ must also be terminated by semicolon.

☞the possibly commas, semicolons, equal and & signs in *`labels`* and *`extra fields`*, must be coded respectively by `\x2C`, `\x3B`, `\x3D` and `\x26`. Alternatively, the attributes can be enclosed in `'` or `"` in order to contains the above characters.

1.2.1 Type

The *`Type`* is indifferent to the case.

- **Buttons**:
  - **B** button;
  - **GB** graphic button;
  - **IB** inline graphic button;
  - **R**, **RDB** radio button, a set of Radio buttons;
  - **RV** a set of vertical Radio buttons;
- **CKB** check box;
- **CKL** check box list;
- **Combo box** and **lists**:
  - **CMB** drop down list for select an item (see par. 1.5 how to make a multiple selection);
  - **L** or **LIST** is a drop down list associated to a text, where we can choose an item or insert one not present;
- **I**, **IMG**, **IMAGE** image,
- **Text** fields:
  - **C**, **CS**, **CE**, **COMMENT** comment;
  - **DATE**;
  - **F** input file;
  - **S** seek bar or slider;
  - **P** or **PSW** password field, the data entered are masked;
  - **T** or **TEXT** text field;
  - **N** integer number, **NS** signed number and **NF** floating number;
  - **U**, **UN** not modifiable field i.e. a protected field, **UN** is numeric for right alignment.

- **H** hidden field.

1.2.2 Field Name

Is the name of the control that, when the form is submitted, it is used by the server to access its value; the name is case-sensitive. The ID of the control that can be used to access or to add an event management, has the form: *formNameFieldName* (*formName* is provided by pseudo type `Form`).

☞ If the name is not present it is generated the name `fg_i`, where *i* is a progressive number.

1.2.3 Field Label

Is the label of control or the caption of button[1]; if omitted it is used the *FieldName*.

1.2.4 Length

Is the length of the control in characters or in pixels for sliders, buttons and radio buttons; see the table at right for the defaults length.

The length for buttons, graphic buttons and radio buttons, has no default value.

IB Inline button	150	pixels
N Number	6	
NF floating number	9	
NS Signed Number	7	
S Slider	150	pixels
Texts	20	

1.2.5 Extra

Extra field(s) is used for add information to the control, these will be specified in the relative paragraphs.

1.3 Summary by type

Type		Length	Compatib.	Extra field
Button	**B,GB,IB**	pixels		Possible name of handling function, possible title
Check box	**CKB**			Possible label at right of check box
Check box list	**CKL**			An item list separated by `comma`: `[`*key*`=]`*value*`[, [`*key*`=]`*value*`[,...`
Combo box	**CMB**			An item list separated by `comma`: `[`*key*`=]`*value*`[, [`*key*`=]`*value*`[,...`
File	**F, FILE**	characters		filters
Date	**DATE**		HTML5	
DataList	**LIST, L**		HTML5	An item list separated by `comma`
Hidden	**H**	characters		The value
Radio button	**RDB, R, RV**	pixels		An item list separated by `comma`: `[`*key*`=]`*value*`[, [`*key*`=]`*value*`[,...`
Slider	**SLIDER, S**	pixels	HTML5	Start, end and step value, default is `0 100 1`
Text	**TEXT, T, PSW, P, N, NS, NF**	characters		Possible hint
Unmodifiable,	**U.UN**	characters		The value

[1] In case of graphic buttons it is the name of the image on the server.

1.3.1 Buttons and graphic buttons

Buttons can be used both for take different actions on form both for show user caption instead of default `Ok`, `Reset` or `Cancel`.

The syntax is:

```
B,name,caption,[length],[function|alert:info],[title]
GB,name,imageSource,[length],[function|alert:info],[title]
IB,name,imageSource,[length],[function|alert:info],label
```

The *length* of **B** and **GB** type, if present, is the dimension of the button or image in pixels but there is no default value; the length of **IB** type is the space, in pixels, of the label, its default value is 150.

The *extra* field can contains a name of function which is called instead of the internal function for submit;

The second possible *extra* field of **B** and **GB** type become the argument of `title` tag and it is shown when the mouse is over the button.

The **IB** (inline button) is a graphic button that is shown at left of *label*.

☞ if the button is after a field (by `after` pseudo type) has a function, the content of the form is **not controlled** otherwise it acts like a normal submission button.

☞ The arguments of the function called is the form (the field `fg_Button` contains the button name).

The type **B** button has `fg_Button` class; the type **IB** and **GB** button have `fg_GButton` class.

☞ There is no generated `id` for standard buttons.

1.3.2 Check box

The extra field of **CKB** type can contain a possible description displayed to the right of the check box; check box can appears after or below another control.

☞ The value returned of check box is present only if it is checked and the value returned is `on`.

1.3.3 Check box List

CKL type generates a set of check boxes.

The extra field contains a list of field names separated by `,` (comma) with syntax: `[key=]value[, [key=]value[,...`; the field name of check box is *key* if present, otherwise is *value*; *value* is the description that appears after the check box.

The *Field Name* of the check box list will contain the number of check boxes selected.

Ex. `CKL,cList,Check list,,C=C\x2cC++\x2cC#,JS=JavaScript,PHP,PYTHON,RUBY,RUST;`

1.3.4 Combo boxes and Lists

CMB type is a Combo box (or Drop Down list) that permits to choice a value from a list; the **LIST** (**L**) type accepts an input value or an item selected from the list.

The `extra`*s* fields contain the item*s* (see description in Radio button).

If there is only one radio buttons set or only one combo in the form, the form does not have buttons and it is exited when a list item or one radio button is selected; the form is erased (unless the form is static, see Form 1.4.6).

```
JS    $("result").innerHTML = $("form2").innerHTML;Fgen.build("result");

HTML  <span id='form2' style='visibility:hidden'>
      Form,frm2,,echo.php;
      CMB,Unit,Measure Unit,,Lt=Liters,Kilos,Mc=Cubic Meters,Wh=Watt/hour;
      </span>
```

Example 1: One choice without buttons

It is possible to have a combo with items separated on group (like `optgroup` tag), the group is identified by the syntax `|=`*`groupLabel`*:

```
CMB,Unit,Measure Unit,,
|=Linear,mm=millimeter,cm=centimeter,m=meter,km=kilometer,
|=Weight,g=gram,kg=kilogram,t=ton;
```

☞ After submission the field *`ctrlName`*`_Group` contains the possible group name.

Using a function of callback for event (see par. 1.5) is possible to make a combo able to accept multiple choices, see the snippet below:

```
Form,frmg2,Form Generator 2,echo.php,receiveData;
CMB,Hellas,Greek letters;
Get,Hellas,getSample.php?Type=Hellas;
...
Fgen = new formGen("result",$("get").innerHTML,cBack);
...
function cBack(event,frm) {
  if (event == "Start") {
      $("frmg2Hellas").setAttribute('multiple', true);
      $("frmg2Hellas").name = "Hellas[]";
  }
}
```

Example 2: Make combo with multiple choice

1.3.5 Comment

The label field is the comment shown:

```
Comment|C|CS|CE,[fieldName],some comment
```

`Comment` and `C` are synonym.

The three forms `C`, `CS` and `CE` are generate with the class respectively `fg_Comment`, `fg_Separator` and `fg_Error` in order to use different styles.

The `CE` type is also generated by *formGen* when an error is encountered in the control list (for example a field type unknown).

☞ Comments are in a `div` tag, for change by program the contents can be used the `innerHTML` method on the ID *`formNamefieldName`*.

1.3.6 Date

HTML 5 Supported.

☞ The possibly default must be in the form *`yyyy-mm-dd`*; it is also accepted `today`.

1.3.7 File

The *extra* field can contains a file filter(s), if many, they are separated by comma:

```
F,mediaFile,Media File,60,audio/*,video/*,image/*
F,psFile,PDF and PS files,50,.pdf,.ps
```

☞ For control the maximum length of a file upload on PHP script, one can use a hidden field with name `MAX_FILE_SIZE` that must precede the file input field (see below).

1.3.8 Hidden field

The *extra* field contains the value:

```
H,MAX_FILE_SIZE,,,5000;
File,Attachment,Attachment file,30,.gif,.jpg,.png;
```

The value can also be set by `default` pseudo type.

1.3.9 Image

`[I|IMG|Image],`*`name`*`,`*`label`*`,[|`*`height`*`],`*`imageFile`*`[,`*`title`*`]`

1.3.10 Radio buttons

The *extra*(*s*) fields contains the labels and value of each radio button. To obtain a key instead of the label, the item must have the form: *key*=value.

```
Rdb,Status,,45,M=Married,S=Single,W=Widow
```

Every item can be enclosed in ' or " if it contains comma or semicolon, it is also that only value is enclosed, for example:

```
Form,frm,Try Sand Box,echo.php,receive;
RV,vRdb1,vRdb 1,,North,South,West,East,NW='North, West','SE=Sud, East';
```

Example 3: Vertical radio button

The length field, if present, determine the distance in pixel from the radio buttons items.

The **RV** type tells to place the radio buttons vertically.

If no radio Buttons are checked the value exists and is the empty string. It is possible to have more than one set of radio buttons in the form.

If there is only one radio buttons set in the form, this does not have buttons and it is exited when a list item is selected; the form is erased (unless the form is static, see Form 1.4.6).

1.3.11 Slider

The *length* is the length on pixel of the slider.

The *extra*(s) fields can contains the start, end and step values in the form start,end,step, e.g. -5,5,0.5; if it is omitted, the values assumed are 0 100 and 1. The result can have decimals depending on the difference from start and end value, see table at right.

☞ The slider has always a value.

start - end	n. decimals
> 99	0
< 100 and > 10	1
< 10 and > 1	2
< 1 and > 0.1	3
...	...

1.3.12 Text fields

TEXT and **PSW** are synonym of **T** and **P** respectively; numeric fields are **N** (unsigned integer), **NS** (signed integer) and **NF** (signed number with possibly decimals).

☞ Numeric texts have the inputmode parameters that it isn't supported on Firefox browser.

☞ In local management all fields are of type string, use toInt or toFloat method if you want perform calculations.

The **U** and **UN** types are a not modifiable texts, **H** type is a hidden text; ☞ their values can be set in the *extra* field or set by pseudo type default(s).

If the length exceeds 50 characters is generated a text area.

The *extra* field, if present, contains a text hint (HTML5 placeholder property); ☞ if the length exceed the field length the *extra* field becomes a title.

☞ The contents of the possibly second *extra* field is shown after the input field.

1.4 Pseudo types

1.4.1 After and Below

These pseudo types are useful for insert some controls (buttons, check boxes, images, combo box and text fields) after or below some others controls.

By defaults the buttons are inserted at the bottom of the form, these pseudo types tell instead where the aforementioned types must be placed; the syntax is:

```
After|Below,ctrlName,ctrlToPlace1[,ctrlToPlace2[,…]]
```

☞ These Pseudo type must appears after the fields involved.

```
JS    $("Agree").addEventListener("click",function() {$('Start').disabled = !
      this.checked;},true);

      CKB,Agree, Consent cookies?,10,I agree;
      B,Start;
HTML  B,fg_Cancel,&#x2718;;
      Defaults,Start=off;
      After,Agree,Start
```

Example 4: Enable button after control

1.4.2 Controls (Check and Required)

There are two pseudo types for controls fields: `Control` or `Check` and `Required` or `Req`.

`Control` or `Check` is used to perform controls on the data. The structure for this command are:

```
[Check|Control],ctrlName,control[=value],error[,control[=value],error…]
```

If a field has more controls they are in and.

```
[Check|Control],*,function=functionName,error
```

This second form is to perform checks not related to a single field.

Examples:

```
Check,Number,min=-200,Not allowed lesser -200,max=200,Not allowed greater 200;
Control,Mail,mail,Required;
Check,psw,pattern=(?\x3D.*\d)(?\x3D.*[A-Z])(?\x3D.*[a-z]).{6\x2C12},Almost one
Uppercase Lowercase and digit;
```

The possibly `control`(s) are:

Type	Value	Note
`required`	none	by pattern `/^\s*$/`
`min`	numeric value	Minimum value allowed
`max`	numeric value	Maximum value allowed
`Mail (*)`	none	Controlled by pattern `/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/`
`Pattern (*)`	a regular expression	The possibly commas, semicolons equal sign and & must be codified respectively by `\x2C`, `\x3B`, `\x3D` and `\x26`.
`function`	a JavaScript function	The parameters of function are: the *`form`*, the *`fieldName`* and the *`value`* (see example below), the function must returns `true` or `false`.

`(*)` The control is not executed if the field is empty.

```
...
N,Qty,Stock Quantity;      function controlWhithdraw(frm,field,value) {// check Quantity
N,wQty,Quantity withdrawn; withdrawn
Check,wQty,function=contro   if (value > parseFloat(frm["Qty"].value)) return false;
lWhithdraw,excess            return true;
quantity;                  }
...
```

Example 5: Function that controls fields

The syntax of `Required` is: `Required,fieldName1[,fieldName2[,...]]`

1.4.3 Defaults

The type `Default[s]` is used for populates the form; the syntax is:

`Default[s],`*`ctrlName=value`*`[,`*`ctrlName=value`*`[,...]]`

- The possible value for check boxes is `on` or `checked`; for list, combo box and radio button the *`value`* can be both the key both the value shown; for disable Buttons is accepted the value `off`.
- If the value is constant (like `on`, `off` etc.) it is case insensitive.
- For Date type the format must be *`yyyy-mm-dd`*; it is also accepted `today`.

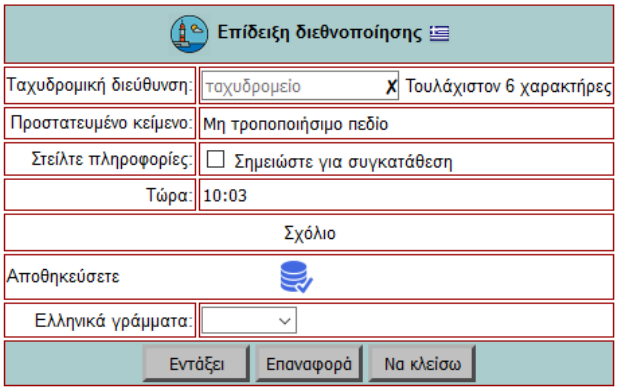
When the `Reset` button is pushed the form is restored with the default values.

1.4.4 Dictionary

`Dict[ionary],`*`dictionaryObject`*

The `Dict[ionary]` pseudo type is intended for form internationalization. *`dictionaryObject`* is a set of `key value` items where the `key` is the word or phrase contained in the control list and the `value` is the translation.
The translation is applied to:

- button's caption,
- comments
- extra field of check box and texts.
- form title,
- hints,
- labels,
- radio buttons and combo box exposed values.



HTML

```
...
<img src="images/its.png" class="imgbtn" onclick="changeLang('IT')" title="IT">
<img src="images/frs.png" class="imgbtn" onclick="changeLang('FR')" title="FR">
<img src="images/ess.png" class="imgbtn" onclick="changeLang('ES')" title="ES">
...
```

JavaScript

```
function changeLang(Lang) {
   dict = {}
   for (w in dictionary) {
     if (typeof dictionary[w][Lang] != "undefined")
           dict[w] = dictionary[w][Lang];
   }
   var form = "Form,ft,"+"images/faro.ico:Demo internationalization:"
     +changeLang.Flags[Lang]
     +",echo.php,receiveData;"
     +formTrans+"Dict,dict;"
     if($("fg_PopUp")) $("fg_PopUp").remove();
     Fgen = new formGen("",form)
     var link = $("fg_PopUp")
     link.style.top = 0.5 * (window.innerHeight - link.offsetHeight);
     link.style.left = 0.5 * (window.innerWidth - link.offsetWidth);
     $("ftfg_Title").classList.add("fg_Movable")
//   $("ftfg_Title").className += " fg_Movable";      // older browsers?
     $("ftfg_Title").addEventListener("mousedown", dragStart.bind(null, event,
"fg_PopUp"))
}
```

```
changeLang["Flags"] = {IT:"images/its.png",FR:"images/frs.png",
              ES:"images/ess.png",EL:"images/els.png",EN:"images/uss.png"}
```

Control list

```
var formTrans = ""
+ "\nT,Mail,Mail address,,Mail,Minimum 6 characters;"
+ "\nU,Protect,Protected text,,Not modifiable Field;"
+ "\nCKB,CheckBox,Send info,10,Check for consent;"
+ "\nU,Time;"
+ "\nCS,Comment;"
+ "\nIB,Save,images/update.png,200,alert:Alert in english,Save;"
+ "\nGET,Time,getSample.php?Type=Time;"
+ "\nCMB,Hellas,Greek letters,,Alfa,Beta,Gamma,Delta;"
```

Dictionary

```
var dictionary = {"Mail address":{IT: "Indirizzo di posta",
            FR:"Adresse e-mail",EL:"Ταχυδρομική διεύθυνση"},
            Mail:{IT:"Posta elettronica",FR:"Courrier",
            ES:"Correo",EL:"ταχυδρομείο"},
...
            Reset:{IT:"Ripristina",FR:"Réinitialiser",ES:"Reiniciar",
                   EL:"Επαναφορά"},
            Cancel:{IT:"Chiudere",FR:"Fermer",ES:"Cerrar",EL:"Να κλείσω"},
            Ok:{ES:"Okay",FR:"Bien",EL:"Εντάξει"}
            }
```

Example 6: Internationalization and popup movable

1.4.5 Event

This pseudo type is used to attach an event handler to a field, the syntax is:

`Event,`*`ctrlName`*`,`*`eventType`*`|enter,`*`function`*`|submit`

Event type `enter` can be associated to a text field, normally for manage the `enter` key (because ☞ it has been disabled by *FormGen)*; this event is an effect a `keypress` event.

`eventTypes` are the events accepted by `addEventListener` function.

`submit` invokes the form submission; ☞ this is meaningful for some events, for example when a combo list changes.

☞ If an event is associated to a set of radio buttons, each of them will reacts.

```
Form,fe,Submit on Enter or Select,echo.php,receiveData;
T,Name;
N,Qty,Quantity;
Event,Name,Enter,Submit;
```

```
...
      } else if (type == "event") {
            var eventFrm = "CMB,Images;"
                  + "Rdb,imageType,Image type,,.gif,.jpg,.png,;"
                  + "B,fg_Cancel,&#x2718;"
                  + "Defaults,imageType=.png;"
                  + "Get,Images,getSample.php?Type=Images&imageType=.png;"
                  + "Event,imageType,click,getImageList;"
                  + "Event,Images,change,showImage;";
            $("fg_list").value = eventFrm;
            Fgen = new formGen("result",eventFrm);
            return;
      }
```

```
...
function getImageList() {
       var url = "getSample.php?Type=Images&imageType=" + event.target.value;
       formGen.prototype.ajax(url,"",function(c){Fgen.createOptions("Images",c)})
}
function showImage() {
  $("insertedData").innerHTML = "<img src='images/" + event.target.value + "'>";
}
```

Example 7: Event, Get and createOptions function

1.4.6 Form

The type `Form` is used to tell how the form is treated when it is submitted; the syntax is:

`Form,`*`name`*`,[`*`caption`*`],[`*`uri`*`],[`*`function`*`],[reset|static],[`*`target`*`]`

`name` is the form ID, if it is omitted the ID is `fg_Form`.

`caption` is displayed, if present, above the controls; *`caption`* can have the form:

$subCaption_1$`[:`$subCaption_2$`[:...]]`

`subCaption` con be an image (`.gif`, `.png`, `.jpg`, `.ico` or `.jpeg`):

`Form,ft,images/els.png:Demo internationalization,echo.php,receiveData;`

`uri` is the server script which receive the form (via submit or `ajax`), if it is not present the form is not submitted and *`function`*, if present, is called with the form as argument.
`reset` restore the form after submission (like the `Reset` button),
`static` a `Ok` and `Cancel` button aren't generated.

`target` can be `newwindow|_blank|_self|_parent|_top|`*`frameName`* where `newwindow` is a synonym of `_blank`.

☞ The form is erased by `Cancel` button; if the form is `static` there is no `Cancel` button. If the form has the `reset` parameter it is cleared.

Before the submission the data are controlled as indicated in the pseudo type `Check` (if it exists), in case of error(s) the form is not submitted and the field(s) in error are bordered in red; it is also generated an alert.

Submission type	uri	function	Note
Form submission	required	empty	a new page is generated.
Ajax	required	required	The *`function`* receives the answer from *`uri`*.
Local	empty	required	The *`function`* receives the form.
Local	empty	empty	Shows a table of data.

Table 1: Form parameters and data management

1.4.7 Get

The pseudo type **`GET`** can be used for retrieve data from Internet via Ajax for set defaults values or populate lists and combo boxes or to periodically update comments, texts or images:

`GET,[* |` *`name`*`],`*`URI`*`[,`*`timeout`*`]`

if *`timeout`* is present, *`URI`* is called every *`timeout`* milliseconds and the control type *`name`* is updated.
`URI` is an Internet function that provides the data that are treated depending on the request:

- if `*` the program expects the data in the format provided by the default pseudo-type **`DEFAULTS`** (this is useful for populate a form by a Data Base fields);
- if *`name`* is provided without *`timeout`* the program expects the data in the form of *extra* field of **`CMB`** or **`L`** (lists and combo boxes) or a single value for others controls (example a text field);

- if *name* is provided with *timeout* the program expects a simple text if name refers to a comment or text control or a name of an image file possibly followed by a description (separated with tabulation), see example below.

The optional *query* component of the *URI* (preceded by a question mark (?)), contains data that depend on the protocol of the script receiving the request (see example below).

☞ The defaults of Combos, Lists and Radio buttons, unlike the case of pseudo-type **DEFAULTS**, is accepted only the value of the key.

```
Form,frm,,echo.php,receiveData;
Image,Image,,200;
Get,Image,getImage.php,11000;
```

```
<?php
$images = array(
       ["Rabbit lake","images/RabbitLake.jpg"],
       ["Bukavu - DR Congo","images/Bukavu.png"],
       ["Brousse on Burkina","images/Burkina.png"],
       ["Mount Olympus","images/Olimpo.jpg"],
       ["Conte Verde","images/ConteVerde.jpg"]);
if (!isset($_COOKIE['imgCount'])) {
       $count = 0;
} else {
       $count = $_COOKIE['imgCount'];
}
setcookie("imgCount", (($count+1) % count($images)));
echo $images[$count][1]."\t".$images[$count][0];
?>
```

Example 8: PHP script for periodic update image

Form Parameters

```
Form,frmg2,Form Generator 2,echo.php;
U,Time;
CMB,WidgetType,Widget Type;
CMB,Hellas,Greek letters;
List,Town;
B,fg_Ok,✎,40;
B,fg_Cancel,✘,40,,Cancel Form;
B,fg_Reset,↶,40,,Reset Form;
Get,*,getSample.php?Type=Defaults;
Get,WidgetType,getSample.php?Type=Type;
Get,Town,getSample.php?Type=Towns;
Get,Hellas,getSample.php?Type=Hellas;
GET,Time,getSample.php?Type=Time;
```

PHP script

```
<?PHP
$type = $_REQUEST["Type"];
if ($type == "Type") {
       echo "|=Buttons,B=Button,R=Radio button,RV=Vertical Radio button,"
       ."|=Lists,CMB=Combo box,L=List,"
       ."|=Texts,C=Comment,F=File,H=Hidden field,N=Numeric,NS=Numeric signed,"
       ."NF=Numeric with decimals,P=Password,T=Text,U=Unmodifiable text,"
       ."|=Others,CKB=Check box,S=Slider";
}
if ($type == "Hellas") {
       echo "Alfa,Beta,Gamma,Delta";
}
if ($type == "Towns") {
       echo "London,Paris,Rome,Turin,Zurich";
}
if ($type == "Defaults") {
       echo "Town=Turin,Hellas=Alfa,WidgetType=F";
}
?>
```

Example 9: Obtain data via Get pseudo type

1.4.8 Link

`Link,comboName,fieldName`

It is also accepted the form `Link,fieldName,comboName`

This command allows to insert a value taken from a combo into a text field.

☞ The `Link` command must be after the fields concerned.

1.5 The (experimental!) callback on form events

The purpose is to have access to the form in particular events, the call is:

```
function callBackFunction(eventType,formID)
```

Event name	Note
`Before`	Before the insertion of form in DOM
`Start`	
`AfterSubmit`	
`End`	When the form is erased

```
function seeDrugCard(item) {
  formGen("Right",drugCard,cBackDrug);
}
...
function cBackDrug(evnt,frmID) {
  if (evnt == "Before") {
      $("Right").innerHTML = "wait...";
      return
  }
  frm = $(frmID);
  if (evnt == "Start") $("fDfg_Title").innerHTML = "Drug "+ $("fDName").value;
  else if (evnt == "AfterSubmit") {
      $("fDQty").value -= $("fDwQty").value;     // access via ID
      frm["Items"].value -= frm["wItems"].value; // access via name
  }
}
```

Example 10: Use of Callback event function

1.6 Data presentation

The data are presented in the order they appears in the parameters list, except for the buttons that appears together the buttons inserted by *FormGen* at the bottom of the form or, possibly, at right or below a control (see par. 1.4.1 After and Below).

The buttons inserted automatically (*standard buttons*) are `Ok`, `Cancel` and `Reset` button, they have the name respectively `fg_Ok`, `fg_Cancel` and `fg_Reset` their presence depends on the controls contained in the form:

- there are no buttons if there is only one Combo box or one Radio buttons set (**`CMB`**, **`R`** and **`RV`** types),

otherwise:

- the `Cancel` button is present if the form is not declared `static`,
- the `Reset` button is present if there are data fields (e.g. Type **`F`**, **`T`**, **`R`**, **`CHK`**, **`CMB`**, **`S`**, etc.),
- the `Ok` button is not present if there are type **B** or **GB** buttons not associated to a field (i.e. by `After` or `Below` pseudo type).

The form is displayed using a `table` tag which has a class name `fg_Table`, the buttons have the class name `fg_Button` or `fg_Gbutton` (Graphic button). The first `td` tag of every rows has class name `fg_Label`; the possibly title has class name `fg_Title`.

In the following paragraphs there are some examples of styling by CSS.

1.6.1 Form container

If the form container doesn't exists or is not indicated the form is build in a `div` created and shown as a popup, this `div` has class name `fg_PopUp`. If the form container is not indicated the div has the `id` `fg_PopUp`.

```
.fg_PopUp {
  background-color: #5ca599;
   box-shadow:10px 10px #4c9589;
}
```

1.6.2 Buttons

For change the caption of `Ok` or `Reset` or `Cancel` button the syntax is:

```
B,[fg_Cancel|fg_Reset|fg_Ok],newCaption;
```

The Unicode characters are a simple and efficient means to create buttons with pictures[2]:

```
B,fg_Cancel,&#x2718;;
B,fg_Reset,&#x21B6;;
B,Start,&#x270E;,,myHandler,Go;
```

Table 2: Some UNICODE characters

Name	Symbol	PHP Code	HTML Entities
edit	✎	\270E	✎
delete	✘	\2718	✘
check	✓	\2713	✓
check bold	✔	\2714	✔
email	✉	\2709	✉
cross	✖	\2716	✖
dollar	$	\0024	$
euro	€	\20AC	€
pound	£	\00A3	£
white square	▢	\25A2	▢
Ballot box		\2610	&#x2610
Ballot box with check	☑	\2611	&#x2611
Eye	👁	\1F441	&#x1F441

The Ok button is replaced if there is almost one type **B** or **GB** control in the list not associated, by `AFTER` or `BELOW` pseudo type, to some control.

The default order of buttons is `Ok`, `Reset` and `Cancel`, when they are explicitly indicated the order is the one in which they are in the list.

CSS:

Buttons

```
.fg_Button {
        font-size:10pt; width:78px; height:24px; margin:0px 3px;
        background:silver;line-height: 1.25;
        border:outset;
}
.fg_Button:hover {background-color:#eee; cursor:pointer;}
.fg_Button:disabled {cursor: not-allowed;}
```

Graphic buttons

```
.fg_GButton {
  border: none;
  background: none;
  cursor:pointer;
  padding-left:2px;
```

[2] If mixed with button with text the style of the button must contains:

```
.fg_Button {...font-size:10pt;line-height:1.25;...}
```

```
    vertical-align:middle;
  }
```

1.6.3 Labels

CSS:

```
  .fg_Label {text-align: right;padding: 0px 5px;}  /* padding left right */
  .fg_Label:after {content: ":";}
```

1.6.4 Table rows

CSS:

```
  .fg_table td, th {border: 1px solid #111;padding:3px}
  .fg_table td {font: normal 9pt Arial}
  .fg_Title {
      font: bold 9pt Arial;
      text-align: center;
      padding:5px;
      background-color:#acc;
   }
  tr:nth-child(2n+1) {background-color:#eee;}
  tr:nth-child(2n+2) {background-color:#ffffff;}
  .fg_Buttons {background-color:#acc;} /* button's row */
```

1.6.5 Title

CSS title with image:

```
.fg_Title {
    font: bold 10pt Arial;
    text-align: center;
    padding:5px;
    background-color:#acc;
}
.fg_Title img {
     padding: 0 5px;
     vertical-align:middle;
}
```

1.6.6 Movable form.

```
.fg_PopUp {
    background:#E0E0E0;
    box-shadow:10px 10px #BFBFBF;
   width: auto;
    height: auto;
    position: absolute;
}
.fg_Movable {
  width: 100%;
  cursor: move;
}
```

1.6.7 Not modifiable fields

CSS to render `U` types as label:

```
.fg_UType {
    border: none;
    border-width: 0;
    box-shadow: none;
    background: transparent;
}
```

1.7 Events

A form is created with some events added:

- Event `change`:
  - display a value of slider,
  - for solitary combo box, radio and list,
  - for combo box and list for capture the possibly group name.
- Event `click`:
  - on buttons for submit, cancel and reset the form,
  - on the undo mark (✘) on the right in the text fields to clear its contents,
  - on check boxes.
- Event `blur` for a possible check of the entered value.
- Event `keypress`:
  - suppress enter key.

Moreover when a form is created one can add events by means of the `id` of the control:

```
JS
$("result").innerHTML = $("agree").innerHTML;
Fgen = new formGen("result");
$("Agree").addEventListener("click",function() {$('Start').disabled = !
this.checked;},true);

HTML
<span id='agree' style='visibility:hidden'>
CKB,Agree,Consent cookies?,10,I agree;
B,Start;
Defaults,Start=Off;
</span>
<span id=result></span>
```

Example 11: Enable button on event

☞ Note that `IDs` are formed by the form name and the field name, in the example above there is no `Form` pseudo type although the form is generated with id = `fg_Form`.

☞ for Radio buttons the `id`*s* are *formIDname*$_0$, *formIDname*$_1$ ...

☞ Events can be set by pseudo type `Event`:

```
setDecimals = function() {
      if ($("frmXsource").value.indexOf("%") > 0) $("frmXdecimals").value = 2;
      else $("frmXdecimals").value = 0;
}
var parmXData = "Form,frmX,Cross Data,call_crossdata.php,show,,static;"
      + "CROSS Product BY Town % ROWS Qty FROM orders,"
      + "CROSS Product BY Town Qty FROM orders,"
      + "CROSS Product BY Seller % SUM Sold FROM orders,"
      + "CROSS Product BY Seller FROM orders;"
      + "H,fnz,,,statement;"
      + "H,decimals,,,0;"
      + "H,decPoint,,,.;"
      + "Event,source,change,setDecimals;"
Fgen = formGen("formCross",parmXData);
```

Example 12: Use of event pseudo type

1.8 Accessing data

The possibly function associated to a Button and the custom function that manage the events can needs to access the data on the form: these can be accessed by *ctrlName* or by `Id`, in this case note that the Id name is `[`*formName*`]`*ctrlName*.

1.8.1 Button function

The function is called passing the form, the fields can be accessed by the *name*:

frmHandler.elements.*ctrlName*.value

Example:

```
function myHandler(frm) { // submit
            alert(frm.elements.Sensors.value)
            var aErrors = Fgen.check(frm);
            if (aErrors.length > 0) {
                  alert("Errors:\n"+aErrors.join("\n"));
                  return;
            }
            frm.encoding = "multipart/form-data";
            frm.target = "_blank";
            frm.submit();
}
```

1.8.2 Handle events functions

These functions receive the object that generate the event; the control that generated the event can be accessed by *this*; the supplied function `$` that stand for `document.getElementById(`*idName*`)` can be used to access the fields in the form: `$(`*formName*`).elements.`*ctrlName* or `$(`*formName*`).`*ctrlName*[3] or directly through his `ID`: `document.getElementById(`*formNamectrlName*`).`

Example:

```
<span id='form' style='visibility:hidden'>
Form,frm;
CMB,Sensors;
...
</span>
...
function retrieveSensor() {
      alert($("frmSensors_Group").value + " " + this.value)
}
...
$("frmSensors").addEventListener("change",retrieveSensor,true);
```

1.9 Errors

1.9.1 Alerted errors

AFTER, BELOW: control *controlName* not exists	
field not exists	in pseudo type `Control`
Error: *ajx.status*: *ajx.statusText*	when the form is submitted

1.9.2 Comments errors

fieldName Unknown type	Field or pseudo unknown
AFTER BELOW: field *fieldName* doesn't exists	
$fieldName_1$ or $fieldName_2$ doesn't exists	Pseudo type `Link`

1.9.3 Console logged errors

fieldName event field not present	Pseudo type `Event`
event *function* isn't a function	Pseudo type `Event`
fieldName get field not present	Pseudo type `Get`

[3] Attention: names can conflict with the HTML attributes of the form.

1.10 Some functions

`formGen` prototype contains some public functions:

If the form is created calling `formGen` these must be accessed whit this syntax:
`formGen.prototype.`*`functionName`*`(...)`
otherwise if the form is created by a new object like *`fGen = new formGen(arguments)`*:
`fGen.functionName``(...)`

1.10.1 Ajax

`ajax(`*`url`*`,`*`frm`*`,`*`handler`*`)`

Used internally for submit the form; *`handler`* is the function which receives the answer.

For external use: `formGen.prototype.ajax(`*`url`*`,`*`frm`*`,`*`handler`*`)`

examples:

```
1. formGen.prototype.ajax("getjson.php?getData,"",function(c){alert(c)})
2. formGen.prototype.ajax("FaRo_ajax.php?fnz=drugsList",frm,function(c){$
   ("Right").innerHTML = c})
3. formGen.prototype.ajax("FaRo_ajax.php","fnz=seeNames&limit=15&Name=" +
   x.value,function(c) {formGenInsertDrug.createOptions("Names",c)})
```

In the first example the data is in the *`url`*, in the second data are both in the form `frm` both in *`url`*. *`frm`* can be a form or a string and is returned as the second parameter, this is useful if the same callback function is called many times (internally is the case of **`GET`** pseudo type).

1.10.2 Get the ID associated with the event

```
formGen.prototype.IDEvent function (e) {
     if(window.event) return event.srcElement.id;     // IE
     else return e.target.id;                         // Netscape/Firefox/Opera
}
```

1.10.3 Change the contents of combo box

The function `createOptions` permits to populate or replace a combo box list contents.

`createOptions(`*`idComboName`*`,`*`newData`*`)`

ex.
```
...
Fgen = new formGen("result",list);
...
Fgen.createOptions("frmUnit","in=Inch,ft=Feet,yd=Yard,Mile")
```

1.11 Controls and form submission

Form data are sent when the `Ok` button is pressed (or his substitute) and there aren't errors.
The `check(`*`form`*`)` function execute the required controls on fields; possibly multiple controls for the same field are in `and` condition. The errors are alerted (see below an example of custom management).

Data are sent depending on the type of submission required (see Table 1: Form parameters and data management). If the script in the Web Server is PHP, data are in the global variable `$_REQUEST`, and `$_FILES` in case of file upload. In the case of local treatment data are properties of the form and can be accessed by the syntax:

`document.getElementById(`*`form`*`).`*`ctrlName`*`.value`

Where *`form`* is the name you have chosen in the **`Form`** pseudo type and *`ctrlName`* is the name of the control.

Moreover the form has also these fields:

- `fg_Button` contains the name of the button which has submitted the form or, in case of single combo, list or radio, the name of the field, in case of `event enter` is the name of the field;
- `fg_Store` used internally, contains the reference to the id of the form container: `fg_`*`ID`*.

- `fg_Changed` contains the list of fields changed. ☞ This is achieved by comparing the initial content of the form (including default values) and the submitted form; the initial content can be set calling the function `formGen.prototype.getFormFields` (see example below). The list also contains the buttons that have possibly changed status.

☞ The value returned of check box is present only if it is checked and his value is `on`; the fields `disabled` aren't returned, instead the fields `readOnly` are returned; the combo box aren't returned if there aren't be any choice.

The function `handle(`*`form`*`,`*`buttonName`*`[,`*`customHandler`*`])` is invoked when a button is clicked; this function invoke the `check(`*`form`*`)` function for execute the required controls on fields and it returns a possibly array of errors.

The `Cancel` button clears the form container; the `Reset` button cleans the form and restores the defaults values.

For others buttons, if the *extra* field doesn't contains a custom function to handle the event, the behavior is described in section 1.4.6 of pseudo type `Form`.

The function of the customer is invoked whit the form as argument, the name of the button involved is contained in the form field `fg_Button`.

```
Form,frm,Custom submitted form,x.php;
...
B,Start,,,myHandler;
Control,Number,min=-200,max=200,pattern=^[-+]?\d{1\x2C3}(\.\d{1\x2c2})?$;
NF,Number,,12,Insert Floating number;
P,psw,Password,15,Insert password;
T,Mail,Mail address,25;
Control,Mail,Required,mail;
...
function myHandler(frm) {
      var aErrors = Fgen.check(frm);
      if (aErrors.length > 0) {alert("Errors:\n"+aErrors.join("\n"));return;}
      frm.target = "_blank";
      frm.action = "http://127.0.0.1/condor/condorinformatique/x.php"
      frm.submit();
}
```

Example 13: Custom form control and submission

1.11.1 URI

The form is submitted to a page, by `Ok` button, the `Cancel` button doesn't submit; the form is erased unless it has been declared `static`.

1.11.2 Function

The function is called with the form as parameter, in case of `Cancel` button the form has only the field `fg_Button`; the form is erased unless it has been declared `static`.

1.11.3 URI and function

The URI is treated as an *ajax* requests and the JavaScript function receive the data. In case of `Cancel` button the form has only the field `fg_Button`; the form is erased unless it has been declared `static`.

1.11.4 No URI and no function

The data replace the form. In case of `Cancel` button the form has only the field `fg_Button`; the form is erased unless it has been declared `static`.

2 Examples

2.1 Local management of the form

```
function received(frm){
       var dt = [];
       dt[dt.length] = frm.type.name + "<td>"+frm.type.value
       dt[dt.length] = frm.name.name + "<td>"+frm.name.value
       dt[dt.length] = frm.label.name + "<td>"+frm.label.value
       dt[dt.length] = frm.length.name + "<td>"+frm.length.value
       dt[dt.length] = frm.Required.name + "<td>"+frm.Required.value
       dt[dt.length] = frm.Mail.name + "<td>"+frm.Mail.value
       dt[dt.length] = $("handleType0").name + "<td>"+frm.handleType.value
       $("insertedData").innerHTML = "<table><tr><td>"+dt.join("<tr><td>")+"</table>";
}
...
<span id='formGen' style='visibility:hidden'>
Form,frm,Form Generator,,received;
CMB,type,Type,,B=Button,CHK=Check box,CMB0Combo box,L=List;
T,name,Control Name,30,The name (and ID) of the control;
T,label,Control Label,30,The label shown of the control;
T,length,Control length,20;
T,extra,Extra field,100;
CKB,Required,,10,Check if field is required;
CKB,Mail,,10,Check if field is mail address;
Rdb,handleType,,10,A=Ajax,S=Submit,L=Local;
Defaults,length=20,Mail=on;
Controls,name,required;
</span>
```

Example 14: Sample of local handled form

3 History

0.1.3 18 December 2016	Corrected default for Radio buttons.
0.1.4 26 December 2016	Corrected error on slide min, max and step parameters.
0.1.5 February 2017	Corrected errors on value of Check Box, rebuild handling of reset; added the RV type (Vertical Radio button); Added the Defaults for Buttons i.e the value off for disable the button. Improved this manual.
0.1.6 28 April 2017	Added pseudo type Required.
0.1.8 14 June 2017	Supported forms whit same named fields, added the parameter static in the second extra field of pseudo type **FORM**: the form is generated without the Cancel button, data on not modifiable type can be set also by DEFAULT pseudo type.
0.2.0 16 October 2017	Added the pseudo type Get for retrieve data via Ajax. The form with only one radio buttons set or one combo box after selection is erased unless the form is declared static.
0.2.2 26 May 2018	The combo box returns the possibly group name.
0.2.3 5 June 2018	Correct bug in image buttons (from img tag to input type image tag).
0.2.5 31 December 2018	`Event` pseudo type added.
0.2.7 10 March 2020	Added the type CKL a set of Chek boxes, aesthetics change on slider
0.2.9 10 September 2020	• The pseudo type Get can update periodically texts and images • Added the type Image • Image can be set after some controls • Emended errors in function that close the form
0.2.10 October 2020	• The `ajax` function has been fixed in treating data as a string • The form is controlled also in personalized submission buttons • Hidden fields can accept default's value • `Link` pseudo type added • Experimental callback on form events
0.2.12 Avril 2021	• Added control of field and function existence on `Event` pseudo type. • Added the field f`g_Changed` that contains the list of fields modified. • Changed the return of data when the form pseudo type doesn't contains URI and JavaScript function. • Added the pseudo event `enter` which, relative to a text, invokes the associated function or submit the form. • Added the IB (inline button) type. • Added CE (comment error) and CS (comment separator) type. • The function callable by buttons can be `alert`. • Introduced the pseudo type `Dict[ionary]` for internationalization. • The form caption can contain images. • The attributes can be enclosed in `'` or `"` if they contain comma or semicolon.
0.2.13 Septembre 2021	• The password field has a button for see the value. • Popup management has been improved. • Some errors emended.

4 Technical notes

4.1 Multiple forms

It is possible to have multiple forms provided that have different names or, if they are unnamed, the field names are different, for the `ID` has the form `[formName]fieldName`.

4.2 Generated ID classes and names

Object	ID	Class	Name	Note
Form	*FormName* fg_PopUp	fg_PopUp	*FormName*	default Name fg_Form If the ID was not provided
			fg_Button	Contains the name of button pushed
			fg_Changed	List of fields changed
			fg_Store	Contains *fg_idOfFormContainer*
Table		fg_Table		
	*formName*fg_Title	fg_Title		if the form has title
		fg_Label		First td of every row
		fg_Buttons		Buttons' row (last)
Buttons		fg_Button		Ok button
		fg_Button		Cancel button
		fg_Button		Reset button
	buttonName	fg_Button		Generic button
	buttonName	fg_GButton		Image button
Combos, Lists	*ctrlName_Group*		*ctrlName_Group*	The possibly group name
Comment	*FormName*	fg_Comment		
	FormName	fg_Separator		For styling separator
	FormName	Fg_Error		For styling error
sliders	s*SliderName*			Where the slide value is shown
	ctrlName		*ctrlName*	For slider also an ID s_*sliderName*
	fg_*ctrlName*			Datalist
	fg_l_*ctrlName*			Input text of datalist
U Text type		fg_UType		
Text type	*name*		*name*	
	*name*_fg			The mark for erase field
		fg_TextArea		For Text Areas

4.3 Structures and variables

name	content	key	value
aButtons	Buttons properties	*buttonName*	See 1.
aLists	combo, radio values	*fieldName*	items (ex. alfa,B:beta,eta...
controls		*fieldName*	Array(*control*[=*value*], [*control*[=*value*]]...
defaults		*fieldName*	the value
errorArray	fields errors		*ctrlName*: *errorType*
events	Custom events	(array)	Array(*fieldName*,*event*, *function*,[*parameter*(s)])
jsForm	Form data		*name*,*caption*,*url*,*function*, [reset|static], *target*, *typeForm*, *formId* See 2.
links	Field associated to a combo	*comboName*	*fieldName*
listGroups	Group of combo item	*fieldIdValue*	group
store	Reference to some objects	fg_*idFormContainer*	See below.
formGen.store		JsForm defaults widgets controls listGroups aLists createOptions callBack buttons	**jsForm** reference defaults reference widgets reference controls reference listGroups reference aLists reference simulate option group possible function for handle events aButtons reference
widgets	controls	*fieldName*	Array of data of control

1. aButtons
   - BOTTOM|BELOW|AFTER,
   - *fieldName*,
   - *caption*,
   - *fieldType*,
   - *function*|HTML of *chekBox*|*input text*|*image*,
   - [*title*],
   - *btnWidth*
2. *name* is *ID* of the form if present; *typeForm* if contains "**F**" (i.e. there is a File control) the form must is multipart; *formId* is the Id of form: *name* or fg_Form if *name* is not present.

4.3.1 Object return methods and properties

createOptions

storeDefaults

`check`	Method for check field validity.
`formFields`	Method for set or retrieve modified fields
`fg_FormID`	ID of form

```
formGen[store] = {jsForm: jsForm,
     defaults: defaults,
     widgets: widgets,
     controls: controls,
     listGroups: listGroups,
     aLists: aLists,
     createOptions:createOptions,
     callBack: callBack,
     check:check,
     buttons:aButtons
     };
```

5 Annexes

5.1 Introduction to regular expressions

A regular expression is a string of characters used to search, check, extract part of text in a text; it has a cryptic syntax and here there is a sketch with few examples.
The regular expression is contained between `//` and can be followed by modifiers such as **`i`** to ignore the case.
The expression is formed with the characters to search in the text and control characters, among the latter there is a `\` said *escape* used to introduce the control characters or categories of characters:

- **`\` escape character**, for special characters (for example asterisk) or categories of characters:
  - **`\w`** any alphabetical and numerical character, **`\W`** any non alphabetical and numerical character,
  - **`\s`** *white space* namely. tabulation, line feed, form feed, carriage return, and space,
  - **`\d`** any numeric digits, **`\D`** any non digit,
- **`.`** any character,
- **quantifiers**, they apply to the character(s) that precede:
  - **`*`** zero or more characters
  - **`+`** one or more characters
  - **`?`** zero or one character (means possibly)
  - **`{n}`, `{n,}`** and **`{n,m}`** respective exactly *n* characters, almost *n* characters and from *n* to *m* characters.

`(...)` what is between parentheses is memorized,
`?=pattern` checks if *pattern* exists,
`[a-z]` any letter from `a` to `z` included,
`[a|b]` `a` or `b`,
`\b` word boundary,
`$` (at the bottom),
`^` (at start).

5.1.1 Examples

`^\s*$`	Empty set or white spaces
`aa+`	Find a sequence of two or more `a`, like `aa`, `aaa`,....
`(\w+)\s+(\w+)\s+(\w+)`	Find and memorize three words
`(\-[a-z])`	Find and memorize `minus` followed by one alphabetic character
`.(jpg|jpeg)$`	Controls file type jpg or jpeg
`^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$`	Control of mail address
`^\d+$`	Only integers
`((?=.*\d)(?=.*[a-z]+)(?=.*[\W]).{6,12})`	`(?=.*\d)` almost a digit from `0-9` `(?=.*[a-z])` almost one lowercase character `(?=.*[\W]+)` almost one special character `.` match anything with previous condition checking `{6,12}` length at least 8 characters and maximum 20
`^[-+]?\d{1,2}(\.\d{1,2})?$`	**Numeric values** `[-+]?` the sign is possible `\d{1,2}` one or two digits `(\.\d{1,2})?` It is possible to have a decimal point followed by one or two digits
`(?=.*\d)(?=.*[A-Z])(?=.*[a-z]).{6,12}`	At most one digit, one capital letter, one minuscule and

	from 6 to 12 characters

5.2 The Sand box

The Sand Box is an Internet application for demonstrate and try *FormGen*.
In particular it contains a `formgen.css` script for styling the forms and `form.js` that contains most of the control list of the demo.

5.2.1 Movable forms

In the *SandBox* there is an example of movable form (and internationalization).
This is achieved through a form generated without indicating the creation tag or indicating a non-existent tag, so *FormGen* generate a `div` tag with class `fg_PopUp`; ☞ the form must have the `form` pseudo type with the *`caption`* (the third parameter) in order to be generate a title row that is the area for the moving.
The script `moveItem.js` contains the code for move the form; the user must add the event listener for dragging the title whose id is *`formName`*`fg_Title` and, possibly, the code for positioning the form.

Widget List

```
form = "Form,ft,Try Sand Box,echo.php,receive;"
       + "T,Text1,Text 1,,place holder;"
       + "RV,vRdb2,vRdb 2,,North,South,West,East;"
```

JS Form creation

```
...
if($("fg_PopUp")) $("fg_PopUp").remove();
Fgen = new formGen("",form)
var link = $("fg_PopUp")
link.style.top = 0.5 * (window.innerHeight - link.offsetHeight);
link.style.left = 0.5 * (window.innerWidth - link.offsetWidth);
$("ftfg_Title").addEventListener("mousedown", dragStart.bind(null,
event, "fg_PopUp"))
```

CCS

```
...
.fg_PopUp {
      background:#E0E0E0;
      box-shadow:10px 10px #BFBFBF;
      max-width: fit-content;
      max-width: -moz-fit-content;
      position: absolute;
}
```

Example 15: Movable form

6 Indexes

6.1 List of Examples