JavaScript Form Generator

Disclaimer

This SOFTWARE PRODUCT is provided by El Condor "as is" and "with all possible 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`*; `underlined` item indicates a default value.

Contents table

1 Form generator

Form generator, briefly *FormGen*, is a JavaScript script which contains the class `fGen` that allows build and handle forms; *FormGen* is sufficiently generalized for create a wide set of useful forms from simple message box to relative complex input forms, based on a list of controls or widgets (some text type, buttons, check boxes, lists, radio buttons, comment and images); moreover *FormGen* supports event management, server interaction by Ajax and dynamic form management.

The form presentation can be customized both via CSS both by the instructions present in the description of the widgets.

The form can be submitted or managed locally.

Furthermore, the `fGen` class exposes some utility functions such as the management of floating objects, the creation of `DOM` objects, etc.

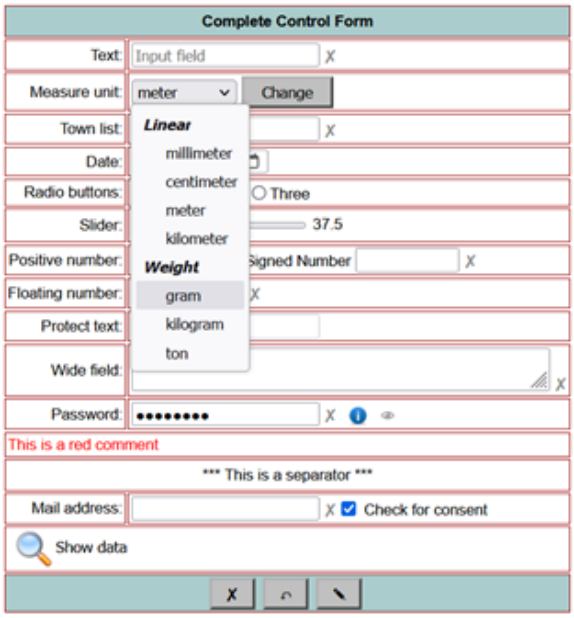


Figure 1: Example of form

1.1 Using the form generator

The form builder is contained in `formgen.js` script, which contains the class `fGen`.

This function can be invoked for create a new form object:

```
fGenObject = new fGen(containerID,control_list)
```

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

☞ If the `id` is not present, it is created a `<div>` tag with `id fg_PopUpn` and `class fg_PopUp` in order to create a movable form (see *1.5.3 Movable forms*).

The second parameter is a characters constant or variable containing the list of controls (widgets).

1.2 Data description

Every control is characterized by a list of attributes separated by space(s) in this order: *`Type`*, *`Field Name`*, *`Field Label`*, and *`Extra field(s)`*. Controls are separated by line terminators[1].

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

`Extra field(s)` are attributes that depends of the control type, if the attribute contains space(s) it must be enclosed by single or double quote.

☞ Some data (see *1.4.4 Dictionary*) can contain hexadecimal values to five digits in the form `\xnnnnn` the possibly commas, equals and `&` signs, must be coded respectively by `\x2C`, `\x3D` and `\x26`. Alternatively, the attributes can be enclosed by single or double quote in order to contains directly the above characters.

1.2.1 Type

The *`Type`* is indifferent to the case; if it starts with `//` it is a comment.

- **Buttons**:
  - **B** button;
  - **R**, **RDB** radio button, a set of Radio buttons;
- **CKB** check box;

[1] All line separators Windows, MacOS, Unix i.e. `CRLF`, `CR` and `LF` respectively.

- **CKL** check box list;
- **Combo box** and **lists**:
  - **CMB** drop down list for select an item;
  - **L** or **LIST** is a drop down list associated to a set of texts, where one can choose an item or insert one not present in the list;
- **I**, **IMG**, **IMAGE** image,
- **Text** fields:
  - **C**, **COMMENT** comment;
  - **DATE**;
  - **S** seek bar or slider;
  - **T** or **TEXT** text field (numbers, File, password);
  - **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 programs local or on 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, it has the form: *`formNameFieldName`* (*`formName`* is provided by pseudo type `Form` see parag. `1.4.6`).

The characters of the name must begin with a letter ([A-Za-z]) and may be followed by any number of letters, digits ([0-9]), hyphens ("-"), and underscores ("_")[2].

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

☞ The possible space(s) contained in the name are replaced by underscore if received by a PHP script.

1.2.3 Field Label

Is the label of control or the caption of button (in case of graphic buttons it is the name of the image on the server); if omitted it is used the *`FieldName`* that it is transformed if it has those formats:

- *`fieldName`* it becomes *`Field name`*,
- *`field_name`* it becomes *`Field name`*.

The label can contain images, the file name must be separated by space from the test:

```
R Sex 'Sex images/sex.png' 'M=&#9794; Male,F=&#9792; Female,N=Not specified'
```

1.2.4 Extra

`extra field(s)` is (are) used for add information to the control, these will be specified in the relative data description paragraphs.

Apart from the first extra field of combo box, check list and Radio buttons, *`extra field(s)`* can contains parameters in the form *`key [value]`*, or *`key=value`* for example:

```
T psFile 'PDF and PS files' Width 50 File 'Accept = .pdf,.ps'
```

☞ The order of parameters are indifferent, the parameter *`key`* is case indifferent.

1.2.4.1 After, Below

The `after` or `below` can be a parameter of buttons, combo boxes, check boxes, comments, radio buttons and texts that are shown after or below others widgets; buttons can also be added to the title of form and tab.

1.2.4.2 Anchor

In Comment `Anchor` *`hrefReference`* (opened in a new card).

[2] This is a subset of what is stated for HTML 4 and it is also compatible with CSS.

1.2.4.3 Call and Server

`call` *`function`*`|'`*`function parameter`*`'`
`server` *`URL`* `[target _blank|_self|_parent|_top|`*`frameName`*`]`

`function` can be:

- a user function (see paragraph *1.7.2 Handle events functions* for how the function is invoked);
- a *FormGen* static function (see paragraph *1.9.3 Static functions*)
  Example: `var div = fGen.createNode("DIV","fg_Box",'border: 1px solid #000')`
- a function declared into an object.

☞ *`function`* is invoked with some parameters, but the function declaration can have only:

`function``(`*`fieldID`*`|`*`serverAnswer`*`, `*`parameter`*`, `*`form`*`)`

`fieldID` is the Id of the control that generated the event
`serverAnswer` is the server response generated by a `server` *`URL`*
`parameter` is the possibly parameter

`alert` *`message`*
`enable|disable|switch` *`fieldName`*
`remove` *`formName`*

1.2.4.4 Class

`Class` *`className`*`|`*`className`*$_1$`,`*`className`*$_2$`[,...]|'`*`className`*$_1$ *`className`*$_2$`[ ...]'`

For add a customer CSS style; more than one class are accepted separate by comma or space.

1.2.4.5 Color

The color can be added to Text, comments, Date, Combo box and List.

1.2.4.6 Disabled

For text and buttons; ☞ The texts `disabled` are marked `readonly` for to be present when the form is closed.

1.2.4.7 Edit(able)

For slider, it permits to insert a value manually.

1.2.4.8 Event

Note that there is also a pseudo type `Event` for handle events: the syntax for inserting the management of an event relating to a control is illustrated below.

- `[Event` *`eventType`*`] server` *`URL`* `alert|call` *`function`*`|set` *`fieldName`*`|`*`ID`* (*)
- `[Event` *`eventType`*`] alert` *`message`*
- `[Event` *`eventType`*`] call` *`function`*`|'`*`function parameter`*`'`
- `[Event` *`eventType`*`] enable|disable|switch` *`formdName`*
- `[Event` *`eventType`*`] remove` *`fieldName`*

(*) `server` *`URI`* is an Ajax command, the response is alerted or is passed to the *`function`* or is inserted in the widget with the *`fieldName`* or *`ID`* indicated.

☞ `Event` *`eventType`* can be omitted for the event characteristic of the widget; see below:

- Button `click`
- Check box: `change`
- Combo box: `change`
- List `keydown: Enter key`
- Radio buttons: `change`
- Text `keydown: Enter key`
- Text with parameter `file`: `change`

Some examples of widgets with event declaration:

```
B Start &#x270E; width 40 call 'myHandler echo.php'
B ShowImage images/faro.ico inline 'Show image' server getImage.php set Image
Rdb imageType '' .gif,.jpg,.png call getImageList
B Clock images/clock.png inline 'Get Time' server getSample.php?Type=Time set
Text
B xExcel images/excel.png Event mouseover alert 'Create Excel file' inline
'Excel file'
```

Table 1: Examples of event on control

☞ Event parameters must be the last information of the control.

1.2.4.9 Link

`link` *`fieldName`* `[exposed|group]` (only for combo box) this parameter allows to put into *`fieldName`* the chosen value of the choice the combo box or its exposed value or its possibly group; in case of text area the text is added to the text that can be already present; *`fieldName`* can be a text field or hidden field or comment.

1.2.4.10 Title

For add `title` attribute to the control.

1.2.4.11 Value or Default

Sets the initial value of the control; for combo box and radio buttons must be the `key`.

1.2.4.12 Width

`Width` the width is on characters for texts fields; for Comments, Buttons and Sliders the width is in pixels.

1.3 Summary by type

1.3.1 Buttons and graphic buttons

`B` *`name caption|imageFile attribute(s)`*

`attribute(s)``:`

- `After|Below` *`fieldName`*
- `Class` *`className`*
- `Disabled`
- `Event interaction` see paragraph *1.2.4.3 Call and Server*
- `Inline` *`label`* the button is located between the controls and replaces the label, *`label`* is inserted after the button
- `Title` *`title`*
- `Width` *`nn`* the dimension of the button or image in pixels

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

Buttons have the class:

- fg_Button
- fg_Gbutton button with image, internal CSS rule is `border:none;background:none`
- fg_Cbutton one character button, internal CSS rule is `border:none;background:none;font-size:16px`

☞ See the `Event` pseudo type for a more flexible data management.

☞ see paragraph *1.7.2 Handle events functions* for how the function is invoked and data can be accessed.

1.3.2 Check box

`Ckb` *`name label atRightlabel attribute(s)`*

attribute(s):

- `After|Below` *fieldName* ☞ the field *label* is ignored
- `Class` *className*
- `Event interaction` see paragraph *1.2.4.3 Call and Server*
- `On` if it is checked on start
- `Title` *title*
- `Value|Default` *value* is the value returned when checked, if omitted the value is `On`

☞ In the form submitted the value returned is present only if the check box is checked.

1.3.3 Check box List

CKL type generates a set of vertical aligned check boxes.

`Ckl` *name label checkList attribute(s)*

attribute(s):

- `Class` *className*
- `Value|Default` can be one of check box that is checked

The *checkList* is 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.

To add an event handler to some of check box use the Event pseudo type (see example below).

The hidden field *name* of the check box list will contain the number of check boxes selected.

Ex.

```
CKL ProgramLanguages '' 'C=C\x2C C#,JS=JavaScript,PHP,PYTHON,RUST' Default JS
Event click on JS alert Javascript
```

1.3.4 Combo boxes and Lists

`Cmb|L|List` *name label items* `[`*attribute(s)*`]`

attribute(s):

- `After|Below` *fieldName*
- `Class` *className*
- `Color` *color*
- `Default|Value` *value|key*
- `link` *fieldName* `[exposed|group]` (only for combo box) this parameter allows to put into a text *fieldName* the value taken from the combo; in case of text area the text is added to the text that can be already present.
- `Event interaction` see paragraph *1.2.4.3 Call and Server*
- `Multiple` enables a multiple choice
- `Title` *title*

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 *items* fields contain a set of `key value` (see description in Radio button).

The `key`(s) can be duplicate.

If there is only one combo or list in the form, the form has no buttons and it is exited when a list item is selected or a value is inserted ending with Enter key (unless the form is static, see Form 1.4.6).

It is possible to have a combo with items grouped (the HTML `optgroup` tag), the group is identified by the syntax *=groupLabel*, see the example below.

Example 1: One choice without buttons

```
Form frm2 '' server echo.php call receive
```

```
CMB Unit 'Measure Unit'
=Linear,mm=millimeter,cm=centimeter,m=meter,km=kilometer,=Weight,g=gram,kg=kil
ogram,t=ton
```

☞ After submission the field *fieldName*_Group contains the possible group name(s) and the field *fieldName*_Exposed contains the value(s) shown.

1.3.5 Comment

The comment is displayed in one line, occupying both columns.

Comment|C [*fieldName*] '*some comment*' [*attribute(s)*]
attribute(s):

- After|Below *fieldName*
- Align Center|right|justify|left ☞ this has an effect only with width attribute
- Anchor *hrefReference* opened in a new card
- Class *className*
- Color *color*
- Default|Value
- Row|rows *n*
- width *pixels*

☞ Comments are in a span tag, for change the contents by program it can be used the innerHTML method on the ID *formNamefieldName*. ~~If the comment contains a carriage return it is converted in
.~~

The Row parameter force the height dimension of the comment and add a possibly scroll bar.

Example 2: Comments

```
Form fc 'Comments and error'
C '' "The label field is the comment shown: <br>Comment|C [fieldName] 'some
comment' align [center|right|justify] [width nnn]<br>Comment and C are
synonym. The comment can be aligned by inserting align center or right or
justify. Comments have the class fg_Comment." align Justify width 350
C '' <hr>
C '' 'images/faro.ico anchor comment with images' anchor images/Bukavu.png
C '' 'Below an error shown by formGen'
Link combo text
```

1.3.6 Date

Date *fieldName label* [*attribute(s)*]
attribute(s):

- Default|Value *yyyy-mm-dd*|today
- Class *className*
- Color *color*

HTML 5 Supported.
☞ The possibly default must be in the form *yyyy-mm-dd*; it is also accepted today.

1.3.7 Hidden field

H|Hidden *fieldName* [*value*]

Example:

```
H MAX_FILE_SIZE 5000
T Attachment 'Attachment file' file width 30 filter .gif,.jpg,.png
```

☞ MAX_FILE_SIZE sets the maximum file size, in bytes, accepted by PHP.

value can also be set by Default pseudo type.

1.3.8 Image

I|Img|Image *name label imageFile attribute(s)*

attribute(s):

- Height *pixels*
- Title *title*
- Class *className* default is fg_Image

If *label* is not present the image occupies all the row.

imageFile can be: *imageFile*|*imageFile*:*Description*|*Description*:*imageFile*

The possible *description* is shown before or after the image depending on its position.

```
Form frm 'Images' server echo.php call receive
I Img1 'El Condor' img/condor.gif title 'El condor pasa'
I Img2 'img/SanMichele.png:Sacra di San Michele' class
fg_Frame
CSS
.fg_Frame {
  border: 2px solid silver;
  margin: 5px;
  box-shadow: 4px 4px silver;
  vertical-align:middle
}
```

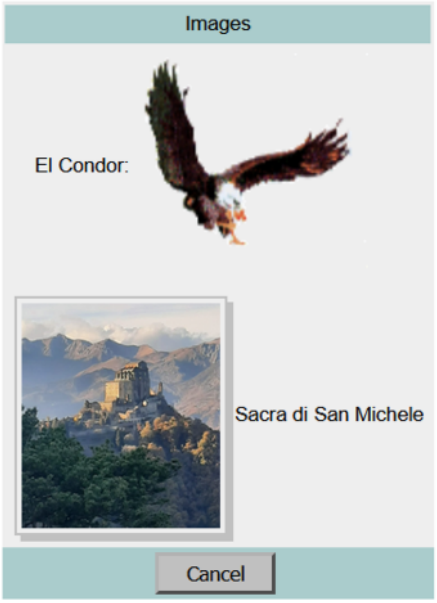


Figure 2: Form with images

1.3.9 Radio buttons

R|RDB *name* [*label*] *buttons attribute(s)*

attribute(s):

- Class *className*
- Event interaction see paragraph *1.2.4.3 Call and Server*
- Title *title*
- Value|Default *value*
- Vertical buttons are arranged vertically

The *buttons* field contains the labels and value of each radio button separated by commas. To obtain a key instead of the label, the item must have the form: *key*=value.

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

The set of items can be enclosed in ' or " if it contains spaces:

Example 3: Radio buttons example

```
CSS .fg_Table,.fg_Title,.fg_Buttons
{background:#acc}
CSS .fg_Table td, .fg_Table th {border: 1px
solid #444}
Form rdb 'Radio buttons example' server
echo.php call receive
R Status ''  M=Married,S=Single,W=Widow
R Sex 'images/sex.png'  'M=&#9794;
Male,F=&#9792; Female,N=Not specified'
R Output ''  E=images/excel.png,None
R Nations '' 'It=images/its.png
Italia,Fr=images/frs.png
France,Es=images/ess.png
```

```
España,Us=images/uss.png United
States,El=images/els.png Ελλάδα' vertical
Defaults Nations=El Sex=M
```

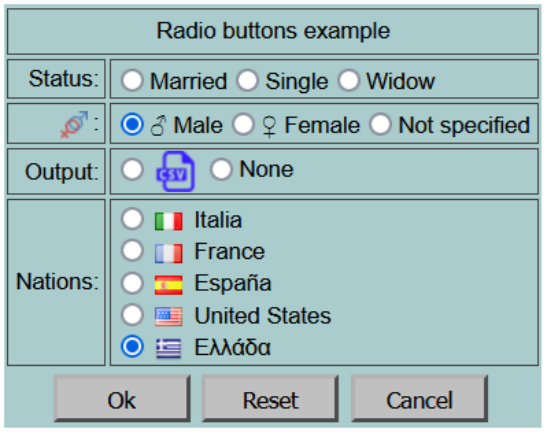


Figure 3: Radio buttons with images

☞ The `value`(s) fields can contain images, the file name must be separated by space; the image can be `.bmp`,`.gif`, `.png`, `.jpg`, `.ico` or `.jpeg`.

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 button is selected; the form is erased (unless the form is static, see Form 1.4.6).

1.3.10 Slider

`S name label` *`attribute(s)`*

`attribute(s)`:

- `Class` *`className`*
- `Color` *`color`* is the color of the value exposed
- `Edit|Editable` allows to insert a value manually
- `Event interaction` see paragraph *1.2.4.3 Call and Server*
- `From` *`value`* if omitted is `0`
- `Step` *`value`* if omitted is (`To - From)/100`
- `Title` *`title`*
- `To` *`value`* if omitted is `100`
- `value|Default` *`nn`*
- `width` *`nn`*`|150` pixels

Ex. `From -5 To 5 step 0.5`

The slider has a text after that show the value.

The result can have decimals depending on the value of *`To - From`*.

☞ `name` and `id` are associated to the text that has the class `fg_Slider`; the id of the slider is the `id` of the text prefixed by `s_`.

1.3.11 Text fields

`T|Text name label` *`attribute(s)`*

`attribute(s)`:

- `accept|filter` *`filterList`* for input type **`file`** *`filterList`* `is the` value for the `accept`[3] attribute ex. `filter=image/*,.pdf`
- `col|cols` *`nn`*
- `Color` *`color`*

[3] See https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/accept#unique_file_type_specifiers

- `disabled`
- `Event interaction` see paragraph *1.2.4.3 Call and Server*
- `file`
- `float` for floating numbers
- ~~`hex` for fields with hexadecimal values~~
- `hint` *`placeholder`*
- `integer` for numeric fields
- `password`
- `positive` for integer positive fields
- `row|rows` *`nn`*
- `title` *`title`*
- `value|default` *`value`*
- `width` *`nn`*`|20` characters

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

`disabled` shows a not modifiable texts; note that the field is returned when the form is exited.

If the `width` exceeds 50 characters generated a text area `50 x Width/50`.

If `col` and/or *`rows`* is present is generated a text area `width|col x` *`rows`*.

The `hint` parameter sets a text hint (HTML5 `placeholder` property); ☞ if the length exceed the field width the `hint` becomes a title.

```
T mediaFile '' File width 50 Filter audio/*,video/*,image/*
T psFile 'PDF and PS files' File Width 50 Filter .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 1.3.7 Hidden field).

1.4 Pseudo types

1.4.1 Add style

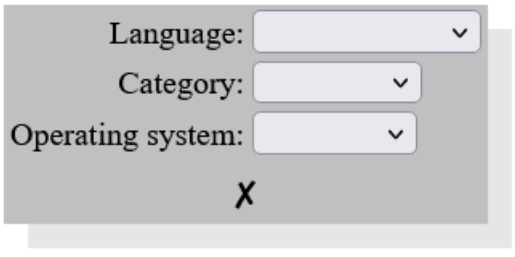
`CSS` is a pseudo type that allows to add styling elements.

`CSS` *`stylingElements`*

`stylingElements` is added to a tag `style` that is generated and appended to `document.head`, it is like acomponent of style sheet with the limitation that it must be on a single line.

Characters after the closed brace are ignored i.e. in fact they are comments.

```
CSS .fg_Table, .fg_Table td {border:
none;background-color:silver;}
CSS .fg_Table {box-shadow: 12px 12px rgb(128 128 128
/ 0.2);}
Form f '' nobuttons
CMB Language '' '' call listSoftware
CMB Category '' '' call listSoftware
CMB OS 'Operating system' '' call listSoftware
Get Language condor_ajax.php?fnz=Language
Get Category condor_ajax.php?fnz=Category
Get OS condor_ajax.php?fnz=OS
B fg_Reset &#x2718; width 40 title 'Reset combos'
```



1.4.2 Controls on data

`CONTROL` or `CHECK` allows to perform validity checks on fields:

`CONTROL|CHECK` *`fieldName control`*

`control := compOperator value|fieldName 'error signal'`

```
        control := is mail|regularExpression 'error signal'
        control := call function 'error signal'
REQ|Required fieldName1[ fieldName2[ ...]]
```

Where *fieldName* is the name of the control subject to check; *compOperator* is a comparison operators.
The `mail` field **is** checked by regular expression **`^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,3}$`**,
function is called with the form, the field name and the field value (see example below).
The field is valid if the control(s) executed is `True`; if there are multiple controls for the same field, they are considered in `and` condition; control aren't executed if the field is empty.
Some examples:

`Control e_mail is mail 'Incorrect mail form'`	field type is a mail address
`Control Password is '(?=.*\d)(?=.*[a-z]).{8,}' 'Almost eight character one numeric'`	at least 8 characters of which at least one is numeric
`Control positiveNumber >= 0 'Number must be greater -1'`	numeric field positive
`Control Min < Max 'Minimum must be less of Maximum'`	

Example 4: Function that controls fields

```
...
T Qty 'Stock Quantity' positive
T wQty 'Quantity withdrawn'
Check wQty call controlWhithdraw 'excess quantity'
...
function controlWhithdraw(frm,field,value) {// check Quantity withdrawn
  if (value > parseFloat(frm["Qty"].value)) return false;
  return true;
}
```

1.4.3 Defaults

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

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

- For list, combo box and radio button the *value* must be the key; in the combo box if key isn't unique the first item with the key is defaulted.
- The values constant, like `on`, `today` etc., are case insensitive.
- For Date type the format must be *yyyy-mm-dd*; it is also accepted `today`.
- If the value contains space(s) the couple *ctrlName=value* must be delimited by single or double quotes.

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

1.4.4 Dictionary

```
Dict[ionary] dictionaryObject|From function [parameter]
```

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 (not the key if it differs from the exposed value).

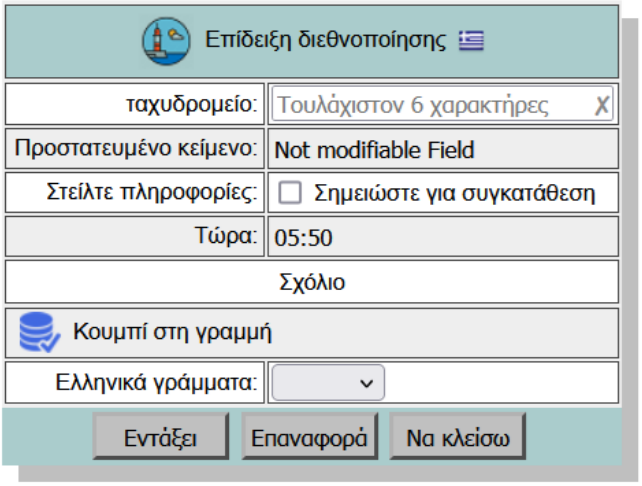


Figure 4: Internationalization

☞ The translation is not applied to the default values.
☞ `dictionaryObject` must be a global variable.
The dictionary is contained in the static variable `fg_dictionary` of `fGen` class, This allows you to indicate the dictionary only at the beginning of the application and possibly change it, if required.

HTML

```
...
<img src="img/its.png" class="imgbtn" onclick="changeLang('IT')"
title="Italiano">
<img src="img/frs.png" class="imgbtn" onclick="changeLang('FR')"
title="Français">
<img src="img/ess.png" class="imgbtn" onclick="changeLang('ES')"
title="Español">
<img src="img/uss.png" class="imgbtn" onclick="changeLang('EN')"
title="English">
<img src="img/els.png" class="imgbtn" onclick="changeLang('EL')"
title="Ελληνικός">
...
```

Control list

```
Form ft 'images/faro.ico Demo internationalization images/its.png' server
echo.php call receive
Dict dict
T Mail Mail address ''  hint 'Minimum 6 characters'
T Protect 'Protected text' value 'Not modifiable Field' disabled
CKB CheckBox 'Send info' 'Check for consent'
T Time '' disabled
C Comment Comment center
B Save images/update.png 'inline=In line button' Event click alert 'Not saved,
only for demo'
GET Time getSample.php?Type=Time
CMB Hellas 'Greek letters' Alfa,Beta,Delta,Epsilon,Gamma
```

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]
     +" server echo.php call receiveData/nDict dict"
     +formTrans
     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").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"}
```

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"}
            }
```

1.4.5 Event

This pseudo type `Event` is used to attach an event handler to a field:

`event` *`eventType`*`|enter on` *`fieldName`*`[,`*`fieldname...`*`] submit|`*`Attributes`*

`Attributes`

- `call` *`jsAndParm`* `|alert` *`text`*
- `server` *`URI`* `submit|call` *`jsAndParm`*`|alert|set` *`fieldName`*`|`*`ID`*

`jsAndParm` `:=` *`javaScriptFunction`*`|'`*`javaScriptFunction parameter`*`'`

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 `keydown` event.

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

`submit` invokes the form submission.

`server` *`URI`* calls, via *ajax*, a *`serverFunction`* (for example a `PHP` script) passing him the form; the response is passed to *`JavaScriptFunction`* with possibly *`parameter`* or alerted or, in case of `set` *`fieldName`*`|`*`ID`* *formGen* treat the response as follows:

- an `IMG` tag: the result is entered in the `SRC` property,
- an `INPUT` tag: the result is entered in the `VALUE` property,

- a `SELECT` tag: the result, that must have the structure of the *extra* field of combo type, is inserted as options,
- else the result is entered in the `innerHTML` property.

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

```
Form fe 'Submit on Enter or Select' server echo.php call receive
T Name ''
Event Enter on Name Submit
T Qty Quantity positive
```

Example 5: Event, Get and createOptions function

```
...
<div id='result'></div>
<div id='result2'></div>
...
<script type='text/javascript'>
var eventFrm = "CMB Images"
      + "\nRdb imageType '' .gif,.jpg,.png"
      + "\nB fg_Cancel &#x2718"
      + "\nB ShowCite images/new.png inline 'Show IT quote'"
      + "\nEvent click on ShowCite server getITCite.php set result2"
      + "\nDefaults imageType .png"
      + "\nGet Images getSample.php?Type=Images&imageType=.png"
      + "\nEvent click on imageType call getImageList"
      + "\nEvent change on Images call showImage result2";
Fgen = new fGen("result",eventFrm);
function getImageList(btn) {
      const frm = $(btn).form;
      const url = "getSample.php?Type=Images&imageType=" + event.target.value;
      fGen.prototype.ajax(url,"",c => frm.fg_createOptions("Images",c))
}
function showImage(field,div) {
  $(div).innerHTML = "<img src='images/" + event.target.value + "'>";
}
</script>
```

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 title [Attribute(s)]`*

`Attribute(s)`:

- `call` *`javascriptFunction`* ☞ the argument is the form
- `class` *`className`* ☞ the class is associated with the title
- `ground` *`CSSBackground`* default is transparent (`rgba(0,0,0,0)`)
- `left` *`pixel`*`|-1` ☞ only for popup forms
- `nobutton[s]` standard buttons aren't generated *
- `onStart` *`javascriptFunction`* ☞ the argument of function is the form
- `reset` the form isn't effaced and the fields are restored at the initial value
- `server` *`URL`*
- `set` *`fieldName`*`|`*`ID`* to receive the server response
- `static` the form isn't effaced
- `target _blank|_self|_parent|_top|`*`frameName`*

- `top` *`pixel`*`|-1` ☞ only for popup forms

* However, it is possible to insert custom buttons.

`name` is the ID assigned to the form, if it is omitted the ID is `fg_Form`*n*.

`title` is displayed, if present, above the controls; *`title`* can contain images (`.bmp`, `.gif`, `.png`, `.jpg`, `.ico` or `.jpeg`):

```
Form ft 'images/faro.ico "Demo internationalization" images/els.png' server
echo.php call receive
```

☞ the image file names must be separated from the text by space(s) and this must be enclosed by quotes if it is subject to translation.

`URL` or `server`*`Function`* is the server script which receive the form (via `submit` or `ajax`), if it is not present the form is not submitted and *`javascriptFunction`*, if present, is called with the form as argument.

`reset` restore the form after submission (like the `Reset` button),

`static` a `Cancel` button isn't generated.

☞ The form is erased by `Cancel` button. The form is cleared if has the `reset` parameter.

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 2: 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 [every milliseconds] [call function]
```

if *`every`* is present, *`URI`* is called every *`milliseconds`* and the widget *`name`* is updated; ☞ this happens only if *`milliseconds`* are greater of `99`.

`URI` is an Internet function that provides the data that are treated depending on the request:

- `*` it is used to obtain default data, for example data from a database; the data must be in JSON format (see the example below);
- if *`name`* is a name of one form field of type:
  - `IMG` the result is entered in the `SRC` property,
  - `INPUT` (texts and lists) the result is entered in the `VALUE` property,
  - `SELECT` the result must have the structure expected for the list of options,
- else the result is entered by the `innerHTML` property.

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.

☞ The possibly *`function`* is called with the result of the query before his treatment allowing any modification; if the function returns false no data is entered in the form.

Example 6: PHP script for periodic update image

```
Form frm '' server echo.php call   <?php
receiveData                        $images = array(
I Img '' height 200 comment                ["Rabbit lake","images/RabbitLake.jpg"],
Get Image getImage.php every 11000         ["Bukavu - DR Congo","images/Bukavu.png"],
                                           ["Brousse on Burkina","images/Burkina.png"],
                                           ["Mount Olympus","images/Olimpo.jpg"],
                                           ["Conte Verde","images/ConteVerde.jpg"]);
                                   if (!isset($_COOKIE['imgCount'])) {
                                          $count = 0;
                                   } else {
                                          $count = $_COOKIE['imgCount'];
                                   }
                                   setcookie("imgCount", (($count+1) % count($images)));
                                   echo $images[$count][1]."\t".$images[$count][0];
                                   ?>
```

Example 7: Obtain data via Get pseudo type

Form Parameters

```
Form frmg2 'Get example' server echo.php call receive
T Time '' disabled
T Widget '' disabled
T piGreco '' value 3.14159 float disabled
CMB WidgetType '' '' link Widget group
CMB Hellas 'Greek letters' multiple
List Town
CMB Languages
Hidden HiddenField
B fg_Ok &#x270E; width 30
B fg_Cancel &#x2718; width 30 'title=Cancel Form'
B fg_Reset &#x21B6; width 30 'title=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
GET Languages getSample.php?Type=Lang call addSomeLanguage
```

Example 8: Function called on data received by GET command

```
function addSomeLanguage(c) {
      var lang = ("Python,Ruby,Basic,"+c).split(",");
      lang.sort()
      return lang.join(",")
}
```

Example 9: PHP script for GET command

PHP script

```
<?PHP
$type = $_REQUEST["Type"];
if ($type == "Type") {
       echo "=Buttons,B=Button,R=Radio button,R vertical=Vertical Radio
button,"
       ."=Lists,CMB=Combo box,L=List,"
       ."=Texts,C=Comment,T file=File,H=Hidden field,T Positive=Numeric,T
integer=Numeric signed,"
       ."T float=Numeric with decimals,T password=Password,T=Text,T
readonly=Read only text,"
       ."=Others,CKB=Check box,S=Slider";
```

```
}
if ($type == "Hellas") {echo "Alfa,Beta,Delta,Gamma,Epsilon";}
else if ($type == "Towns") {echo
"London,Paris,Rome,Toulon,Toulouse,Turin,Zurich";}
else if ($type == "Defaults") {echo {echo
'{"Town":"Turin","Hellas":"Alfa","WidgetType":"T
file","Languages":"Pascal","HiddenField":"El Condor"}';};}
else if ($type == "Lang") {echo "Algol,Cobol,Fortran,JavaScript,Pascal,PHP";}
else if ($type == "Time") {date_default_timezone_set("Europe/Rome");echo
date("h:i");}
?>
```

1.4.8 Required

`Req[uired] fieldsList`
Example: `Required Mail Measure`

1.4.9 Tab

`Tab name caption [title]`
The Tab pseudo type permits the creation of Tabs; below the structure generated by the control list.

Control list structure	Presentation	Table structure
Common widgets Tab Tab_1 Tab_1 *widgets* ... Tab Tab_n Tab_n *widgets*	Title	thead
	Common widgets	tbody
	Tabs navigator	
	Tabs container	tbody
	Common buttons	tfoot

Table 3: Tab structure

The widgets before the first Tab are commons to all `Tab`s.
Every `Tab` has a `Reset` button and possibly buttons present after the pseudo typo `Tab`.

1.5 Data presentation

The data are presented in the order they appears in the parameters list unless they are expressly placed `after` or `below` another field; the buttons appears together the buttons inserted by *FormGen* at the bottom of the form if they aren't assigned by `After` or `Below` to another field.
The buttons inserted automatically (*standard buttons*) are `Ok`, `Cancel` and `Reset`, 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 from Combo box, Radio buttons set, Text field or Date field, 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 **`Text`**, **`R`**, **`CHK`**, **`CMB`**, **`Slider`**, etc.),
- the `Ok` button is not present if there are buttons (type **B**) not associated to a field i.e. by `After` or `Below` parameters.

The form is displayed using a `table` tag which has class name `fg_Table`, the buttons have `fg_Button` or `fg_GButton` or `fg_CButton` class respectively for buttons with text or buttons with image or one character text. The field labels have class name `fg_Label`; the possibly title has class name `fg_Title`.
In the form there are some embedded styles:

```
.fg_Buttons {text-align:center;padding:3px 0}
.fg_Error {color:red}
.fg_Number {text-align:right;margin-right:12px}
.fg_Erase {color:#888;margin-left:-12px}
.fg_See {margin-left:6px;font-size:20px}
.fg_ButtonTab {border-top-right-radius:15px;height:30px;min-width:80px;border:1px solid
#000;padding:5px;border-bottom:none;background:rgba(0,0,0,0)}
.fg_Button,.fg_ButtonTab,.fg_Erase,.fg_See,.fg_CButton,.fg_GButton,.fg_CheckBox
{cursor:pointer}
.fg_CButton {border:none;background:rgba(0,0,0,0);font-size:16px}
.fg_Button:disabled, .fg_GButton:disabled, .fg_CButton:disabled.fg_CheckBox:disabled{cu
rsor: not-allowed;}
.fg_GButton {border:none;background:none}
.fg_Slider {padding-left:5px;border:none;background:rgba(0,0,0,0)}
.fg_UType {border:none;background:rgba(0,0,0,0)}
.fg_Table td {padding:3px 2px}
.fg_alignAfter {display: grid;grid-template-columns: max-content repeat(3,1rem);align-
items: center;}
```

Note that those styles can be overridden by setting `!important` to the style, for example for change the color of the erase marker the CSS can be: `.fg_Erase {color:blue !important;}`
The presentation can be manipulated using style sheets or by the pseudo type `CSS`.

1.5.1 Form container

If the form container doesn't exists or is not indicated the form is build in a `div` with `id` `fg_PopUp`*n* and class name `fg_PopUp`, see parag. *1.5.3 Movable forms*.

1.5.2 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:

```
B fg_Cancel &#x2718;
B fg_Reset &#x21B6;
B Start &#x270E; Event click Call
myHandler
```

The Ok button is replaced if there is almost one type Button 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.

☞ if the button caption is of one character, possibly written in UNICODE notation the button has no border and no background ☞ therefore this is no the case if the character is a HTML Entity.

Name	Symbol	PHP Code	HTML Entities	JavaScript UNICODE
edit	✎	\270E	✎	\x270E
delete	✘	\2718	✘	\x2718
check	✓	\2713	✓	\x2713
check bold	✔	\2714	✔	\x2714
email	✉	\2709	✉	\x2709
cross	✖	\2716	✖	\x2716
dollar	$	\0024	$	\x24
euro	€	\20AC	€	\x20AC
pound	£	\00A3	£	\xA3
white square	▫	\25A2	▢	\x25a2
ballot box		\2610	☐	\x2610
ballot box with check	☑	\2611	☑	\x2611
Eye		\1F441	👁	\x1F441

Table 4: Some UNICODE characters

1.5.3 Movable forms

In the *SandBox* there is an example of movable form (and internationalization).
This is achieved through a form generated without indicating the `containerID` or indicating a non-existent `ID`, so *FormGen* generate a `div` tag with class `fg_PopUp`; ☞ the `title` of the form is the area where the move is activate, if there is not a pseudo type `Form` all form is responsive to the move (but without the move cursor).

Example 10: Movable form

Widget List

```
form = "Form ft 'Try Sand Box' echo.php receive"
       + "\nT Text1 'Text 1' hint 'place holder'"
       + "\nRDB vRdb2 'vRdb 2' North,South,West,East vertical"
```

JS Form creation

```
...
if($("fg_PopUp")) $("fg_PopUp").remove();
Fgen = new fGen("",form)
```

CCS

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

1.6 Controls and form submission

Form data are sent to URI or function 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.

Data are sent depending on the type of submission required (see Table 2: Form parameters and data management). If the script in the Web Server is a PHP script, 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 chosen in the `Form` pseudo-type and `ctrlName` is the name of the control.

Moreover the form has also some other 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_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.
- `fg_TimeStamp` contains the browser date and time in the form: `YYYY-MM-DD HH:MM:SS`.

☞ 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 `fg_handle(form,buttonName)` 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 not `After` or `Below` a field, if doesn't contains a custom function to handle the event, the behavior is like an `Ok` button.

1.6.1 URI

The form is submitted to a server, by `Ok` button (or his substitute) , the `Cancel` button doesn't submit; the form is erased unless it has been declared `static`, however the `Cancel` button, if present, can erase the form.

1.6.2 Function

The function is called with the form as parameter after a possibly positive check; the form is erased unless it has been declared `static`.

1.6.3 URI and function

The URI is treated as an *ajax* requests and the JavaScript function receive the response from the WEB server. The form is erased unless it has been declared `static`.

1.6.4 No URI and no function

The data are shown as popup; the form is erased unless it has been declared `static`.
☞ No URI and no function is the case also when the form is submitted by a custom function (see the above paragraph).

1.6.5 Custom management of form

The function of the customer is invoked whit the button id[4] as argument:

- the form is accessed by: `const frm = $(button).form`
- the possibly controls: `var aErrors = frm.fg_check(frm);`
  return in case of error detected: `{alert("Errors:\n"+aErrors.join("\n"));return;}`
- the form can be submitted, provided the `form` pseudo type has the server parameter: `frm.submit()`

Example 11: Custom form control and submission

```
Form form 'Example Form'
T Text '' width 30 hint 'Text placeholder'
S Slider '' From 34 To 43 step 0.1
T psw Password Password width 25 title 'Insert password'
T graphFile '' File filter .gif,.jpg,.png
Control psw is (?=.*\d)(?=.*[A-Z])(?=.*[a-z]).{6,12} 'Almost one Uppercase,
Lowercase and digit\x0d from 6 to 12 characters'
B Start &#x270E; width 40 event click call 'myHandler echo.php'
Defaults Slider=37.55 psw=Corkone6
Required graphFile
...
function myHandler(button,URI,frm) {      // can be called with button or form
      if (typeof button == "string") {
            var frm = $(button).form
            var aErrors = frm.fg_check(frm);
            if (aErrors.length > 0) {alert("Errors:
\n"+aErrors.join("\n"));return;}
      } else frm = button
      frm.fg_formFields(frm)  // set list of widgets changed
      frm.encoding = "multipart/form-data";
      frm.target = "_blank";
      if(URI != "") frm.action = URI
      frm.submit();
      $("result").innerHTML = "The form has been erased in myHandler function";
      frm.remove()
```

[4] *In reality the complete signature is:* `functionName(fieldID,parm,form)`

```
}
```

1.6.6 Form with one control

If the form contains only one input field the form is submitted when the field is compiled or selected, it may contain, however, images, hidden fields and comments:

```
Form frm '' server echo.php call receive
I Image_2 'images/SagraSanMichele.png:Sacra di San Michele' class fg_Frame
RDB Agree '' Low,Medium,High
```

Table 5: Events on one field form

Type	Event(s)	Note
`Combo (CMB)`	`change`	
`Date`	`change`	
`List (L)`	`change` `keydown`	`keydown` exits on Enter key
`Radiobutton`	`change`	
`Text`	`keydown`	exits on Enter key

`Text file`	`change`	☞ this works only if the form is sent via Ajax:
		`Form frm 'Upload .png file' server echo.php call receive`
		`T .pngFile '' file accept .png`

1.7 Events

A form is created with some events added depending on the control:

- Event `change`:
  - sliders: display a value of slider,
  - for solitary combo box, radio, Date, File field and list.
- 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 eye icon of password fields.
- Event `keyup`:
  - for manage numeric fields,
- Event `keydown`:
  - for capture the Enter key for form submission,
  - for solitary list to intercept the Enter key.

Moreover events can be added by the `Event` pseudo type or by JavaScript `addEventListener` method:

Example 12: Enable button on event

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

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

```
      <span id=result></span>
```

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

Example 13: Use of event pseudo type

```
setDecimals = function() {
      if ($("frmXsource").value.indexOf("%") > 0) $("frmXdecimals").value = 2;
      else $("frmXdecimals").value = 0;
}
var parmXData = "Form frmX 'Cross Data' server call_crossdata.php call show static"
      + "\nCMB source '' 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"
      + "\nH decimals '' value 0"
      + "\nEvent change on source call setDecimals"
Fgen = new fGen("formCross",parmXData);
```

1.7.1 Button Events

This paragraph deals on not submit buttons i.e. the buttons that are AFTER or BELOW a form field.
The behavior is influenced by the presence or absence of the Event parameter i.e. without Event the buttons acts as submit button; with Call or Server parameter the main cases are (see parag. 1.4.5 Event):

- only Server *URI URI* is submitted via Ajax without response.
- Server *URI* and Call *function* the answer of the server is managed by the JavaScript *function*.
- Only Call *function* the *function* receive the button id and a possible parameter; note that in the JavaScript function it is possible call the server via Ajax, see the example below.

Example 14: Call a function that call Ajax

```
Form '' 'Call function example'
B GB1 'images/info.png' inline Quote click call 'showITCite 15'
C quote   width 300
...
function showITCite(btn,n,form) {
      fGen.prototype.ajax("getITCite.php?n=" + n, form,
            function(c){$("quote").innerHTML = c})
}
```

1.7.2 Handle events functions

The pseudo type **Event** allows you to assign to a field both the script server and the JavaScript that will process the response, moreover, if instead of the JavaScript function is indicated an element of the DOM, this will receive the server data (see paragraph. 1.4.5 Event).
The function has this signature:

functionName(*fieldName*,*parm*,*form*)

fieldName is the name of the object that generated the event; the data on the form can be accessed by *fieldName* or by *fieldId*, in this case note that the *fieldId* is [*formName*]*fieldName* :

form.ctrlName.value or document.getElementById(*ctrlId*).value

Example:

```
Form frm
CMB Sensors
...
function retrieveSensor() {
      alert($("frmSensors_Group").value + " " + event.target.value)
}
...
$("frmSensors").addEventListener("change",retrieveSensor,true);
```

Example 15 Adds `change` event to a combo box and `enter` to a text input

```
Form fe 'Submit on Enter or Select' server echo.php call receive
T Name '' Event Enter Submit
CMB Category '' '=Anti,Antibiotic,Anti-inflammatory,=Others,Beta-
blocker,Cardiovascular,Dermatological,Endocrine,Gastroenterological,Gynecologic
al,Neurological,Respiratory,Restorative' Event change Submit
```

1.8 Errors

1.8.1 Alerted errors

`Error: ajax.status: ajax.statusText`	when the form is submitted

1.8.2 Errors reported as comments

`Unknown type: fieldType`	Field or pseudo unknown
`function isn't a function`	Form call function
`ID the form ID exists`	Form name coincide with an existent ID
`CSS incorrect`	Lack of { or }

1.8.3 Console logged errors

`Attempt to remove non-existent id: id`	In remove function
`Ajax timeout after nnnms`	Server not responding
`Default value value for field not in list`	Combo box and Radio buttons defaults
`fieldName not existent`	Pseudo type `Control` and `Get`, commands `switch`, `enable` and `disable`
`fieldName field ID not present`	Pseudo type `Event`
`function isn't a function`	Pseudo type `Event`
`fieldName get field not present`	Pseudo type `Get`
`Event for field fieldName not present`	Event
`Field fieldName or id nonexistent`	Event Set parameter

1.9 Functions

`fGen` contains some functions in the object prototype or are referenced as form property or declared static:

- Call prototype functions
  - `fGen.prototype.functionName(...)`
  - `fGenObj.functionName(…)` where `fGenObj` is the name of the object `fGen` instantiate
- Call functions accessible from the form
  - `fGenObj.functionName(...)` where `fGenObj` is the name of the object `fGen` instantiate
- Call static functions
  - `fGen.functionName(...)`

1.9.1 Form exposed functions

A set of function where the address is in the form: *formid.function*.

1.9.1.1 Check the form data

The function `fg_check(form)` performs the control indicated by the pseudo types `Control` and `Required` returning an array of errors or an empty string; see *Example 11: Custom form control and submission*.

1.9.1.2 Get data from form

The form data are obtained:

```
data = form.fg_formFields(form)
```

data is a hash table of data of form; the key is the control name, it contains all fields including those not submitted in submission (like check box not checked).

form can be the form `ID` or the form itself.

```
formGen = new fGen(containerID,"Form frm ...")
...
var data = formGen.showData(frm.fg_formFields("frm"))
```

1.9.1.3 Set or change the combo box content

```
fg_createOptions(id,optionList)
```

The *id* of the widget that has the form *formName.widgetName*, the *optionlist* has the form of the homonym parameters of the combo (CMB) widget type.

```
function changeMUnits(field,parm,frm) {
        const newCombo = "=Time,s=Second,m=Minutes,H=Hour,Day,Month,Year"
        frm.fg_createOptions(frm.MeasureUnit.id,newCombo)
}
```

1.9.1.4 Set widget value

```
fg_setValue(widgetName,value)
```

1.9.2 Prototype functions

The prototype functions can be invoked `fGen.prototype.`*function* or *obj.function* where *obj* is the formGen instantiated.

1.9.2.1 Ajax

```
Promise = fGen.prototype.ajax(url,data,handler|ID[,parameter(s)])
data := form|dataForm|querystring|hashTable
```

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

examples:

1. `fGen.prototype.ajax("getjson.php?getData,"",function(c){alert(c)})`
2. `fGen.prototype.ajax("FaRo_ajax.php?fnz=drugsList",frm,"Right")`
3. `fGen.prototype.ajax("FaRo_ajax.php","fnz=seeNames&limit=15&Name=" + x.value,function(c) {fGen.prototype.createOptions("Names",c)})`
4. `onClick='fGen.prototype.ajax("../condor_ajax.php?count&name=JsFormGen")'`

☞ In the first example the data is in the *url*, in the second data are both in the form `frm` both in *url* and the result is put in the DOM element with ID `Right`; in the fourth example the `ajax` function has only the *url*.

If *data* is present it can be:

- a reference to a form;
- a `FormData` object;
- a *querystring* example: `key1=value1&key2=value2`;

- a hash table example: `{key1: value1, key2: value2}`.

handler function is invoked, if present, with the response, the invoking form and the possible one or two *parameter(s)*.

1.9.3 Static functions

These functions do not require the creation of the `fGen` object.

1.9.3.1 Create node

```
Node = createNode(tagType,id[,style])
Node = createNode(tagType,attributes[,style])
```

Examples:
```
var span = fGen.createNode("span")
fGen.createNode("DIV",formID+"fg_GridData",'border: 1px solid #000')
```

1.9.3.2 Create widget

```
createWidget(id,widgetList)
```

This function allows the insertion of widget(s) rows, with possibly `After` widgets, and `Hidden` field(s); the rows are inserted after the row that contains the widget with ID *id*.

The row tag and any hidden fields have the property `data-form` that contains the name of the form (the name is a generated name: `fg_formn`), this is useful in case it is necessary to delete the added line.

☞ The widgets can have an event associated: the third parameter of the handling function is not the form.

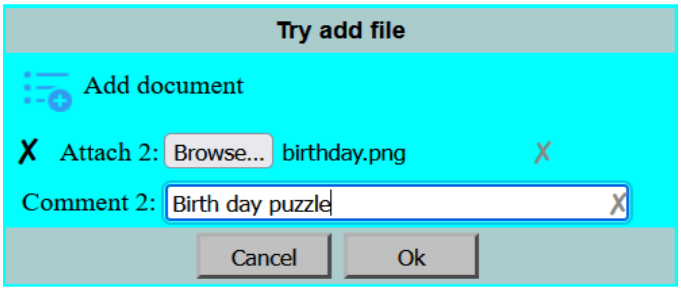
☞ The `reset` button doesn't restore the widget added.

☞ The *id* of the widget inserted is the widget name: note that it isn't prefixed by the form name of the module to which it is added.

Example 16: Insertion of form elements

```
                              *** widget List ***
CSS #frm_Table {width:400px;background: #0ff}
Form frm 'Try add file' server echo.php call receive
B AddFile images/add-list.png inline 'Add document' call insertFile
B fg_Cancel Cancel
B fg_Ok Ok
                              *** JavaScript ***
insertFile.count = 0;
function insertFile(id) {
     insertFile.count++
     var list = `
B delete%n \\x2718 call fGen.deleteWidget inline
T attach_%n '' after delete%n width 40 File
T Comment_%n '' width 35
`
     var idAfter = id
     fGen.createWidget(idAfter,list.replace(/%n/g,insertFile.count))
}
```

☞ In the example above the widgets are inserted after the Button `AddFile` for the first parameter of the function associated is the button *id*; for insert after another widget the above fragments can be modified (see below) keeping that the id is formed by the form name followed by the widget name.




```
...
T Comment '' width 30
B AddFile images/add-list.png inline 'Add document' call 'insertFile frmComment'
...
function insertFile(ButtoniId,id) {
...
```

1.9.3.3 Delete widget

`fGen.deleteWidget(`*id*`)`

id is the id of a widget in the row that will be deleted; the function can be invoked by a program or by event on the widget belonging to the row to be eliminated.

`fGen.deleteWidget(`*id*`,`*id2*`)`

this form of invocation is used when the widget that capture the event isn't in the row to be deleted, in this case *id2* is the id of a widget in the row that will be deleted.

☞ If *id* or *id2* is the Id of a form the form is removed.

1.9.3.4 Extract tokens

The function `fGen.extractTokens(s, delimiter)` generates an array of tokens from a character string where the tokens are separated by space(s) or delimited by quote or double quote if they contains spaces.
If `delimiter` is present and is `true` the tokens are returned with possibly string delimiters.

1.9.3.5 Get the widget added

`getAddedWidgets(id)`

Returns an array of the widgets added.

1.9.3.6 Is graphic file

`fGen.isGraphicFile(`*fileName*`)`

The function returns `true` if the file has one of those extension: `.png`, `.gif`, `.jpeg`, `.jpg`, `.ico`

1.9.3.7 Move an object on the screen

The function `fGen.dragStart` makes an object movable as long as it has the style `position:absolute`.
The function is the second parameter of the `addEventListener` function with event `mousedown` and must know the object Id, see the example below.

Example 17: Movable form

```
<div id='condor' style='position:absolute;cursor:move'>
...
</div>
...
$ = id => document.getElementById(id);
$("condor").addEventListener("mousedown", fGen.dragStart.bind(null,"condor"))
```

1.9.3.8 Position an object on the screen

`fGen.setObjPosition(`*Object*`, `*top*`, `*left*`)`

The HTML object is positioned as indicated by *top* and *left* values; if these are omitted the object is centered on the screen.

☞ The object must have in the style `position:absolute`.

1.9.3.9 Show image

`fGen.showImage(`*ID*`,`*imageNameDescription*`)`

imageNameDescription := imageFile|imageFile:Description|Description:imageFile

ID must be an ID of an IMG tag.

The possible *Description* is inserted before or after the image provided there are a `previous` or `next` sibling tag.

1.9.3.10 Show data

`fGen.showData(`*data*`)`

data is a key/value object.

Creates a table (class `fg_Table`) with ordered data by key.

In the fragment below (contained in the form data are displayd by *FormGen*.

```
var data = fGen.showData(formData);
setTimeout(d => new fGen("",d),100,`Form '' Data\nC '' "${data}"`)
```

1.9.3.11 TimeStamp

`fGen.timeStamp` returns the browser date and time in the form: `YYYY-MM-DD HH:MM:SS`

Ex. `fGen.prototype.timeStamp(new Date())`

1.10 Compatibility

`Date`, `List`	HTML 5	
`Get` pseudo type	Explorer 9	The function called by `setInterval` is not switch off when the form is erased.

1.11 Sandbox

The Sand Box is an application for demonstrate and try *FormGen*; it can be also used for build a (skeleton) of control list.

Sandbox contains a `formgen.css` script for styling the forms and `form.js` that contains most of the control list of the demo.

2 History

0.3.0 April 2024	• First release
0.3.1 August 2024	• Fixed loss of styles of static form • Removed (not signaled) incompatibility between widget names and Form tag properties, ex `id`, `method`
0.4.1 October 2024	• Eliminated GET for combo with one item setting default • Add parameter `row` to comment • Improved and corrected documentation • The possibly call to `onStart` function is now the last action of the form creation
0.4.2 May 2025	• Added Tab pseudo • Improved the form movable • In the Comments, Carriage Return is no longer replaced by ` ` • Added the ability to insert/delete widgets into a module already created
0.4.3 August 2025	• The Combo box items obtained by GET pseudo type are added to the possibly existing items • Added to the event commands event `enable|disable|switch` *`fieldName`*
0.4.4 January 2026	• `Class` parameter on `form` pseudo type is associated to the table that contains the form • In forms with tab these have the same width (not yet the same height) • Amended the documentation the `Cancel` button simply remove the form • ☞ In the `Get` pseudo type he possibly *`function`* is called with the result of the query before his treatment. • Added to the event commands event remove `form`*`Name`* • `ajax` function is realized with `fetch` command and returns a `Promise`

3 Technical notes

3.1 Multiple forms

It is possible to have multiple forms provided that the controls have different names or, however to avoid name collision the field names has the form *formNamefieldName* where *formName* is the name of the pseudo type `Form`.

3.2 Generated ID classes and names

Note that `ID` are prefixed by *FormName* if exists; hereinafter, *ctrlName* means *FormName* + *ctrlName*.

Object	ID	Class	Name	Note
Form	*FormName*|fg_Form*n*	fg_Form		fg_Form*n* if the *FormName* was not provided
PopUp form		fg_PopUp		
Added returned fields			fg_Button	Contains the name of button pushed or the lonely widget
			fg_Changed	List of fields changed
			fg_TimeStamp	Timestamp of submission
Images		fg_Image		
Table	*FormName*_Table	fg_Table		
	*FormName*_Title	fg_Title		The possible th containing the title
		fg_Label		First td of row with widget
Tabs	*FormNameTabName*	fg_Tab		Tab tbody
	*FormNameTabName*_Tab Title	fg_TabTitle		
	*FormNameTabName*_Tab	fg_ButtonTab		Button tabs
Buttons	*FormName*fg_Ok	fg_Button (*)	fg_Ok	Ok button
	*FormName*fg_Cancel	fg_Button (*)	fg_Cancel	Cancel button
	*FormName*fg_Reset[*n*]	fg_Button (*)	fg_Reset	Reset button. *n* in case of tab reset button
	FormNamebuttonName	fg_Button (*)	*buttonName*	Generic button
	FormNamebuttonName	fg_Gbutton	*buttonName*	Image button
	FormNamebuttonName	fg_CButton	*buttonName*	One character button
	*FormName*fg_Buttons	fg_Buttons		Bottom buttons container
Check box		fg_CheckBox		
Combos, Lists			*ctrlName_Group*	The possibly group name
			ctrlName_Exposed	The exposed value

Object	ID	Class	Name	Note
Comments	*ctrlName*	fg_Comment		
Sliders	s_*ctrlName*			The slide
	*ctrlName*_List			Datalist
Text type	*ctrlName*			
	ctrlName	fg_UType		Disabled text widgets
		fg_Erase		✘ erase sign
		fg_See		Button to see password
	ctrlName	fg_TextArea		For Text Areas
Added Styles	*FormName*_fgCSS			

(*) the class can change if the button is single character or image.

3.3 Structures and variables

name	content	key	value
aGets	Get pseudo	array	See paragraph Get 1.4.7
controls		*fieldName*	Array(*control*[=*value*], [*control*[=*value*]]...
errorArray	fields errors		*ctrlName*: *errorType*
events	Custom events	*fieldName*	Array(*event*, *function*,[*parameter*(s)])
jsForm	Form data		Contains the properties: • containerID • eventOnStart the possible function to call before the display of the form • left possible left position of the movable form • title • server HTTP request to manage the form • call local function • moveID the ID where the move occurs: ◦ the title area ◦ the form if the pseudo type form is not present or it has no title field • nobuttons no automatic buttons • prefix the form name • static 1 static, 2 static and reset form • target default _blank • top possible top position of the movable form • encoding if there is a File control • ID is the ID attributed to the form, is the form *name* or fg_Form*n* if *name* is not present.
widgets	widgets	*fieldName*	Array of widget's components; can contain also some

			properties: • call • default • disabled • hint • ID • inline • place After|Below • placevalue • server • source • type float|integer|positive • width char or pixel

4 Annexes

4.1 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,
- `.` (point) any character,
- **quantifiers**, they apply to the character(s) that precede:
  - **`*`** zero or more characters
  - **`+`** one or more characters
  - **`?`** zero or one character
  - **`{n}`**, **`{n,}`** and **`{n,m}`** respective exactly ***n*** characters, almost ***n*** characters and from ***n*** to ***m*** characters **.**

`(...)` what is between parentheses is memorized, (unless see line above)
`(?:...)` a non-capturing group,
`?=`*`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).

4.1.1 Regular expression examples

`^\s*$`	Empty set or white spaces
`(\w+)\s+(\w+)\s+(\w+)`	Find and memorize three words
`(\-[\W])`	Find and memorize `minus` followed by one alphabetic character
`.+\.jp[e]?g$`	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+$`	Signed integer
`^[+-]?(?:\d+|\d+\.\d+)$`	Floating point number
`((?=.*\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
`[0-9A-F]{1,5}`	Up to 5 hexadecimal digits (with case insensitive flag)
`(?=.*\d)(?=.*[A-Z])(?=.*[a-z]).{6,12}`	At most one digit, one capital letter, one minuscule and from 6 to 12 characters

5 Indexes

5.1 List of Examples

5.2 List of Tables

5.3 List of figures