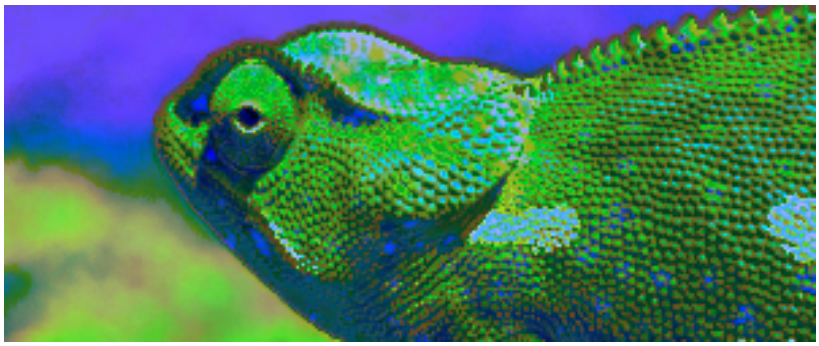


Blue Chameleon CMS
(Object management)

For links outside this document,
download the relevant chapter or the

Blue Chameleon Content Management System full documentation.

May 8, 2012

Contents

List of Figures

List of Tables

Chapter 3

Object management

Now that the use of Blue Chameleon Content Management System's main functions has been detailed, it is time to describe which useful objects are integrated into pages.

What follows principally deals with how elements like images, downloads,... are created/uploaded, then integrated as variables and in leaf model contents :

- a leaf frame model's contents, as written via **Headline models** / **Modify** / **a leaf frame model** is told to `INCLUDE` objects such as images, links, text areas, using **variables** ;
- those are defined via **Headline models** / **Variables** / **a leaf frame model** according to the type of object ;
- afterwards, when a leaf frame as based upon this model will be edited through  , i.e. the values of the variables will be given thanks to menus, browsers...

3.1 Images

User group rights for Images to be handled are detailed at 1.3.4.

Images are that are uploaded through Blue Chameleon Content Management System are afterwards available whenever a Choose image button is available, for instance when a leaf frame contains an Image variable (cf. Fig.2.6).

3.1.1 Uploading an image

Fig.3.1 shows how to do so ; there, the following is inputed :

- a description (name under which this image will appear) ;

- its type (`jpg` or `gif`) ;
- optional width and height in pixels (if not filled, the image will be shown in its native size) ;
- a caption ;
- where on your computer it will be uploaded from ;
- a folder (**only folders available under the current column and the Publisher are available**) ;
- whether to automatically publish image or not.

It is to know that if this checkbox is not ticked, it will not be available for pages. If an image has been uploaded without having been published, it can nonetheless be published anytime through **Image** / **Publish**.

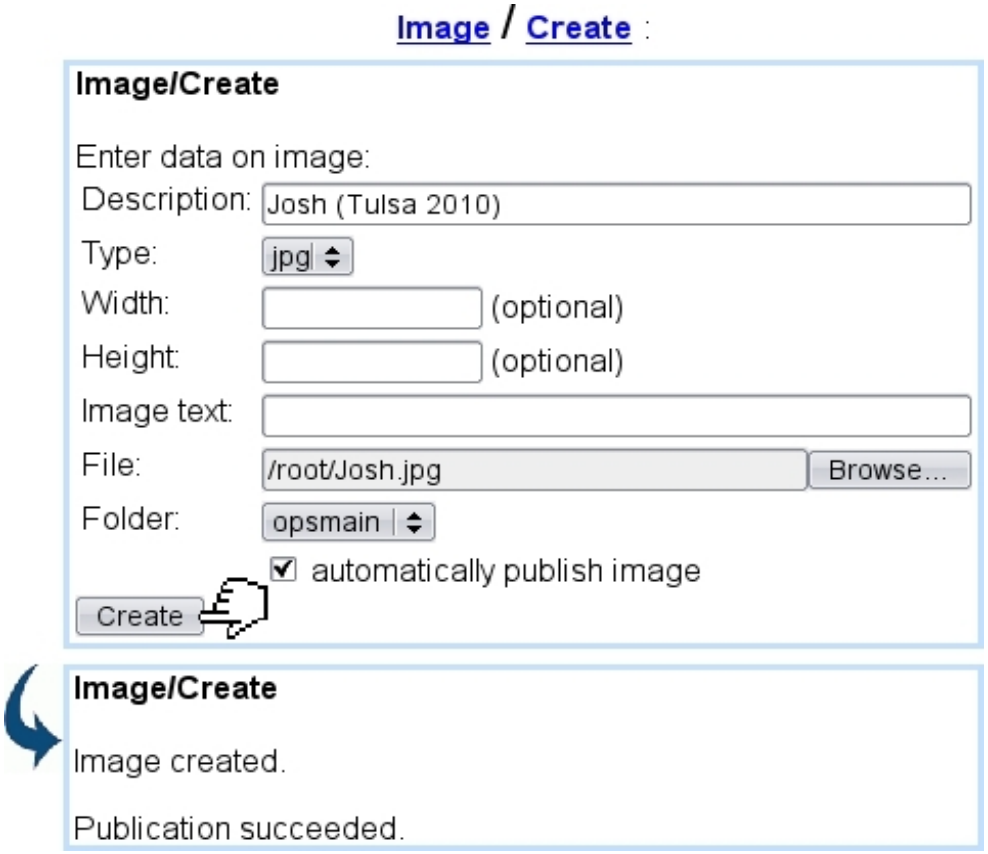


Figure 3.1: Uploading an image.

After uploading, all previously-entered data (except for folder location) can be updated through **Image** / **Modify**.

3.1.2 Image folders

Provided that group rights for images are sufficient, it is possible, in the **Image** environment, to create a folder under any column ; for an image to be uploaded onto a new folder not under the current column, the column will have to be changed, though (1.4.3.1).

Folder creation is then done as featured in Fig.3.2.

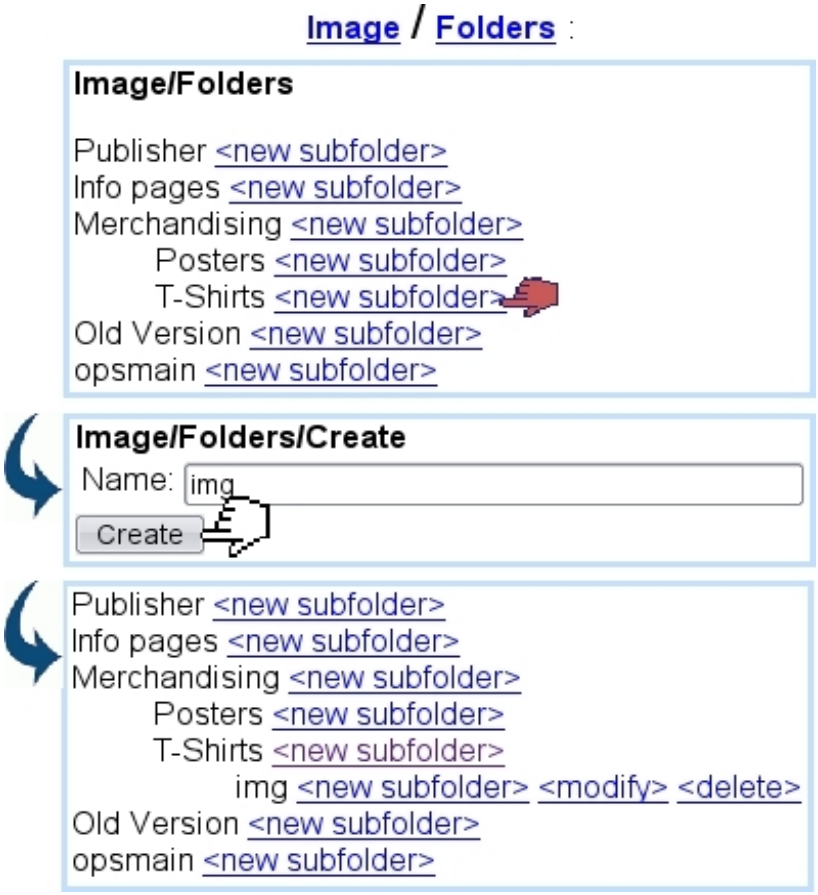


Figure 3.2: Creating a folder for images.

3.1.3 Managing images

The **Manage** link in the **Image** environment provide a tree-view of folders and image files, allowing to perform various actions such as deleting pictures (which can also be done through **Delete**), renaming them (Fig.3.3)...

3.1.4 Integrating images

In a Leaf Frame, images that have been uploaded through this means are then integrated either from the Choose image button (in the context of an Image variable) or by clicking the  (in the context of the Text Editor). A file and folder structure similar as shown

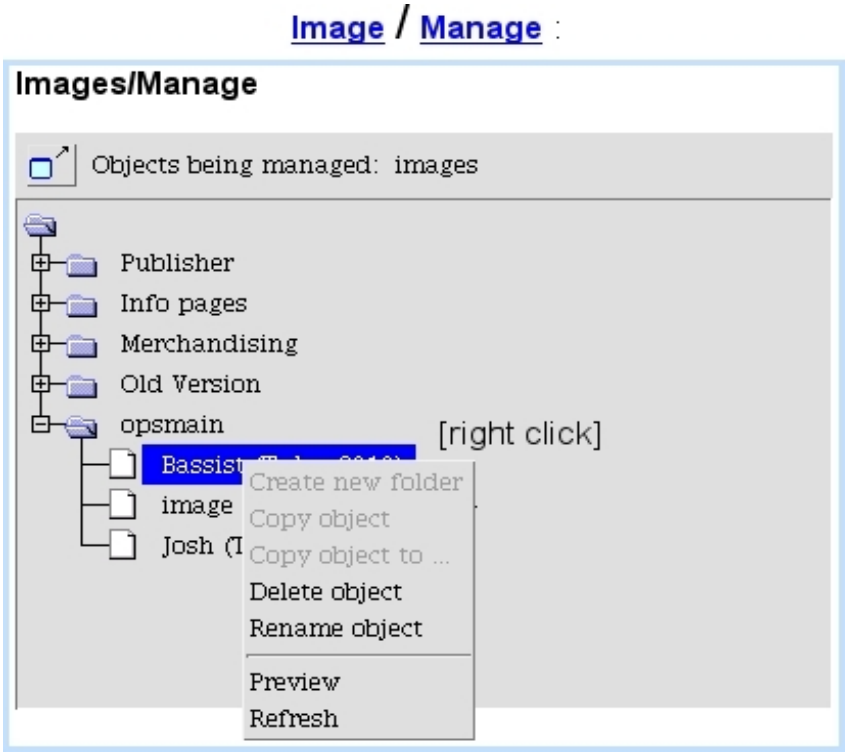


Figure 3.3: From here, all images and folders can be globally managed.

above will then be displayed (Fig.3.4).

If an image needs to be integrated in the contents of a Headline, such as a Banner, it is advised to upload the picture as a file (4.1.1).

To make image available in the context of an Image variable, leaf frame's contents will have to call `image.phs`, like it was done as an example in Code 3.

Table 6.4 in the Annex shows more options for system script *image.phs.*

3.2 Links

User group rights for Links to be handled are detailed at 1.3.6.

Link are that are created through Blue Chameleon Content Management System are afterwards available whenever a Choose link button is available, for instance when a leaf frame contains an **Link variable**.

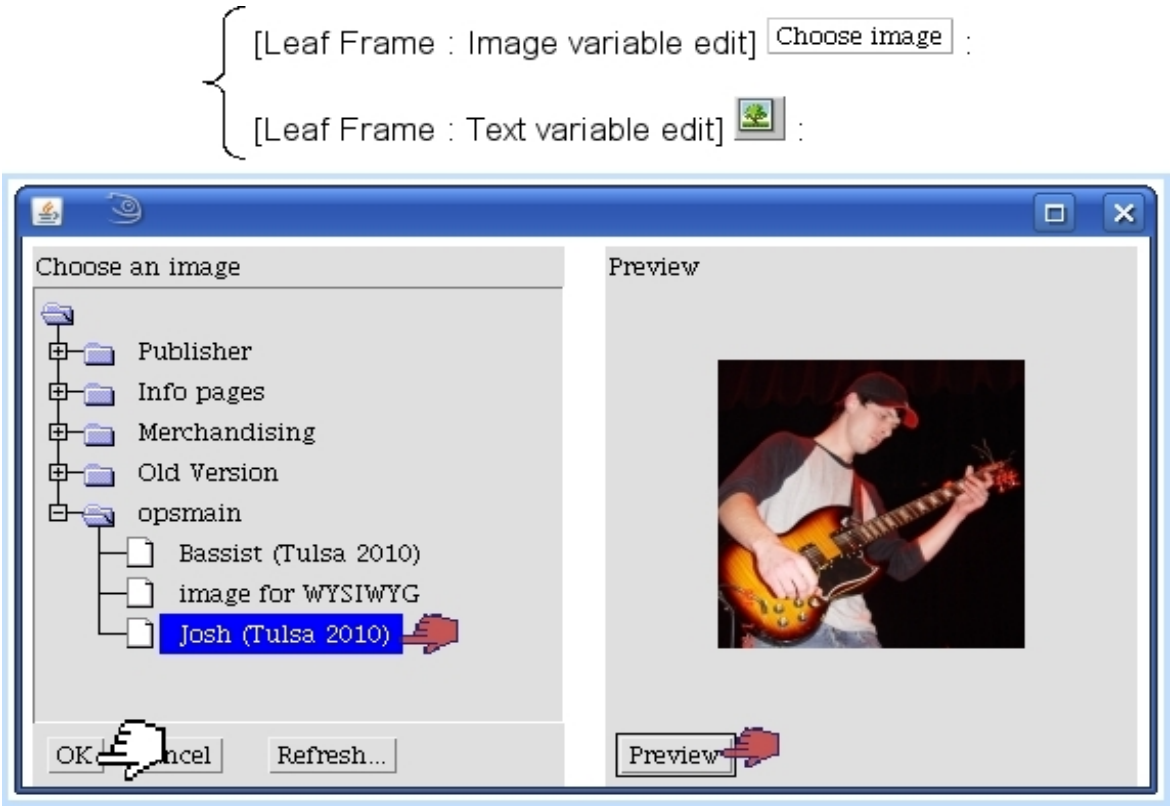


Figure 3.4: Integrating an uploaded image.

The creation of links is used for 'URL' links, i.e. that redirect to a `http` or `mailto` address ; otherwise, linking to already-existing object such as an headline or a download does not require a link to be created.

3.2.1 Creating a URL link

Fig.3.5 shows how a URL link is created :

- for an external page, the full `http://...` or `https://...` address must be indicated ;
- for a local page (see 2.7.2 on how, address-wise, pages are published), the full path from Publisher name (not included) to publication name has to be indicated ;
- for an email address to write to, the `mailto:...` is included.

There is also the option of making link available in any column of the Publisher.

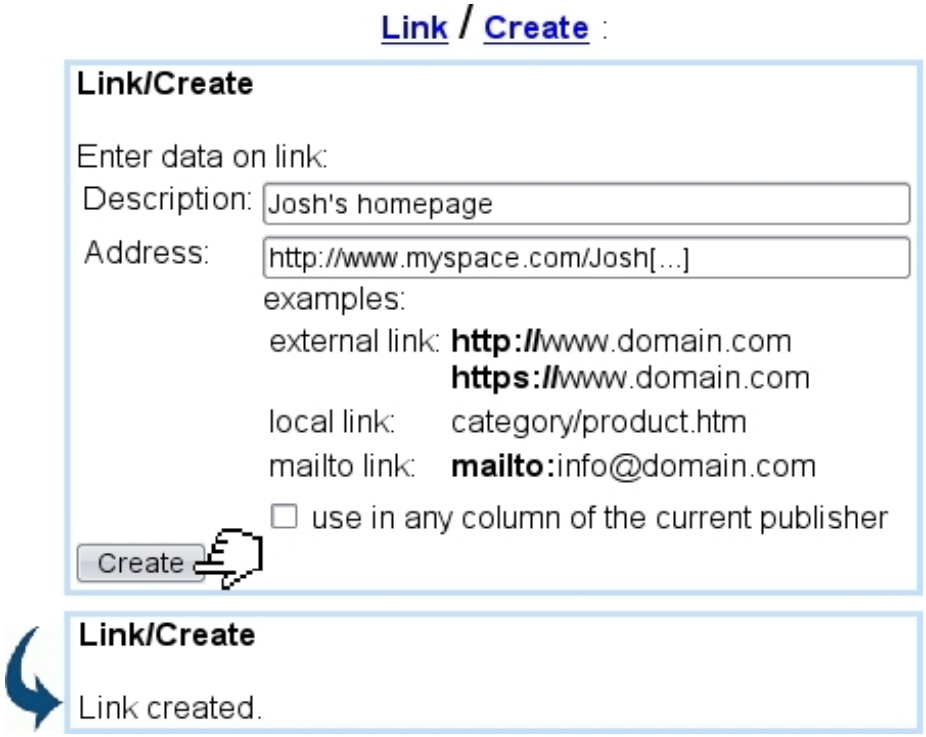


Figure 3.5: Creating a link.

3.2.2 Managing URL links

After having been created, the address and name for a link can be updated through **Link** / **Modify**.

A link can also be previewed (i.e. available as a click link to test whether it redirects rightly) or deleted through the eponymous links.

3.2.3 Integrating links

In a Leaf Frame, links that have been created through this means are then integrated either from the Choose link button (in the context of an Link variable) or by clicking the  icon (in the context of the Text Editor). A file and folder structure will then be displayed (Fig.3.6).

It is to note that the 'link type' menu as seen in that figure offers various options : `URL`, which is how links created as described above are integrated, but also Articles and Headlines, as well as books (4.3.2) and downloads (3.4) can be linked to. The example below shows how a link to the headline "Featured Artist of the Month" - without knowing its specific address - is selected for a link variable :

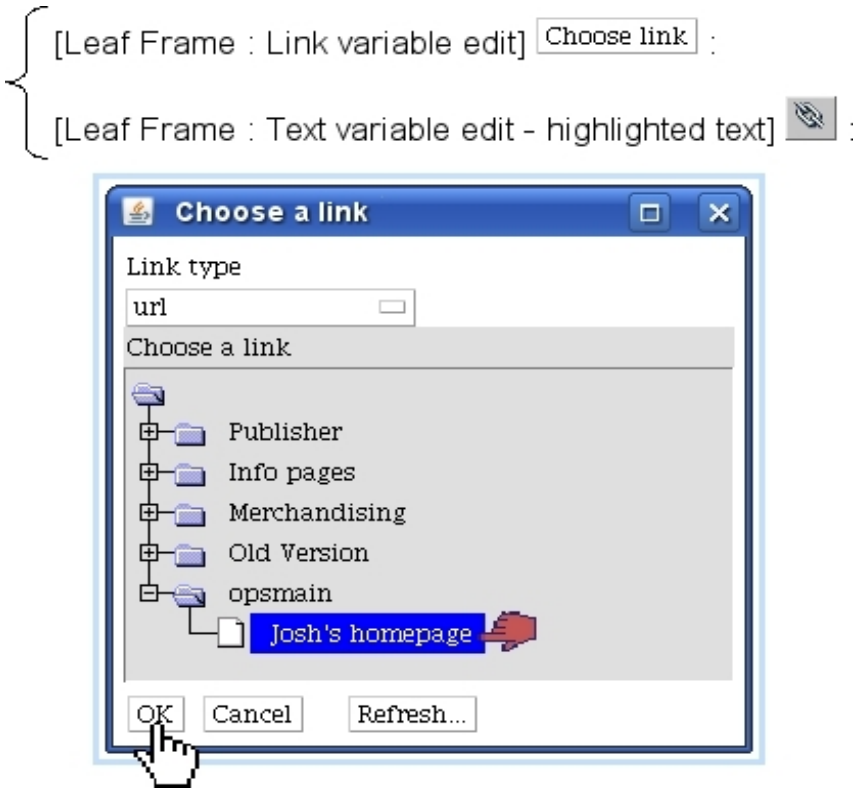
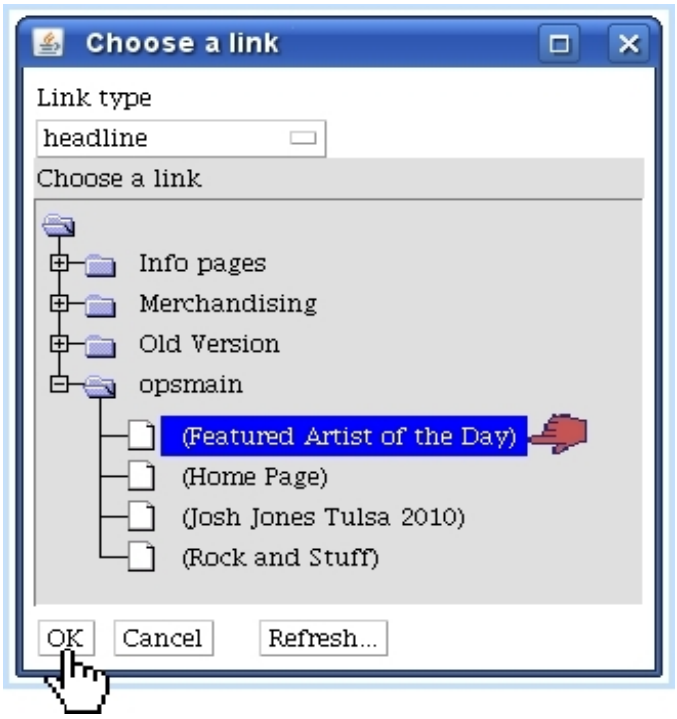


Figure 3.6: Putting a previously-defined URL link.



For link variables, if `_LinkVar` is the name of the link variable, the leaf frame contents will have then include the following call to make link available :

Headline models / **Modify** / **a leaf frame model** :

```
 (...)
«INCLUDE href.phs;_Link="_LinkVar"»
(...)
```

Links can also be enriched with styles, for instance triggering pop-ups (3.3.3).

Table 6.4 in the Annex shows many more options for system script `href.phs`.

3.3 Scripts

User group rights for managing Scripts are detailed at 1.3.18.

Blue Chameleon Content Management System offers the possibility for `OSL`-based code to be executed for certain objects like frames, links... This might be useful when developing a Front-Office.

3.3.1 Creating a script - Management

As featured in Fig.3.7, creation of a script (apart from name and whether it will be used in any column of the Publisher) requires a type, amongst those :

- Frame Edit/View : to be included in leaf frames using a 'script' variable (see below) ;
- Link style : for leaf frames with 'link' variable, providing a specific style for the link (see below) ;
- Menu, dedicated to menu elements (3.5).

After, scripts can be tested through **Run**, modified via the eponymous link (allowing to either **Modify description** or **Edit script**).

Figure 3.7: Creating a script to be used in a leaf frame.

3.3.2 Integrating a frame script

A leaf frame that handles scripts must have the following contents :

Headline models / Modify / A script frame :

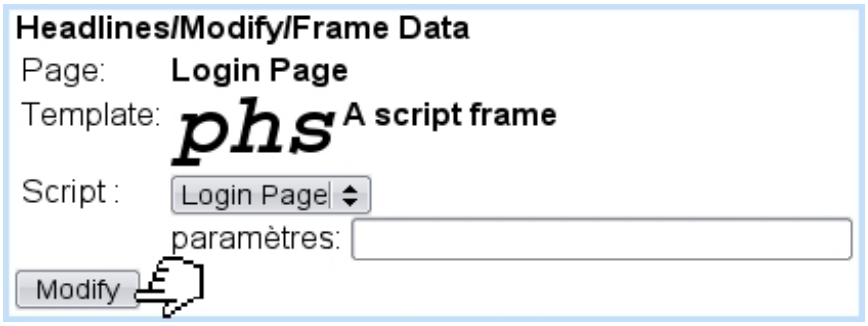
```
«INCLUDE scriptfile.phs»
```

This system script handles calls to scripts. As for the 'script' variable, it uses identifier '5' :

Headline models / Variables / A script frame :

```
Script;Script;5;
```

As a result, during editing of a headline containing this frame template, a frame script will be chosen :



It is possible to pass parameters to the script thanks to the eponymous field, in the following way :

```
_Param1=1;_Param2=yes;...
```

3.3.3 Integrating a link style script

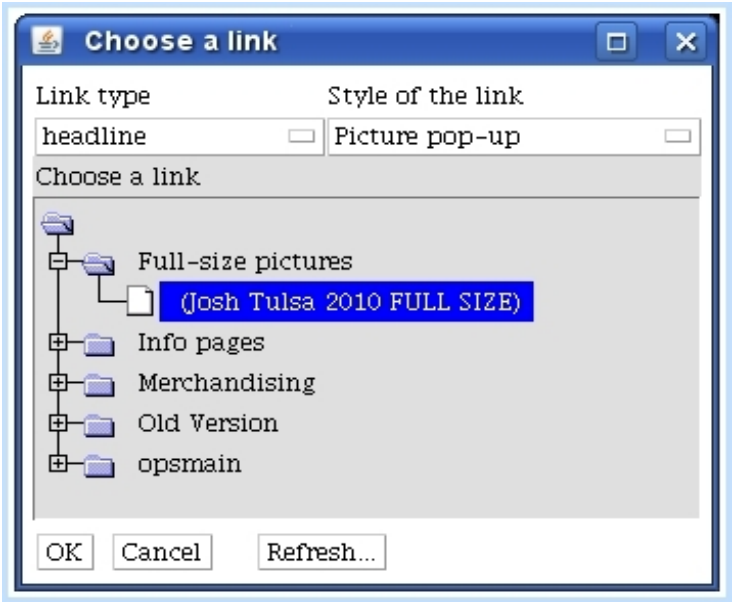
This type of script is useful when a clicked link must trigger a specific action, such as a pop-up of defined pre-dimensions.

For that matter, a script 'Picture pop-up' is created/defined as follows :

Scripts / Create :
...

```
HREF="#"
onClick="window.open('«URL»','PopUpWindow','toolbar=no,location=no,menubar=no,
status=no,scrollbars=no,resizable=no,directories=no,copyhistory=no,
width=1280,height=984')"
```

On a frame containing a link (3.2.3), we then make that link to point to a headline (for instance, a headline containing a full-sized picture), using style 'Picture pop-up' :



Thus, upon click on that link, a pop-up sized 1280x984, showing the headline "Josh Tulsa 2010 FULL SIZE", will appear.

3.4 Downloads

User group rights for handling Downloads are detailed at 1.3.12.

Downloads are created through Blue Chameleon Content Management System serve as to provide on published pages a link to download any kind of file. In difference with uploaded files (4.1.1) that would be linked to with a self-made link, downloads allow, with variables of the 'download' type, to provide an easier way to create a download-link.

3.4.1 Creating a download

The process is shown at Fig.3.8 ; apart from the name under which the uploaded file will appear ('Description:'), an optional field 'Rename filename:' serves as to automatically rename the file when it will downloaded on the resulting page.

The type of file (amongst a wide variety of choices as set at the Publisher settings' file types, 1.5.1) is also chosen, as well as in which folder to put it in.

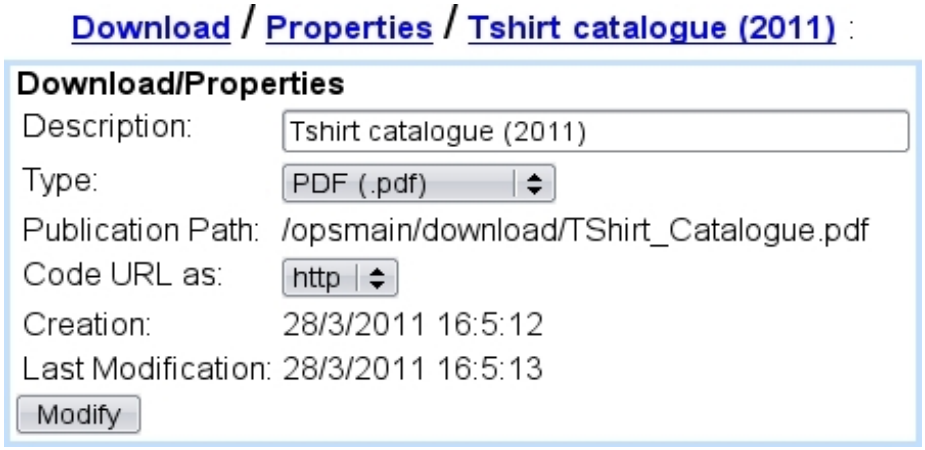
Figure 3.8: Uploading a file that will be used for downloading purposes.

If the 'automatically publish' checkbox had not been ticked during creation, **Download** provides a **Publish** link that allows to do it anytime.

3.4.2 Managing downloads

After creation, the **Download** / **Modify** link allows to change the file it links to. This is useful when a updated document (for instance in a weekly or monthly way) has to be uploaded regularly : nothing else has to be done than uploading the file again. It also offers information on publication path and creation, last modification dates.

The **Properties** link (apart from the same information as given by the Modify page), offers to change the type of file (provided sufficient group rights) through a menu as well as to choose either a `http` or `https` coding for the URL :



A **Preview** link also allows to test any created download, and it is possible to **Manage** them through a file-folder tree structure similar to Fig.3.3. Deletion of a download is also permitted through the eponymous link.

For secure downloads to function, they must be enabled at the Publisher Settings Page (1.5).

3.4.3 Integrating downloads

Downloads can be integrated on a leaf frame simply through a 'link' variable : indeed, the 'link type' menu (3.2.3) as provided when integrating a link provides a 'download' option, allowing to select a download from browsing the usual tree structure.

A download can also be integrated through a 'download' variable proper, as identified by 10, for instance as in

```
_Dwnl;Download;10
```

Fig.3.9 shows how the download variable is filled.

Leaf frame contents must then include the following call to make download available :

Figure 3.9: Integrating a download, with a 'download' variable.

Headline models / **Modify** / **a leaf frame model** :

```
 (...)
«INCLUDE download.phs;_Download="_Dwnl"»(...)
```

This will generate the `<A HREF="...">`, `</A>` tags around the download description as given by the user.

3.5 Menus

User group rights for handling Menus and Menu models are detailed at 1.3.27.

A "menu" as defined in Blue Chameleon Content Management System is yet another object that can be put in a leaf frame ; menus generated this way are fully customizable.

A menu is based upon a menu model which uses itself a menu-type script.

3.5.1 Menu scripts

A script (3.3) for instance called 'Top Menu script', of type 'menu', could be defined in the way shown at Code 4.

In this simple example, menu items simply link to an element and do not contain children menu items (3.5.4).

Using the system table `OPSMENUITEM` which stores menu elements (added by user when filling the menu (3.5.3.1)), this example menu script does the following :

Code 4 An example of menu script, outputting menu items :

Scripts / **Create** :

...

```
«VAR NEW MenuLinkId»
«VAR NEW MenuLinkType»
«VAR NEW MenuLinkStyle»

<TABLE cellspacing=2 cellpadding=2>
 <TR>
  «SQLOUTPUT»

   «LET MenuLinkId=iLinkId»
   «INCLUDE LinkTypeConversion.phs;_Way="int2char";_IntVar="iLinkType";
   _CharVar="MenuLinkType"»
   «LET MenuLinkStyle=iLinkStyle»

   <TD class="«#SQLWWWHOME#»/css/MyClass.css">
    <A HREF="«INCLUDE href.phs;_Link="MenuLink";JustURL»">«azItemName»</A>
   </TD>

  «/SQLOUTPUT SELECT * FROM «#SQLWPA#» OPSMENUITEM WHERE iMenuId=«_MenuId» AND
(iStatus=1 OR iStatus=3) AND ORDER BY iIndex»
 </TR>
</TABLE>
```

- on the relevant menu (as identified by `_MenuId`), information is fetched on all menu items that are not hidden (test on `iStatus`) ;
- for each of those particular menu items, in a cell formatting (using a custom `css`), the corresponding Url is retrieved and put as a link around menu item's name.

3.5.2 Menu models

With at least a menu script defined, a menu model can be added as featured in Fig.3.10.

There :

- as usual, are given name and whether use in all publisher is allowed ;
- a 'Rights' vector, which must be as long as there are items in the menu (3.5.3.1) ; if the n-th element of the vector is '1', the n-th item of the menu will be allowed to have a picture as a link, '0' if not ;
- methods for creating, editing and viewing : usually, the same menu script (here, 'Top Menu script') as described above is taken for the three.

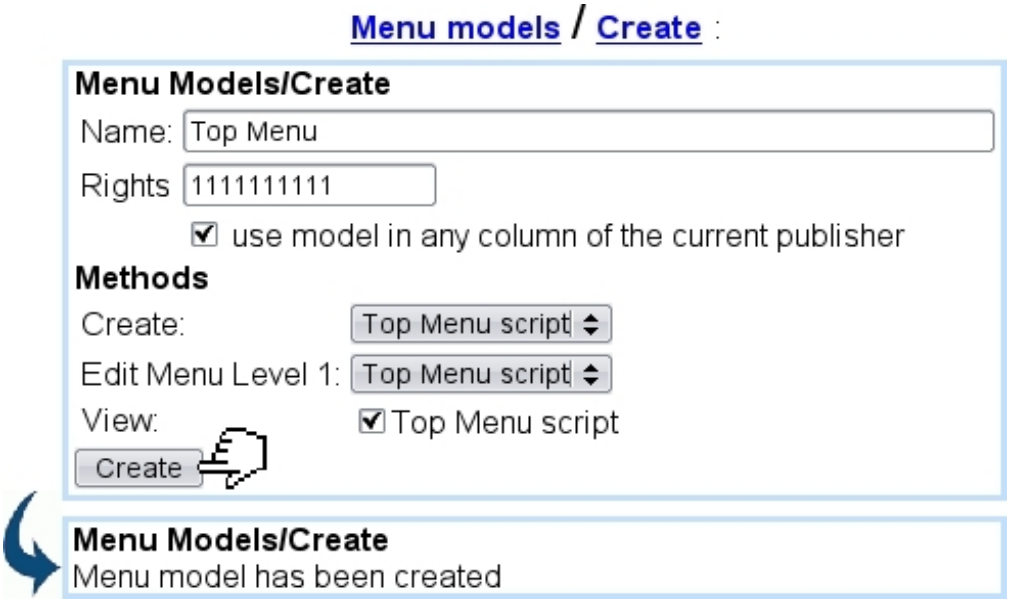


Figure 3.10: Creating a menu model.

3.5.3 Creation of a menu

With a menu model defined, a menu is defined as described in Fig.3.11.

After creation, it is to note that a menu can be updated (i.e. its name changed) through **Menu** / **Properties**.

Now, once created, menu is to be filled.

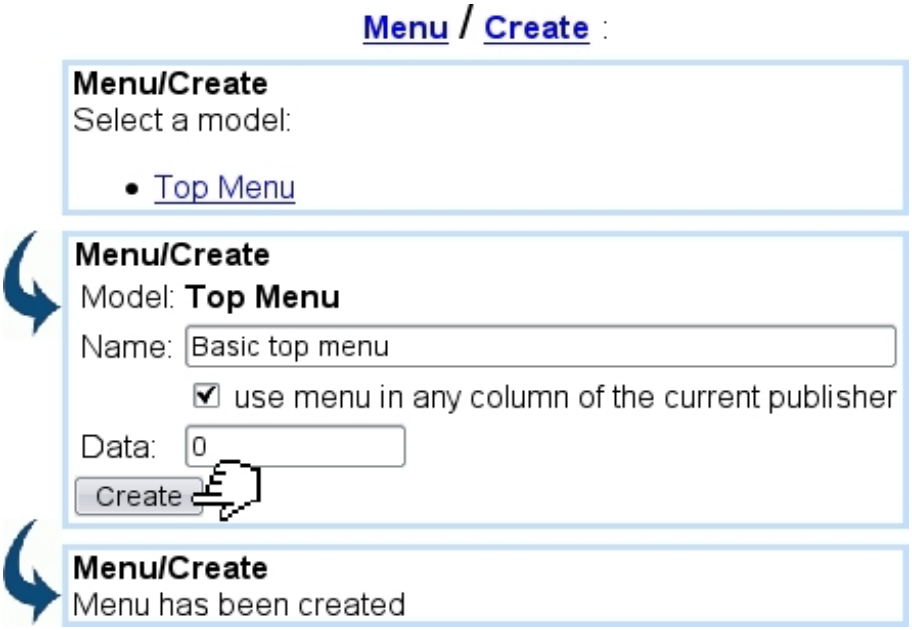
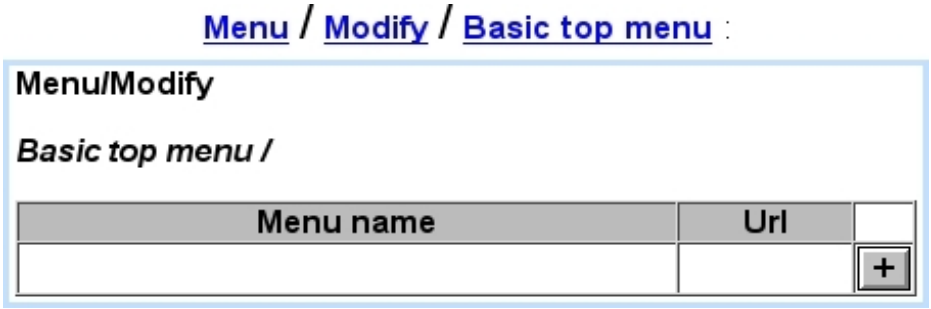


Figure 3.11: Creating a menu "Basic top menu" based upon model 'Top Menu'.

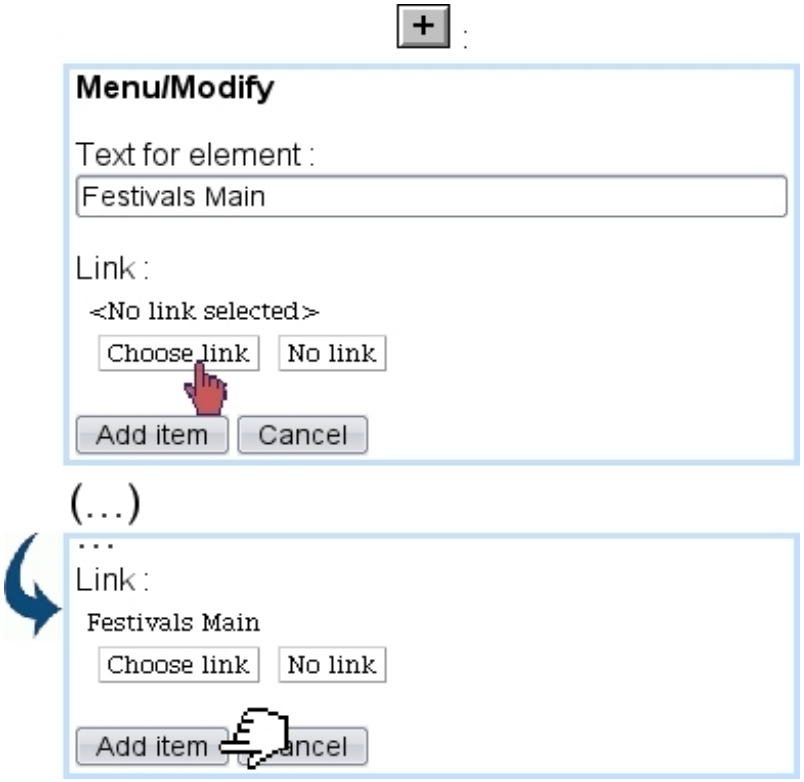
3.5.3.1 Filling of a menu

For menus to function properly, linked pages or objects must be published (2.7.1).

This task is achieved through Menu / Modify, where a table featuring 'Menu name', 'Url', and one (and then several) icon(s) are proposed :



Clicking then on the + icon leads to a screen where a name for the first menu element is chosen, as well as a link (which can be of any type (3.2.3) and style (3.3.3)), or none at all ; in this example, link is chosen as Headline "Festivals Main" :



Upon validation on Add item, menu item is finally added, and now a whole range of icons is available :

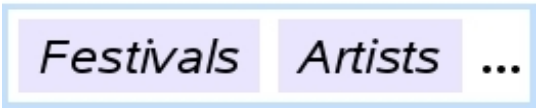
For each of menu items, their functions are :

- : hides (by greying it) / displays the menu item ;
- : edits the menu item's text and link ;
- : views the output of the menu script, applied to that menu item. In the example of Code 4 the output is identical for all menu items, as they do not have children ;
- , : adds another (sibling) menu element before or after this menu item ;
- , : for menus with more than one item, places this menu item before the previous one (if not the first in the menu) or after the next one (if not the last in the menu) ;
- : appends a child menu item to this menu item (see 3.5.4) ;
- : removes this menu item from the menu.

3.5.3.2 Menu result and integration

With a simple menu script as defined as in Code 4, the output of the "Basic top menu" menu gives :

[Any "Basic top menu" item] :



Each of the menu elements as rendered here is clickable, redirecting to its assigned Url.

Now, for the frame that will accept a menu, its contents must be written in the following way :

Headline models / **Modify** / **Menu frame** :

```
«INCLUDE showmenu.phs;_Menu=_Menu»
```

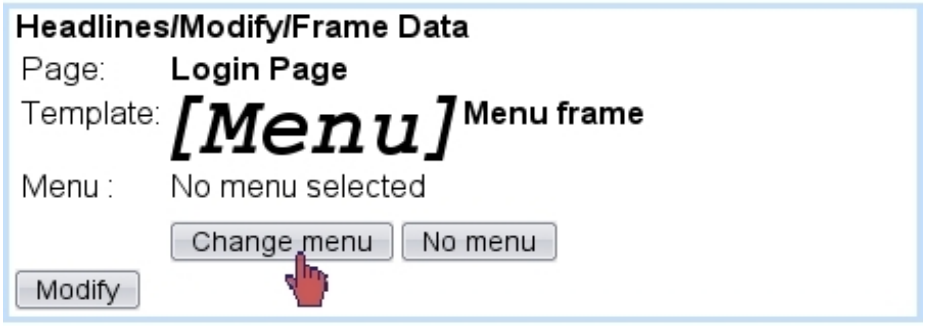
This system script handles menus, with the variable `_Menu` set as follows,

Headline models / **Variables** / **A script frame** :

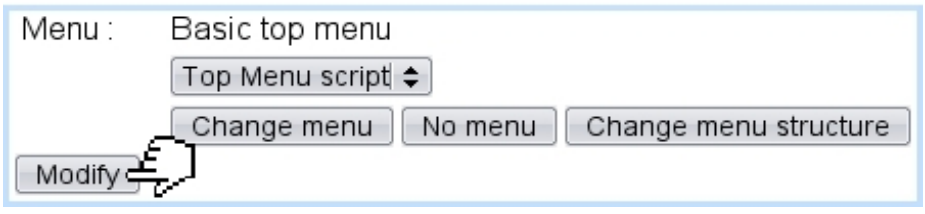
```
_Menu;Menu;12;
```

As a result, when editing this frame, a Change menu will allow to choose a menu, here "Basic top menu" :

[Editing frame "Menu frame"] :



[After menu is selected] :



After selection of the menu, button Change menu structure redirects to a page where menu items are managed, similarly as in 3.5.3.1. The No menu button allows to remove the selected menu.

3.5.4 More complex menus

The above dealt with a simple menu, where each item corresponds to one link. Now, Blue Chameleon Content Management System allows more advanced menus to be implemented, where a menu item yields other menu items.

3.5.4.1 A menu script that handles children menu items

To do so, Code 4 must be altered and enriched : an example of what could be done is given at Code 5.

Basically, there are now two `SQLOUTPUT`s, one for constructing the menu horizontally like before, and the other, embedded, to construct children menus vertically if they exist :

- the first selection comes now with a `iParentItemId=0` clause, so that only "root" menu items are selected ;
- inside the loop, for a menu item, a test is made on how many children it has ; if it has not any, menu item is displayed as before ;
- ...if it has children, menu item comes with `javascript` functions aimed at showing/hiding a `DIV`, where the children of this menu item are constructed (now vertically) with a similar mechanism, but using menu item's identifier `iItemId` for `iParentItemId`'s value ;
- this inner loop is exited, and next horizontal menu item is processed.

3.5.4.2 A menu with children menu items

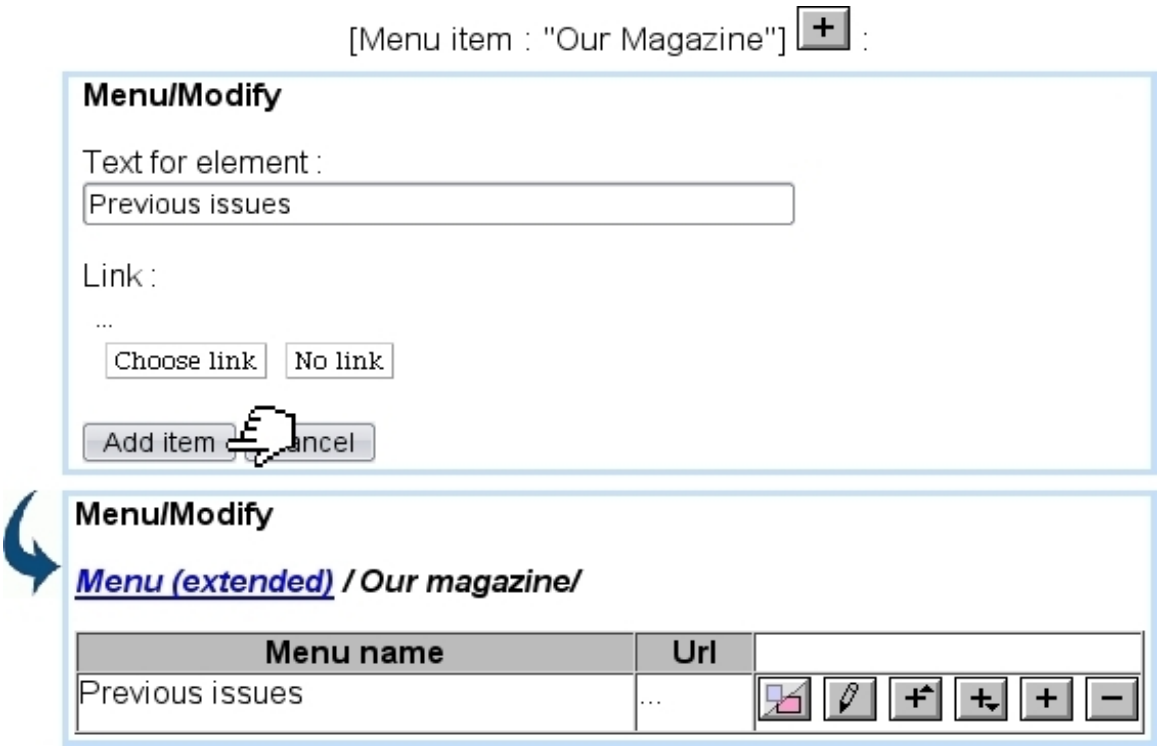
Such a menu (called in the following example "Menu (extended)") is constructed by using the  available for each menu item (3.5.3.1) ; child menu item is then created as usual with a name and link. In this example, we aim to create, for menu item "Our Magazine", a child menu item :

Code 5 A menu script which handles menu items that could have children :

Scripts / Create :

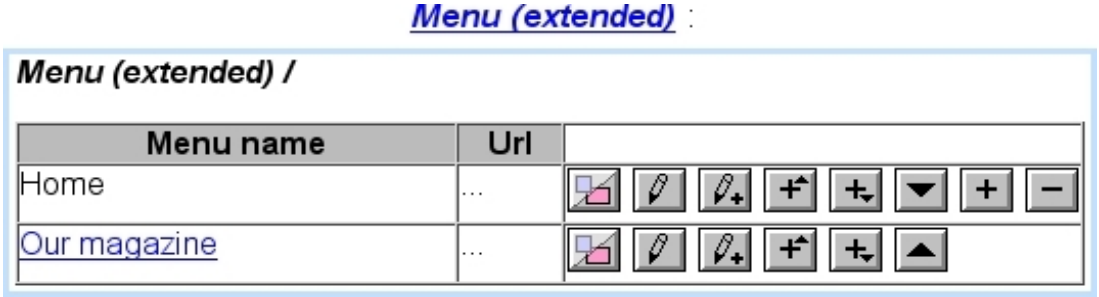
...

```
«VAR NEW MenuLinkId»
«VAR NEW MenuLinkType»
«VAR NEW MenuLinkStyle»

<TABLE cellspacing=2 cellpadding=2>
 <TR>
  «SQLOUTPUT»

   «LET MenuLinkId=iLinkId»
   «INCLUDE LinkTypeConversion.phs;_Way="int2char";_IntVar="iLinkType";
   _CharVar="MenuLinkType"»
   «LET MenuLinkStyle=iLinkStyle»

   «SQLEXEC INT _nbChildren=SELECT COUNT(*) FROM «#SQLWPA#»OPSMENUITEM WHERE
iMenuId=«_MenuID» AND iParentItemId=«iItemId»

   «IF _nbChildren==0»
    <TD class="«#SQLWWWHOME#»/css/MyClass.css">
     <A HREF="«INCLUDE href.phs;_Link="MenuLink";JustURL»">«azItemName»</A>
    </TD>
   «ELSE»
    <TD onMouseOver=...  onMouseOut=...  class="«#SQLWWWHOME#»/css/MyClass.css">
    «azItemName»</A> 
     <DIV id=...>  «*Output child menu vertically*»
      <TABLE>
       «SQLOUTPUT»
        «LET MenuLinkId=iLinkId»
        «INCLUDE LinkTypeConversion.phs;_Way="int2char";_IntVar="iLinkType";
        _CharVar="MenuLinkType"»
        «LET MenuLinkStyle=iLinkStyle»
        <TR><TD><A HREF="«INCLUDE href.phs;_Link="MenuLink";JustURL»">
        «azItemName»</A></TD></TR>
       «/SQLOUTPUT SELECT * FROM «#SQLWPA#»OPSMENUITEM WHERE iMenuId=«_MenuId»
        AND (iStatus=1 OR iStatus=3) AND iParentItemId=«iItemId»
        ORDER BY iIndex»
      </TABLE>
     </DIV>
    </TD>
   «ENDIF»

  «/SQLOUTPUT SELECT * FROM «#SQLWPA#»OPSMENUITEM WHERE iMenuId=«_MenuId» AND
(iStatus=1 OR iStatus=3) AND iParentItemId=0 AND ORDER BY iIndex»
 </TR>
<TABLE>
```



A new view *Menu (extended) / Our magazine* is then provided, showing we are now inside menu item "Our magazine". There, menu items can be managed in a similar way as for the root menu. We can add thus several menu items for this sub-menu :

This sub-menu is exited by clicking on the root :



The sub-menu in question can be managed anytime through the **Our magazine** link.

It is to note that, in order to delete a menu item that has children, those have to be deleted first.

3.6 Forms

User group rights for handling Forms and Form models are detailed at 1.3.29.

3.6.1 Form models

As featured in Fig.3.12, the creation of a form model, apart from a name, entails the following :

- the script type it will use, either Blue Chameleon (`OSL`) or `PERL` ;
- which action it will perform (apart from none, either 'Send E-mail' or 'Record to file' ;
- how labels will be aligned :
  - normal (no alignment) ;
  - two lines (field, menu or radiobutton is below the label) ;
  - labels aligned left (or right), including colons ;
  - labels aligned left (or right), without colons ;
  - labels aligned left, including colons aligned right.
- how buttons will be aligned, left, center or right.

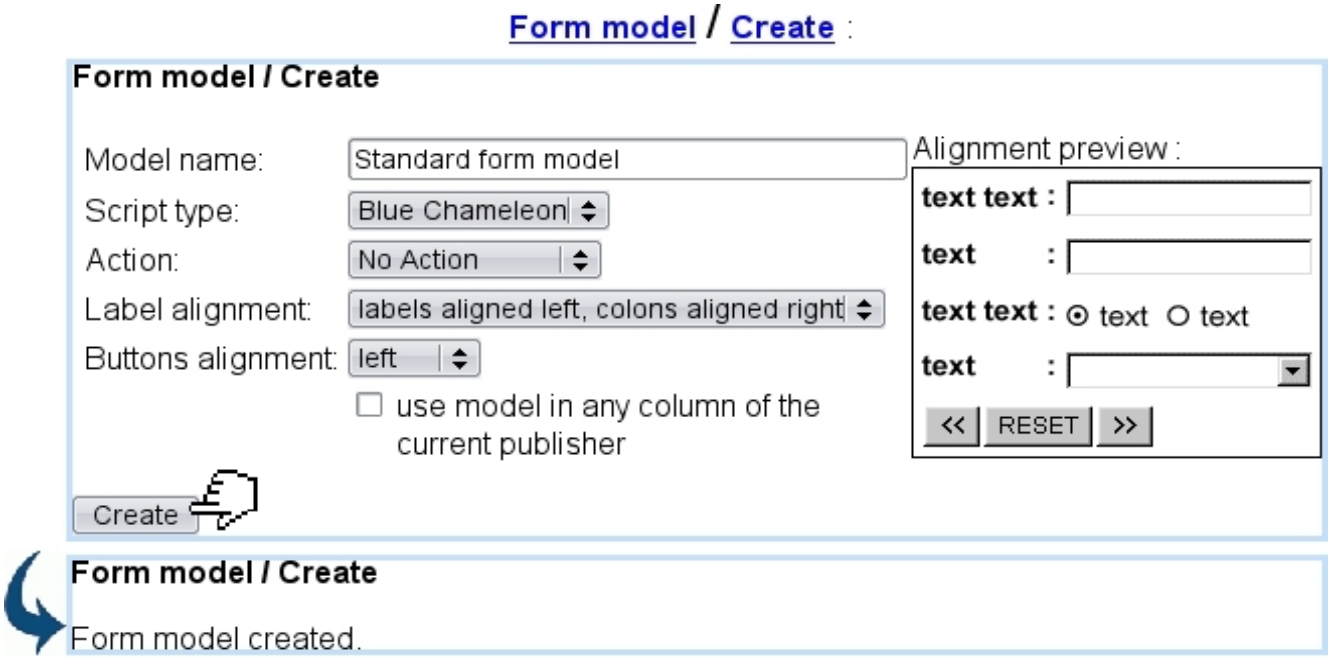


Figure 3.12: Creating a form model.

3.6.1.1 Form model management - parameters

Once created, a form model can be modified, offering the same options as described above, plus now the defining of parameters, as Fig.3.13 shows. A parameter name and value must be given and it can also be flagged as read-only.

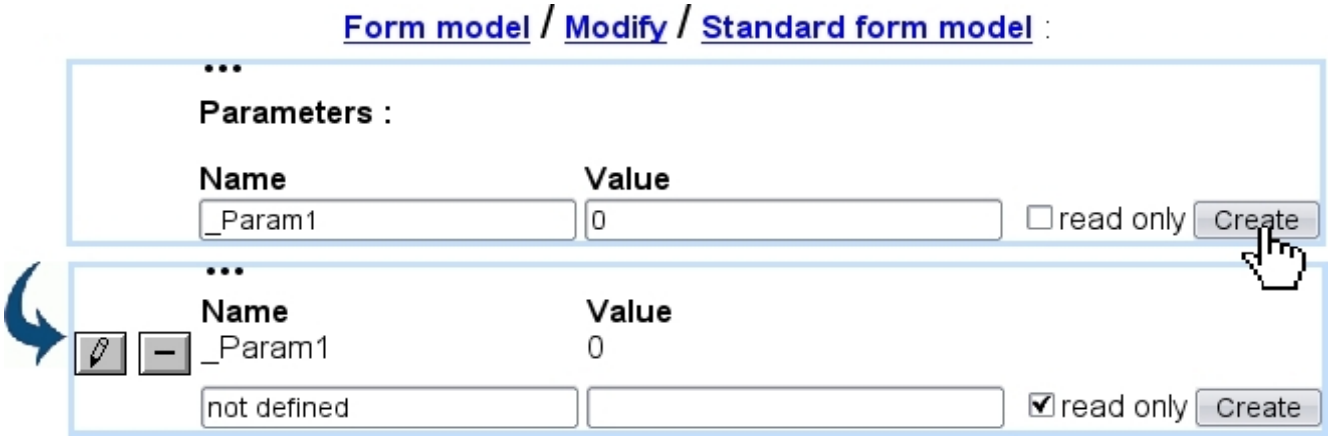


Figure 3.13: Adding parameters to a form model.

As many parameters as needed can be created, and each has its set of  ,  icon allowing to respectively edit or delete it.

3.6.2 Defining forms

Once form models as described above exist, forms themselves can be created as featured at Fig.3.6.2.

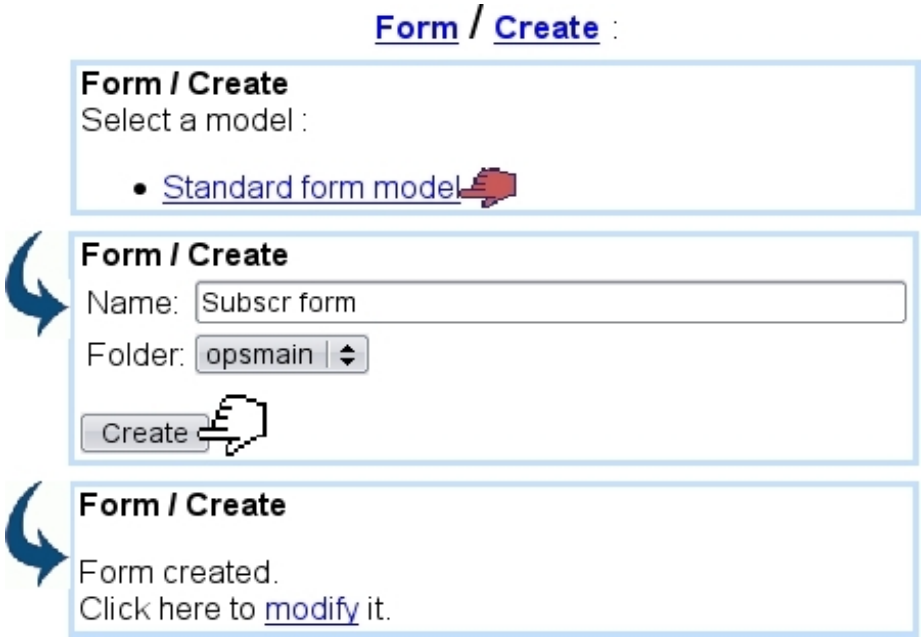
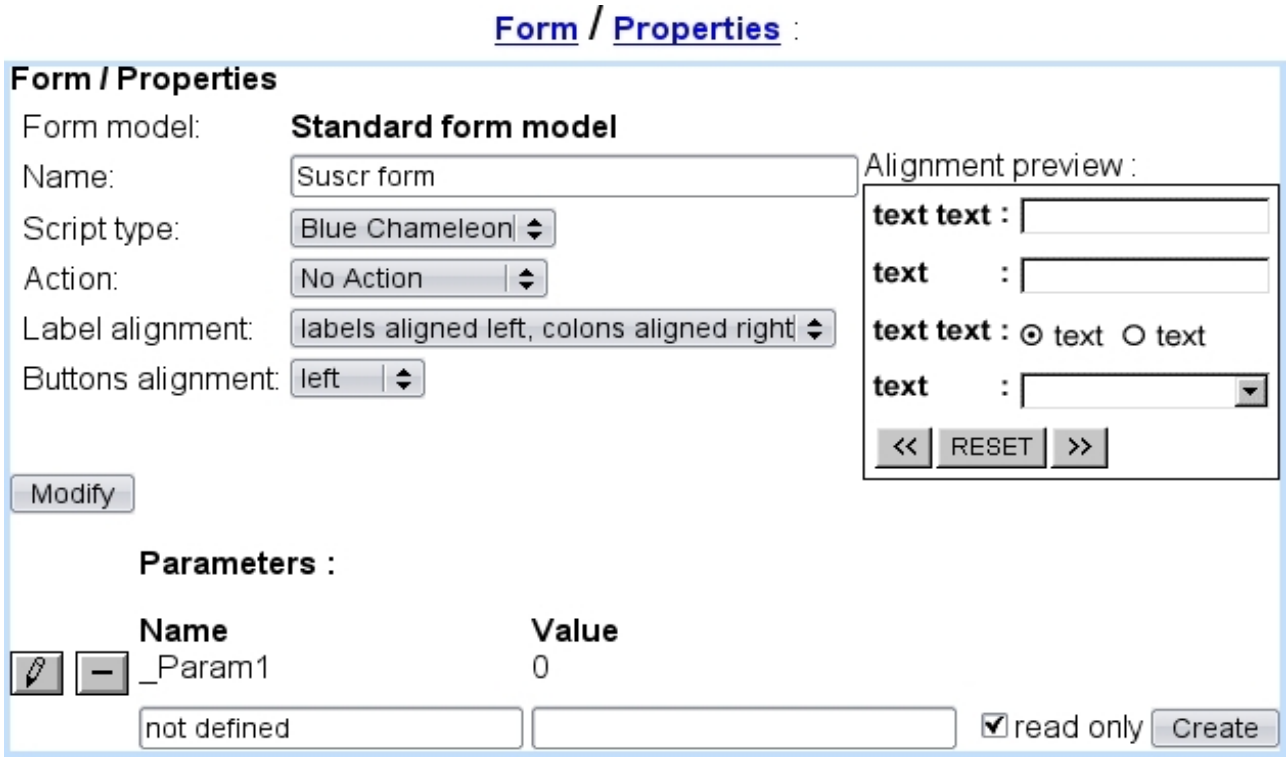


Figure 3.14: Creating a form based on the example model.

3.6.2.1 Updating a form

A form inherits the properties of its referent model (script type, alignment, parameters...) ; nonetheless, they can updated along form's name as follows :



3.6.3 Composing a form

In Blue Chameleon, composing a form is done using a similar interface as the one seen for headlines.

Going for modifying a form displays the following :

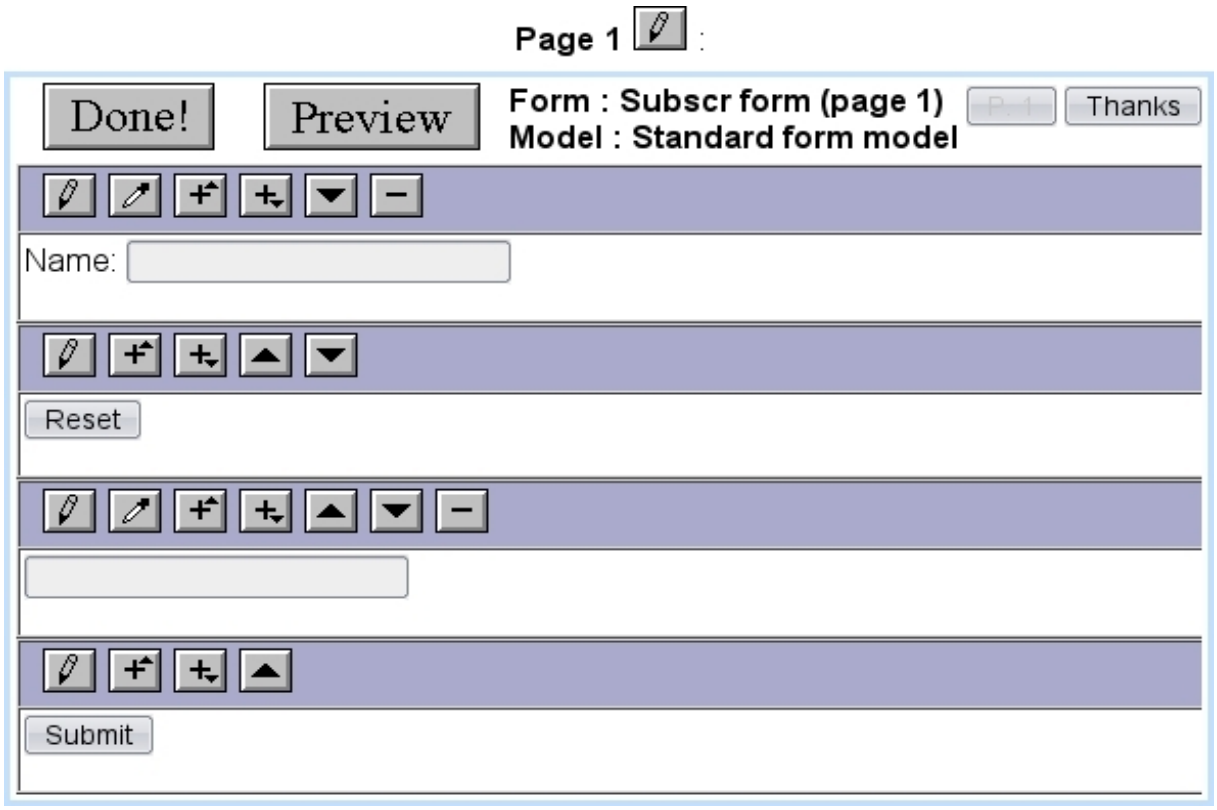
A list of "design pages" for this form is featured there, each of them accompanied with icons aimed at editing this page () and adding a new page before, after ( , ).

A "Thank you" item (**which is not a form page**) serves as to provide, if wanted, a confirmation message (3.6.3.4) after the submission of the form.

If form contains several pages (3.6.3.6), a  icon is available to remove a page.

3.6.3.1 Editing a form page

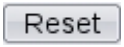
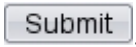
Upon click of the  , **for a form that has only page**, the following is shown :



This headline-editing-style environment features, along with the usual  and  buttons, the name of the current form as well as the current page, in a greyed-out style,  . If form contains several pages (3.6.3.6), as many buttons as pages would be available, each clickable to switch pages.

Whichever the number of pages is, a  button is always there to switch to the composing of the post-submission message (3.6.3.4).

It can be seen that, at first, four elements are featured by default :

- two fields, which can be altered or removed ;
- two buttons,  and  , which can only be altered ; they function as non-removable buttons that respectively cancel the form and submit it.

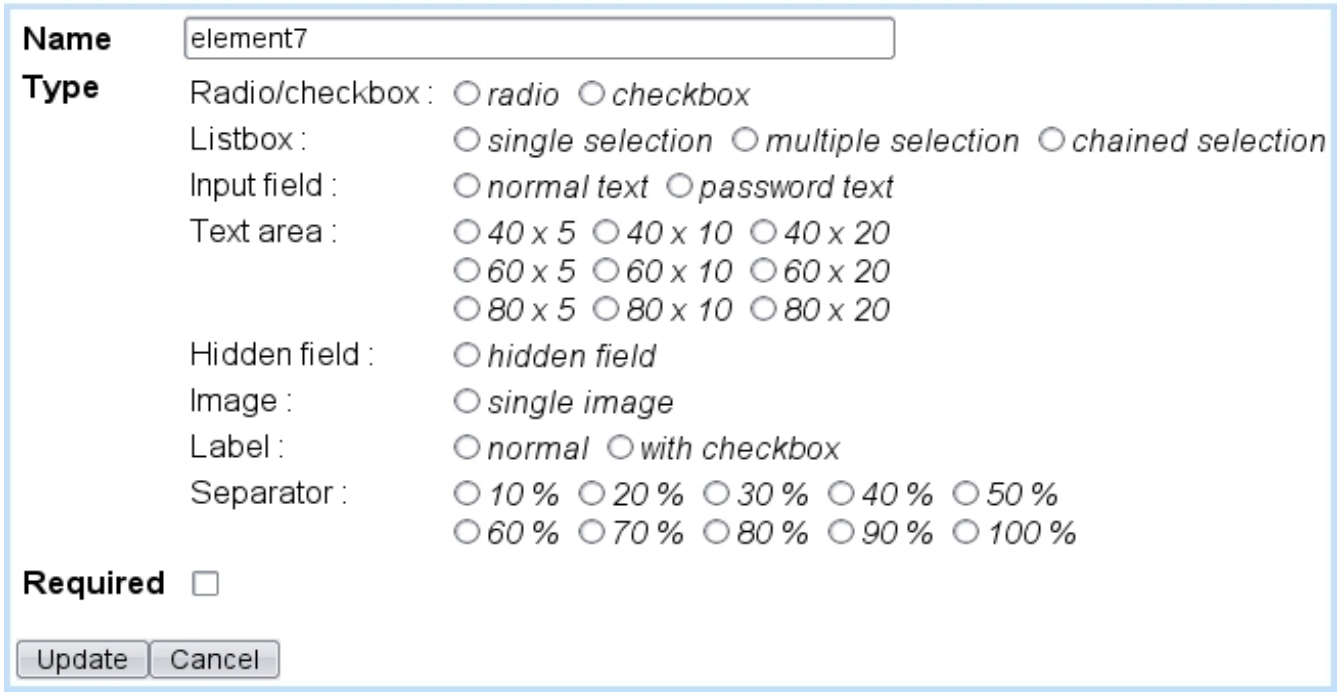
For a given form page element, icons allow to perform actions ; according to the function of the element, and/or its placing, they might not be available :

-  : allows to edit this element (3.6.3.3) ;
-  : allows to edit the type : checkbox, button, text input... of this element (3.6.3.2) [not available for the default Reset and Submit buttons] ;
-  ,  : allow to add a new form page element before, after this element ;
-  ,  : allow to place the element before the previous one (if not on the top) or after the next one (if not on the bottom) ;
-  : allows to remove this element (not available for the two default buttons).

3.6.3.2 Form page element types

A large choice of types of form elements is proposed when clicking on the  icon for an element :

 :



Types include the following :

- radiobutton ;
- checkbox ;

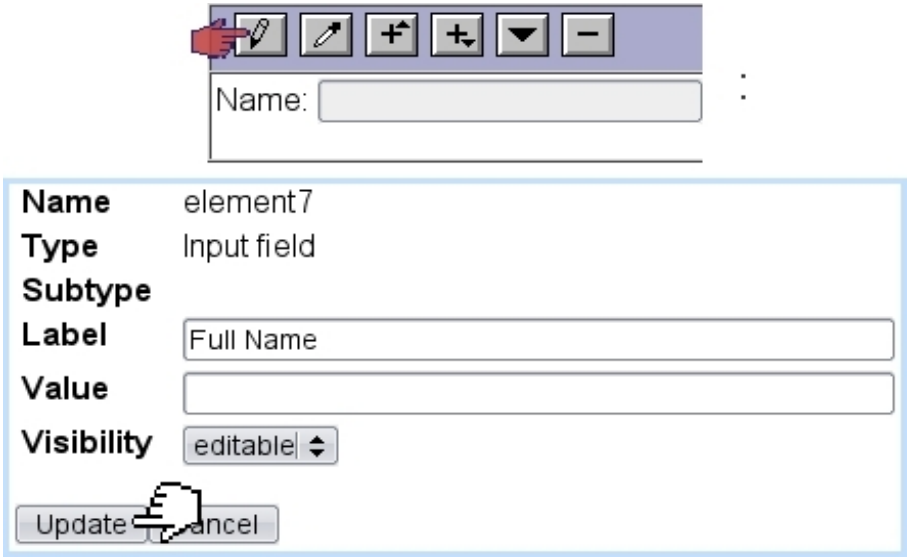
- input text ;
- text area ;
- a hidden field ;
- an image ;
- a label ;
- a separator.

Also, it is possible to flag the element as 'Required' for the submission of the form and alter the element's name.

It is to note that buttons are not featured, as they perform actions such redirecting to a page, submitting a form... that are reserved for the default buttons already provided by Blue Chameleon Content Management System.

3.6.3.3 Editing a form page element

The illustration below shows how, for instance, an input field can be edited :



Here, label and value can be edited, as well as the given visibility, chosen amongst 'hidden', 'visible' and editable. As for the 'Name', it can be edited while modifying the type of the element (see next).

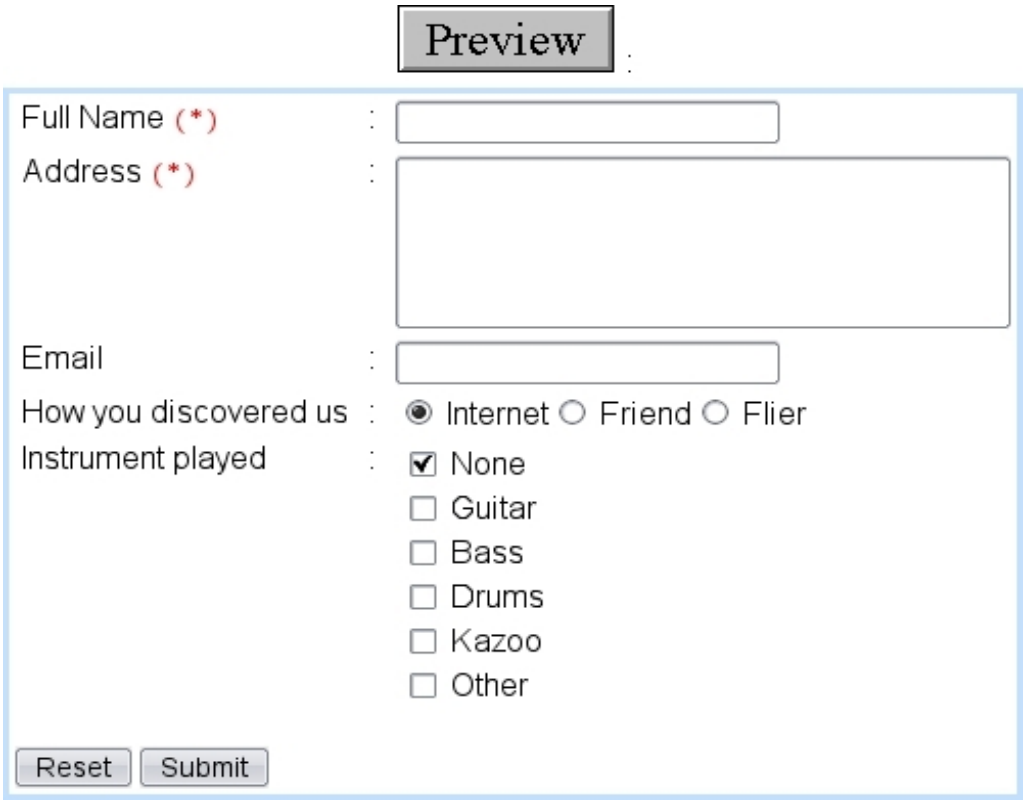
The full description of how form elements can be edited is available in Annex (6.4).

3.6.3.4 After form is submitted

The "Thank you" item as seen on the *Form Modify Page*, or the Thanks button as seen on the top when editing a form, permit to create a small message displayed after form has been submitted.

3.6.3.5 A form result

As several inputs are added, form is saved by the Done! button. Here is an example of the preview of a form :

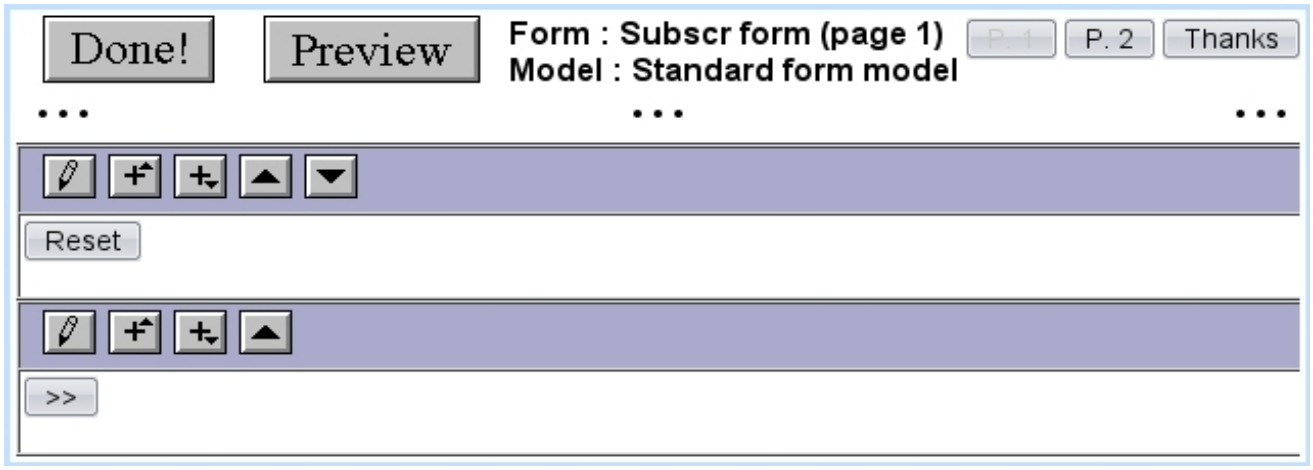


3.6.3.6 A form with several pages

The previous described the example of a form that contained only one page, therefore with a set of cancel and submit buttons already featured at the bottom of the page.

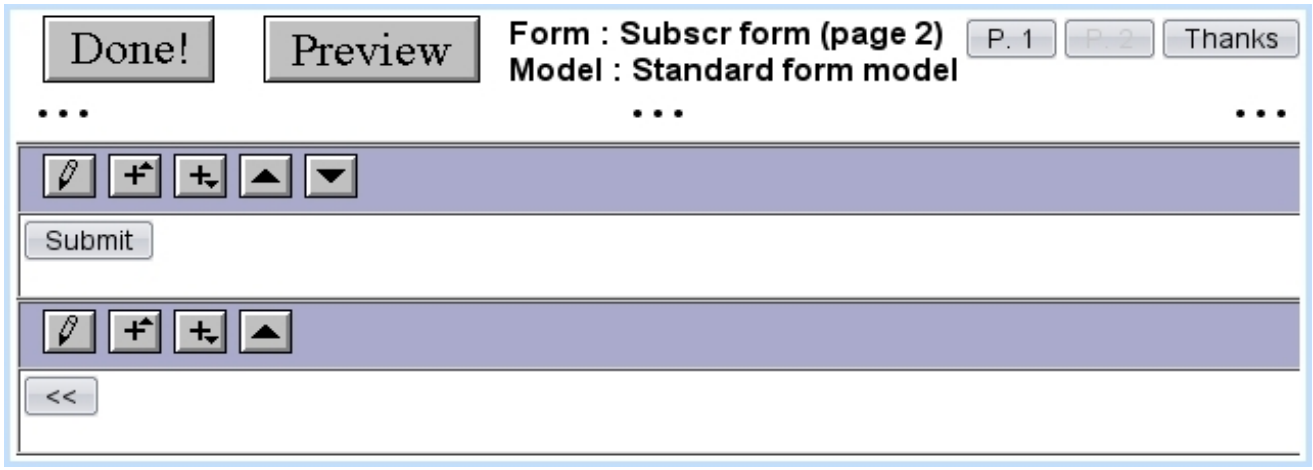
Now, need may rise of a form - with usually a large quantity of information - that features several steps (for instance, each terminated by some Next button) until final submission is done. Blue Chameleon Content Management System allows to do this, by adding as many steps ("pages") as needed, thanks to the , as seen on the *Modify Form Page* (3.6.3).

For instance, if a second page is created, the edit for Page 1 will now look like this :



It can be seen that a P. 2 button is now available to edit the second page and that the Submit button has disappeared, replaced by a >> aimed at displaying the next page on the resulting form. This button can be edited. Pages after the first page also include an editable << aimed at displaying the previous page.

Finally, on the last page, the Submit button will be available :



3.6.4 Integrating a form

If a headline is to use forms, it must contain the following call :

Headline models / Modify / Subscribe Model :

```
(...)
«INCLUDE form.phs»
(...)
```

3.7 Variable files

As previously seen in this Chapter, to summarize :

- a frame model integrates one or more variables ;

- afterwards, when a headline is created, the corresponding frame based upon the aforementioned is filled, variable-wise.

This assessing of variable values (as inputed formatted text, image file, link...) creates or updates a file in the system called a variable file (varfile). This `.var` file is situated in the folder corresponding to the current column and contains, the `NAME`s and `VALUE`s of all variables used for the frame.

Therefore, there is one and only one variable file for a leaf frame.

3.7.1 Example of varfile contents

Varfiles are binary files that cannot be opened directly through **Site management** / **Edit file** ; nonetheless, their output can be seen when an export of the current column or the Publisher (4.1.5.1) is done and the resulting export file is viewed.

For instance, the leaf frame as seen at Fig.2.6, related to headline "Josh Jones Tulsa 2010" saved in column `opsmain`, can be shown to have the following (formatted) varfile contents :

export.txt :

```
...
<VARFILE>
NAME=20110323160033_17848.var
<VAR>
NAME=Title
VALUE=Josh rockin' Oklahoma once again
</VAR>
<VAR>
NAME=Date
VALUE=2010/12/04
</VAR>
<VAR>
NAME=Text
VALUE=<P>It was a cold evening when Josh took the stage at...</P>
</VAR>
<VAR>
NAME=_Pict1
VALUE=2
</VAR>
<VAR>
NAME=_Pict1IMap
VALUE=0
</VAR>
<VAR>
NAME=_Pict2
VALUE=3
</VAR>
<VAR>
NAME=_Pict2IMap
VALUE=0
</VAR>
</VARFILE>
...
```

In this example export file, the variables in the varfile `20110323160033_17848.var` and their values can be well seen (see 4.1.5.2 for more details about export file contents).

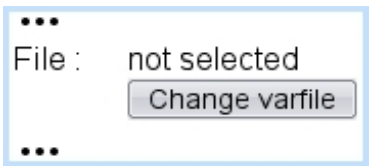
3.7.2 Varfiles as variables

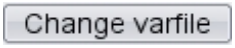
Varfiles can be as a matter of fact loaded as variables for a leaf frame ; to do so, the related frame model integrates the following :

Headline models / **Variables** / **A script frame** :

```
_Varf;Varfile;6;
```

As a result, when a headline containing this frame model is edited, choice is offered to select a varfile :



Upon click of the 

button, a list of headlines is then proposed, and, after one specific headline is clicked, its frames are listed as click-links. Finally selecting a frame this way then stores the related varfile :