



Blue Chameleon CMS  
(Content Management System)



September 10, 2012



# Contents

<b>1</b>	<b>Configuring and administration</b>	<b>13</b>
1.1	System interface . . . . .	13
1.2	Users and usergroups . . . . .	14
1.2.1	Users . . . . .	14
1.2.1.1	User groups that a user belongs to . . . . .	15
1.2.1.2	User management through time . . . . .	16
1.2.2	User groups . . . . .	16
1.2.3	User groups access rights . . . . .	16
1.3	Blue Chameleon Content Management System user rights . . . . .	17
1.3.1	Rights on headlines . . . . .	17
1.3.2	Rights on Articles . . . . .	18
1.3.3	Rights on page objects . . . . .	18
1.3.4	Rights on Images . . . . .	18
1.3.5	Rights on Search . . . . .	18
1.3.6	Rights on URLs . . . . .	18
1.3.7	Rights on columns . . . . .	19
1.3.8	Rights on publishing . . . . .	19
1.3.9	Rights on Data Base Publishing . . . . .	20
1.3.10	Rights on Editions . . . . .	20
1.3.11	Rights on Background Frames . . . . .	20
1.3.12	Rights on Downloads . . . . .	20
1.3.13	Rights on Site management . . . . .	21
1.3.14	Rights on Article Models . . . . .	21
1.3.15	Rights on Headline Models . . . . .	21
1.3.16	Rights on Notice boards . . . . .	21
1.3.17	Rights on System log . . . . .	22
1.3.18	Rights on Scripts . . . . .	22
1.3.19	Administrator rights . . . . .	22
1.3.20	Rights on Hermetic groups . . . . .	22
1.3.21	Information level . . . . .	23
1.3.22	Rights on Books . . . . .	23
1.3.23	Rights on Book Models . . . . .	23
1.3.24	Java option menu . . . . .	23
1.3.25	Connection speed . . . . .	23
1.3.26	Rights on User management . . . . .	24
1.3.27	Rights on Menus, Menu Templates . . . . .	24

1.3.28	Rights on Packages . . . . .	24
1.3.29	Rights on Forms . . . . .	24
1.4	Columns . . . . .	25
1.4.1	Publisher and objects . . . . .	25
1.4.2	Creating columns . . . . .	25
1.4.3	Managing columns . . . . .	26
1.4.3.1	Changing columns . . . . .	26
1.4.3.2	Updating columns . . . . .	26
1.4.3.3	Backups . . . . .	27
1.4.4	Blue Chameleon directory structure . . . . .	28
1.4.4.1	Virtual columns . . . . .	28
1.5	Publisher settings . . . . .	28
1.5.1	File types . . . . .	29
1.6	FTP profiles . . . . .	30
<b>2</b>	<b>Composing content</b>	<b>33</b>
2.1	Principles . . . . .	33
2.2	Headlines models . . . . .	33
2.2.1	Creating a new headline model . . . . .	34
2.2.2	Updating a headline model . . . . .	35
2.2.3	Contents of a headline model . . . . .	36
2.2.3.1	Enriching a headline model with frames . . . . .	37
2.2.3.1.1	Frame templates used by a headline model . . . . .	39
2.2.3.1.2	Updating frame number . . . . .	40
2.2.3.2	Meta Tags . . . . .	40
2.3	Frame templates . . . . .	41
2.3.1	Creating and updating frame templates . . . . .	41
2.3.2	"List" frames . . . . .	42
2.3.3	"Leaf" frames . . . . .	43
2.3.4	Frame style . . . . .	44
2.4	Headlines : creation and editing . . . . .	45
2.4.1	Editing a headline : a basic example . . . . .	45
2.4.2	Editing a leaf frame . . . . .	46
2.4.3	Editing a list frame . . . . .	47
2.4.3.1	Example for a leaf frame as used by a list . . . . .	48
2.5	Layouts . . . . .	50
2.6	Articles and article models . . . . .	50
2.7	Managing pages . . . . .	50
2.7.1	Publishing pages . . . . .	50
2.7.1.1	Multiple publish . . . . .	51
2.7.1.2	Scheduled publishing . . . . .	52
2.7.2	Properties of headlines . . . . .	53
2.7.3	Other management options for pages . . . . .	54

<b>3</b>	<b>Object management</b>	<b>57</b>
3.1	Images . . . . .	57
3.1.1	Uploading an image . . . . .	57
3.1.2	Image folders . . . . .	59
3.1.3	Managing images . . . . .	59
3.1.4	Integrating images . . . . .	59
3.2	Links . . . . .	60
3.2.1	Creating a URL link . . . . .	61
3.2.2	Managing URL links . . . . .	62
3.2.3	Integrating links . . . . .	62
3.3	Scripts . . . . .	64
3.3.1	Creating a script - Management . . . . .	64
3.3.2	Integrating a frame script . . . . .	65
3.3.3	Integrating a link style script . . . . .	66
3.4	Downloads . . . . .	67
3.4.1	Creating a download . . . . .	67
3.4.2	Managing downloads . . . . .	68
3.4.3	Integrating downloads . . . . .	68
3.5	Menus . . . . .	69
3.5.1	Menu scripts . . . . .	69
3.5.2	Menu models . . . . .	71
3.5.3	Creation of a menu . . . . .	71
3.5.3.1	Filling of a menu . . . . .	72
3.5.3.2	Menu result and integration . . . . .	74
3.5.4	More complex menus . . . . .	75
3.5.4.1	A menu script that handles children menu items . . . . .	75
3.5.4.2	A menu with children menu items . . . . .	75
3.6	Forms . . . . .	78
3.6.1	Form models . . . . .	78
3.6.1.1	Form model management - parameters . . . . .	78
3.6.2	Defining forms . . . . .	78
3.6.2.1	Updating a form . . . . .	79
3.6.3	Composing a form . . . . .	80
3.6.3.1	Editing a form page . . . . .	81
3.6.3.2	Form page element types . . . . .	82
3.6.3.3	Editing a form page element . . . . .	84
3.6.3.4	After form is submitted . . . . .	84
3.6.3.5	A form result . . . . .	84
3.6.3.6	A form with several pages . . . . .	85
3.6.4	Integrating a form . . . . .	86
3.7	Variable files . . . . .	87
3.7.1	Example of varfile contents . . . . .	87
3.7.2	Varfiles as variables . . . . .	88

<b>4</b>	<b>Other elements</b>	<b>91</b>
4.1	Site management . . . . .	91
4.1.1	Uploading files . . . . .	91
4.1.2	Creating directories . . . . .	92
4.1.3	Managing files and directories . . . . .	93
4.1.4	Executing SQL . . . . .	93
4.1.5	Export and Import . . . . .	95
	4.1.5.1 Generating export files - importing them . . . . .	95
	4.1.5.2 Export file contents . . . . .	96
4.2	Model defaults . . . . .	98
4.2.1	Model defaults and headlines . . . . .	98
4.3	Books . . . . .	99
4.3.1	Book models . . . . .	99
4.3.2	Creating books . . . . .	99
4.3.3	Modifying a book . . . . .	101
4.4	Notice boards . . . . .	102
4.4.1	Creating and managing notice boards . . . . .	102
4.4.2	Association with a column . . . . .	102
4.4.3	Notice board entries . . . . .	103
4.5	Editions . . . . .	104
4.5.1	General edition management . . . . .	105
4.5.2	Publishing editions . . . . .	106
4.6	Options . . . . .	106
4.6.1	Version . . . . .	106
4.6.2	System Logs . . . . .	107
4.6.3	Expiration report . . . . .	108
4.7	Searching for content . . . . .	108
4.8	Management of objects . . . . .	110
<b>5</b>	<b>Packages</b>	<b>111</b>
5.1	Mandatory scripts . . . . .	111
5.1.1	PackageInstall.phs . . . . .	111
5.1.2	PackageGroupModify.phs, PackageGroupModify1.phs . . . . .	112
	5.1.2.1 Result : package group rights . . . . .	112
5.1.3	PackageMain.phs . . . . .	115
	5.1.3.1 Result : package main page . . . . .	116
5.1.4	PackageUninstall.phs . . . . .	118
5.2	Package general management . . . . .	119
5.2.1	Installing a package . . . . .	119
5.2.2	Updating a package . . . . .	119
5.2.3	Uninstalling a package . . . . .	120
5.3	Using object types and objects to compose headlines . . . . .	120
5.3.1	Enabling a leaf frame to handle packages and an object . . . . .	121
5.3.2	PackageObjTypeList.phs . . . . .	122
5.3.3	PackageObjectList.phs . . . . .	124
5.3.4	PackageObjectName.phs . . . . .	124

5.3.5	PackageObjectView.phs . . . . .	126
5.4	Using attributes . . . . .	127
5.4.1	Enabling a frame to handle attributes . . . . .	127
5.4.2	Scripts for attribute . . . . .	128
5.5	Featuring multiple objects on the same headline . . . . .	129
5.5.1	A leaf frame that handles multiple objects . . . . .	130
5.5.2	Packages : defining more of them . . . . .	132
5.5.3	Object and attribute scripts for this case (guidelines) . . . . .	132
5.5.3.1	The PackageObjectView.phs script for this example . . . . .	134
<b>6</b>	<b>Annex</b>	<b>137</b>
6.1	Leaf frame variables . . . . .	137
6.1.1	"Table" variables . . . . .	139
6.1.2	"Select" variables . . . . .	139
6.1.3	"SQL select" variables . . . . .	140
6.2	System script glossary . . . . .	140
6.2.1	Headline model scripts . . . . .	140
6.2.2	Leaf frame model scripts . . . . .	141
6.3	Icon glossary . . . . .	143
6.4	Form page elements : detailed . . . . .	145
6.4.1	"Active" elements . . . . .	145
6.4.1.1	Checkboxes . . . . .	145
6.4.1.2	Radiobuttons . . . . .	146
6.4.1.3	Listboxes . . . . .	146
6.4.1.4	Input field and text area . . . . .	147
6.4.1.5	Form label - 'with check box' . . . . .	148
6.4.2	"Passive" elements . . . . .	148
6.4.2.1	Hidden field . . . . .	148
6.4.2.2	Form image . . . . .	148
6.4.2.3	Form label - 'normal' . . . . .	148
6.4.2.4	Form separator . . . . .	149





# List of Figures

1.1	The Publisher's main page. . . . .	13
1.2	Defining a new user. . . . .	15
1.3	Groups a user belongs to. . . . .	15
1.4	Defining a user group. . . . .	16
1.5	The most important access rights as defined for a user group. . . . .	17
1.6	Creating a new column. . . . .	26
1.7	Updating a column. . . . .	27
1.8	The properties of your Publisher. . . . .	29
1.9	Defining a FTP profile. . . . .	31
2.1	Creating a headline model. . . . .	34
2.2	Here, a headline model can be updated and its children frame templates be chosen. . . . .	36
2.3	Thanks to this, the pages created on this model will have meta-tags called Author, LastUpdate and a Page Title. . . . .	41
2.4	Creating a new headline. . . . .	45
2.5	First edit ; as both frames use only one template ("Basic text frame" and "Side Menu List"), so clicking on the icon shows them immediately. . . . .	46
2.6	Filling in with information the leaf's various variables. . . . .	47
2.7	A preview of the page. . . . .	48
2.8	Populating a list frame. . . . .	49
2.9	Publishing several headlines in one single time. . . . .	51
2.10	Configuring and viewing data related to headline "Home Page". . . . .	53
2.11	Managing headlines in a file system way. . . . .	55
3.1	Uploading an image. . . . .	58
3.2	Creating a folder for images. . . . .	59
3.3	From here, all images and folders can be globally managed. . . . .	60
3.4	Integrating an uploaded image. . . . .	61
3.5	Creating a link. . . . .	62
3.6	Putting a previously-defined URL link. . . . .	63
3.7	Creating a script to be used in a leaf frame. . . . .	65
3.8	Uploading a file that will be used for downloading purposes. . . . .	67
3.9	Integrating a download, with a 'download' variable. . . . .	69
3.10	Creating a menu model. . . . .	71
3.11	Creating a menu "Basic top menu" based upon model 'Top Menu'. . . . .	72
3.12	Creating a form model. . . . .	79

3.13	Adding parameters to a form model. . . . .	79
3.14	Creating a form based on the example model. . . . .	80
4.1	Uploading a file to be used in headline contents. . . . .	92
4.2	Creating a new directory, directly under the «#SQLWWWHOME#» root. . . . .	93
4.3	Editing a text-based file. . . . .	94
4.4	Outputting the contents of a table. . . . .	95
4.5	Exporting the current column. . . . .	96
4.6	An example of export file contents. . . . .	97
4.7	Creating a model default for a headline model. . . . .	98
4.8	Creating a book model. . . . .	100
4.9	Creating a book. . . . .	100
4.10	From here, book elements can be managed. . . . .	102
4.11	Creating a general notice board. . . . .	103
4.12	Manually adding two entries to this notice board. . . . .	104
4.13	Checking one's own activity for last week. . . . .	108
4.14	Performing a search on objects which publishing name begin by 'Josh'. . . . .	109
4.15	From here, any object can be managed. . . . .	110
5.1	Modifying this package's rights for a particular user group. . . . .	112
5.2	The main page for package Festivals. . . . .	116
5.3	The first install of the package "Festivals". . . . .	119
5.4	The process of selecting an object type and an object for a leaf frame that is package-enabled. . . . .	121
5.5	A selection of object types. . . . .	123
5.6	Finally choosing the object. . . . .	125
5.7	A page design that calls for multiple objects to be featured. . . . .	130
5.8	The 3-object frame after all relevant choices of objects and attributes have been made. . . . .	134
6.1	Composing a checkbox selection for a form. . . . .	146

# List of Tables

5.1	Scripts, functions and variables as used for attribute setting . . . . .	128
6.1	Leaf frame variables . . . . .	138
6.2	Leaf frame variables (continued) . . . . .	139
6.3	System scripts for headline, list and container models . . . . .	141
6.4	System scripts for leaves . . . . .	142
6.5	System scripts for leaves (continued) . . . . .	143
6.6	Blue Chameleon Content Management icons . . . . .	144
6.7	Blue Chameleon Content Management icons (continued) . . . . .	145

In this Chapter, matters related to Blue Chameleon Content Management System's interface as well as usergroup/user prerogatives will be unveiled.

# Chapter 1

## Configuring and administration

### 1.1 System interface

Upon login with your Admin or User accounts, you will find what is featured at Fig.1.1. The Publisher page is composed of various links, which are always displayed whichever page you are on ; therefore any function is easily accessible from anywhere.

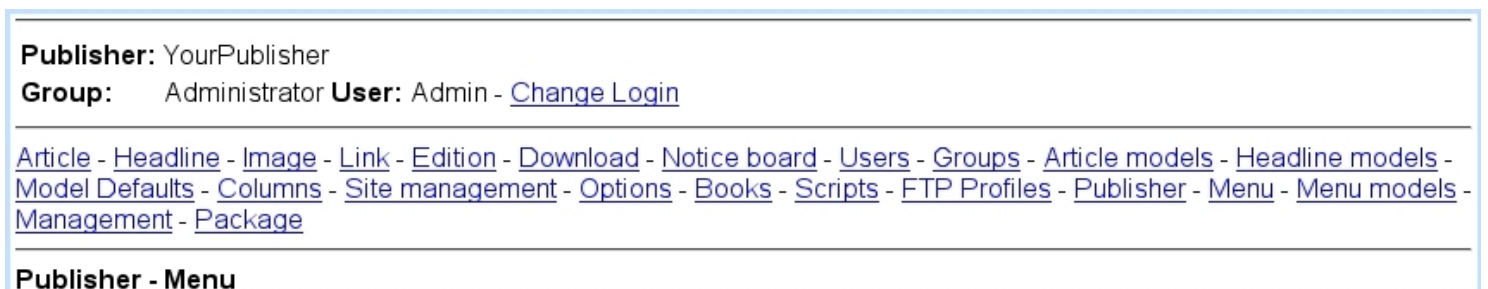


Figure 1.1: The Publisher's main page.

Upon click of a [Link](#), related content then displays below, often with a new group of links (for instance [Modify](#) - [Create](#) - [Delete](#)) allowing to perform further actions.

In the following documentation, how to access a page will be thus displayed as the succession of links clicked to access it, as for instance [Users](#) / [Modify](#) :

## [Users](#) / [Modify](#) :

**Publisher:** YourPublisher  
**Group:** Administrator **User:** Admin - [Change Login](#)

---

[Article](#) - [Headline](#) - [Image](#) - [Link](#) - [Edition](#) - [Download](#) - [Notice board](#) - [Users](#) - [Model Defaults](#) - [Columns](#) - [Site management](#) - [Options](#) - [Books](#) - [Scripts](#) - [Management](#) - [Package](#)

---

**Users:** [Modify](#) - [Create](#) - [Delete](#)

---

**User/Modify**

Select a user:

- Admin [Properties](#) - [Groups](#)
- not defined [Properties](#) - [Groups](#)

What is more, links to follow, if there is any ambiguity, will be made more explicit by the mention of the context : for instance, in the image above, [Admin] [Groups](#) would invite to follow the [Groups](#) link that is found right to "Admin".

## 1.2 Users and usergroups

### 1.2.1 Users



*User group rights for managing Users are detailed at 1.3.26.*

Blue Chameleon Content Management System of course allows several users to use the facilities at the same time ; for that matter, several users can be defined/configured.

Fig.1.2 shows the page where a new user is defined ; there, the following data is inputed :

- the user's real name, under which s/he will be known in Blue Chameleon CMS (maximum length : 20 characters) ;
- login name, used to log on the system using the defined password (for both : minimum length : 5 characters ; maximum length : 10 characters) ;
- the name of your publisher (cannot be modified) ;
- the default column, i.e. which column (1.4) user logs in to by default ;

- the default user group (1.2.2) s/he will belong to ; it is to note that, if user belongs to several user groups (as shown below), this choice defines the group rights that will be used when accessing the default column (provided that group has access rights to that default column) ;
- the default language (not implemented yet) ;
- an email address, that will be in use for document publication functionalities.

Users / Create :

**User/Create**

Enter data of user:

Real Name:

Login Name:

Password:

Publisher: YourPublisher

Default Column:

Default Group:

Default Language:

Email Address:

Figure 1.2: Defining a new user.

### 1.2.1.1 User groups that a user belongs to

As illustrated in Fig.1.3, for each defined user group, the selected user can be [add](#)'ed or [remove](#)'d.

Users / Modify / Robert Groups :

**User/Groups**

To specify the groups the user should belong to, click on remove or add:

<b>Robert</b>	<b>Group</b>	
is not member of Administrator		<a href="#">add</a>
is not member of For temps		<a href="#">add</a>
is member of	Main Group	<a href="#">remove</a>

Figure 1.3: Groups a user belongs to.

### 1.2.1.2 User management through time

The data as inputed during user creation can be modified anytime by following [Users / Modify](#) / [User name] [Properties](#).

If needed, a user can be deleted through [Users / Delete](#) ; this can only be done user does not belong to any group anymore.

## 1.2.2 User groups



*User group rights for managing User Groups are detailed at 1.3.26.*

User groups are generally managed through groups, where they can be created (Fig.1.4), modified and deleted (provided they do not contain any user - if case might be, a list of belonging user(s) is then shown).

**[Groups / Create](#) :**

**Group/Create**

Enter data of group:

Name:

Publisher: LOLTest

Figure 1.4: Defining a user group.

While the data for a user group simply consists in a name, their [Access Rights](#), on the other hand, leave far greater configuring options.

### 1.2.3 User groups access rights

From a user perspective, the different Blue Chameleon CMS columns that will be accessed depend on how the access rights for her/his default user group (as defined in Fig.1.2) have been defined.

To do so, Fig.1.5 then shows an excerpt of the access rights that can be configured for a given user group.

The detailed list of those is explicated next.



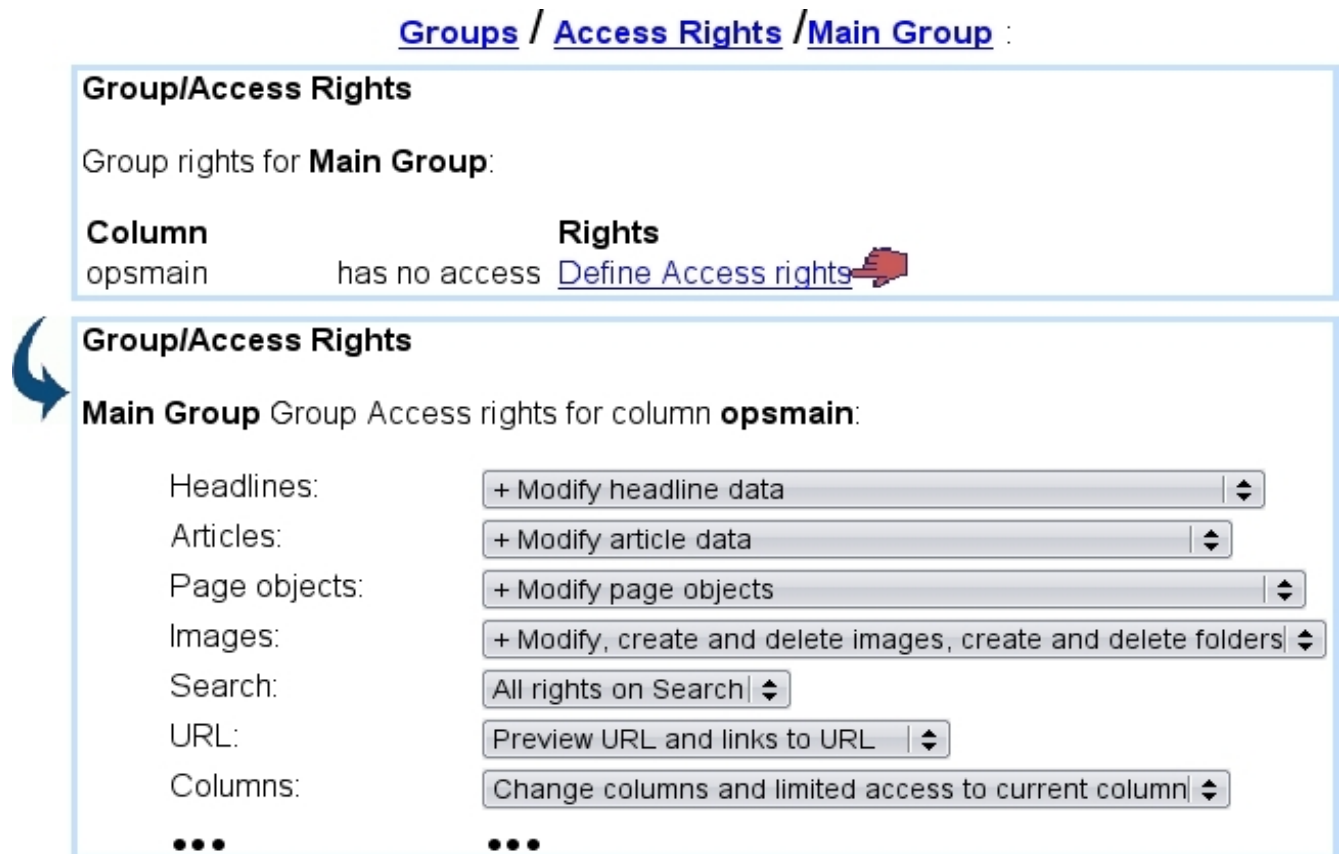


Figure 1.5: The most important access rights as defined for a user group.

## 1.3 Blue Chameleon Content Management System user rights

The access rights of a user group provide an extensive list of functionalities to which gradual rights can be granted. It is to note that rights are being cumulative, i.e. the higher value of a right includes the functions granted by the lower levels.



As Fig.1.5 shows, **each column has its own set of rights**.

### 1.3.1 Rights on headlines

They define a user group's rights to handle headlines (2.4).

One of the following values can be set :

- No rights on headlines (the [Headlines](#) link will simply disappear from the main page) ;
- Preview headlines and link to headlines : the [Preview](#) link for any headline is available and links that point to them can be defined ;

- Modify headline data : the  icon becomes available, allowing to modify a headline frame's data ;
- Modify headline layout : the  icon becomes available, allowing to change the layout selected for a frame (if applicable) ;
- Create and delete headlines, create and delete folders : in addition to free creation and deletion of headlines, those possibilities are also allowed for folders (4.1.2).

This right is combined with the **Column** (1.3.7) and **Publication** (1.3.8) rights.

### 1.3.2 Rights on Articles

Rights for articles (2.6) are similar to the ones provided for headlines (see above), only that there is no "layout" right here.

This right is also combined with the **Column** (1.3.7) and **Publication** (1.3.8) rights.

### 1.3.3 Rights on page objects

#### 1.3.4 Rights on Images

Rights for images (3.1) are defined as follows :

- Preview images and use them : images will be allowed to be previewed and selected when composing a headline ;
- Modify, create and delete images, create and delete folders : user group is omnipotent on image management, and can also create and delete image folders (3.1.2).

This right is combined with the **Publication** (1.3.8) and **Hermetic group** (1.3.20) rights.

### 1.3.5 Rights on Search

Rights for performing searches on contents (4.7) can be either set to 'No rights' or 'All rights'.

### 1.3.6 Rights on URLs

They either rule if URLs (3.2) can be only previewed and used, or if they can be also modified, created and deleted.

### 1.3.7 Rights on columns

They rule the access to Blue Chameleon columns (1.4).

The first two levels as listed below combine with **Headline** (1.3.1) and **Article** (1.3.2) rights to define what user group can do to headlines and articles (modify, delete) while the Modify Column and Create Column rights define the object the user group might access :

- Change columns and limited access to current column : user group has only access to selected headlines/articles in this column ; objects it has no access to will be invisible. When defining the rights for a group that has limited access to a column, the [Access Rights / Group name / Modify Object rights](#) link will allow to define precisely which objects user group has access to ;
- Unlimited access to current column : user group has access to any headline/article in this column ;
- Modify column : user group is allowed to change the configuration of columns ;
- Create column : user group is allowed to create columns.

### 1.3.8 Rights on publishing

They rule how user group is allowed to publish (2.7.1) pages. A user group may only publish the objects it can modify.

The 4-eyes method as mentioned in the rights below is based on the principle that two different users need to review the document's contents before it can be published. This implies that the user who did the last modification will not publish the document herself/himself, for the modification will have to be validated by another user.

- No publishing allowed for articles, headlines, images, notice boards edition : user group cannot publish anything ;
- 4 eyes : ready for publishing : if the last modification to the object has been made by someone else, the user can set the object's publication status to 'ready for publication' ;
- Ready for publishing : user cannot publish the object, but s/he can set its status to 'ready for publication' ;
- 4 eyes : Publishing : user is allowed to publish the object, provided that the last modification has been made by someone else ;
- Publishing : any object that can be modified can be published.

This right is combined with the **Headline** (1.3.1), **Article** (1.3.2) and **Images** (1.3.4) rights.

### 1.3.9 Rights on Data Base Publishing

This is an obsolete function that has been replaced by the Data Base Management package. Depending on the type of user, different menu options are displayed :

- OPS User : this option must be selected so that the standard web-publication interface is displayed ;
- DBP User : not used anymore

### 1.3.10 Rights on Editions


Blue Chameleon CMS columns can be configured so that they work in Edition Mode (4.5) ; the various rights for Editions are :




- Change editions : user can choose the edition in which s/he wants to work ;
- Set active edition : user can define the edition that is currently to be published ;
- Create new edition ;
- Delete edition ;
- Open more than one edition after active one.

### 1.3.11 Rights on Background Frames

When designing headline template frames (leaves, lists and containers), it is possible to define a template as being a background frame (2.3.4), so that it will not be accessible to everybody :

- Background frames do not display : frame is invisible to everyone ;

open button : it is possible to zoom in to background frames, thanks to the appearing  icon ;

- Modify headline data + list manipulation : the  and  become available in order to respectively hide/display and modify data in a background frame ;
- Modify headline layout : the  becomes available to change the layout of a background frame by an other template.

### 1.3.12 Rights on Downloads

Rights ruling downloads (3.4) are as follows :

- Links to download files : only links pointing to downloads can be defined ;
- Modify download files : user can update download files ;

- Create and delete download files : in addition to download files, user will also be able to create and delete folders for downloads ;
- Modify download type : obsolete ;
- Create and delete download types : obsolete ;

### **1.3.13 Rights on Site management**

The management of site (4.1) is ruled by two values, either 'No rights' or 'All site management options'. With the latter, user is able to upload, edit, delete and rename files, create and delete directories as well as execute SQL statements (4.1.4).

### **1.3.14 Rights on Article Models**

They rule how article models (2.6) are managed :

- No rights ;
- Modify article description and image : user can only modify those two items representing the template ;
- Modify article model and variables : any kind of modification (except deletion) on article models is allowed ;
- Create and delete article models : anything can be done for article models.

### **1.3.15 Rights on Headline Models**

They rule how headline models (2.2) are managed :

- No rights ;
- Modify headline description, image and frame style : in addition to headline model name and representing image, user can alter its frame style ;
- Modify headline model, variables, number of frames and model relations : any kind of modification (except deletion) on headline models is allowed ;
- Create and delete headline models : anything can be done for headline models.

### **1.3.16 Rights on Notice boards**

They rule how notice boards (4.4) are managed :

- No rights ;
- Add and delete notice board entries : only board entries can be added and removed to an already-existing notice board ;
- Modify notice board : user can update notice board itself ;
- Create and delete notice board : anything can be done for notice boards.


### 1.3.17 Rights on System log

They rule how system logs (4.6.2) are managed :

- No rights ;
- User's own log : the user can only see her/his own activity in the system log ;
- User's current group log : user sees the activities of all users belonging the group the user is currently logged in as ;
- Log of all groups user belongs to : user sees all the activities of users that belong to user groups s/he also belongs to ;
- Everything : all activities can be seen.

### 1.3.18 Rights on Scripts

They rule how scripts (3.3) are managed :

- No rights ;
- Extended edition mode button : in headline mode, the  icon appears so that user is able to change the parameters of the scripts, as well as the scripts that are selected for frames that handle them ;
- Modify scripts : user can edit any existing script ;
- Create, Delete scripts : user can do anything with scripts.

### 1.3.19 Administrator rights

Set to either 'No rights' or 'All rights', this menu allows in the latter case user to access the following functions :

- define variables which are shared varfiles,
- frame style button in headline edition mode,
- number of model and frame index for a headline frame (2.2.3.1.2),
- managing FTP profiles (1.6),
- updating Publisher data.

For any other group than 'Administrator', it is advised to set this menu to 'No rights'.

### 1.3.20 Rights on Hermetic groups

Certain objects (images, downloads and links), might be required to be available not only on a column level, but also on the whole publisher (any column thereof). According to the choice made for this menu, either 'User's group is open' or 'User's group is hermetic', user will or will not be able to use objects belonging to the publisher.

### **1.3.21 Information level**

This menu holds three choices on how information is displayed :

- No information is displayed ;
- Current menu option is displayed ;
- All information is displayed.

### **1.3.22 Rights on Books**

They rule how books (4.3.2) are managed :

- No rights on books ;
- Preview books : user can only preview books and defined links that point to them ;
- Modify books : user can edit books ;
- Create, delete books : user can do anything on books.

### **1.3.23 Rights on Book Models**

They rule how book models (4.3.1) are managed :

- No rights on book models ;
- Modify books models : user can edit book templates ;
- Create, delete books : user can do anything on book templates.

### **1.3.24 Java option menu**

The options of this menu rule whether Java is used for editing headlines and articles :

- Java disabled : in article/headline edition mode, the standard Blue Chameleon interface will be used to edit a document's data (link and image selection, text input) ;
- Java 1/2 enabled : in article/headline edition mode, Java applets will be available to edit document's data.

### **1.3.25 Connection speed**

The options of this menu rule whether Java applets will preview ('Medium', 'High') or not ('Low') the currently selected image.

### 1.3.26 Rights on User management

They rule how users (1.2.1) and user groups (1.2.2) are managed :

- No rights ;
- Modify one's own user data : user can only change her/his own information ;
- Create users, for all users : select groups in one's own groups : user can create other users S/he can add users to any of the group(s) s/he belongs to ; also, s/he can remove any user from the group(s) s/he belongs to ;
- Delete users, create/delete groups, change group rights : user can create and delete users, groups and change their access rights.

### 1.3.27 Rights on Menus, Menu Templates

They rule how menus (3.5) and menu templates (3.5.2) are managed :

- Select menus
- Modify menus : user can update menu items (change links and labels) and change their order ;
- Create and delete menu items : anything can be done to menu items ;
- Create and delete menus : anything can be done to menus ;
- Modify menu templates : user can change menu templates ;
- Create and delete menu templates : anything can be done to menu templates.

### 1.3.28 Rights on Packages

They rule how packages (5) are managed :

- Access installed packages : user can access any installed package ;
- Install and uninstall packages.

### 1.3.29 Rights on Forms

They rule how forms (3.6) and form models (3.6.1) are managed :

- No rights ;
- Select forms : user can associate a document with a form by selecting the form in the document's properties ;
- Modify forms : user can modify, create and delete form elements as well as form pages ;



- Create and delete forms, create and delete folders : anything can be on forms and folders for forms ;
- Modify form models : user can change the properties of form models ;
- Create and delete form models : user anything can be done on forms and form models.

## 1.4 Columns



*User group rights for Columns to be handled are detailed at 1.3.7.*

The maintenance of a website is generally a several-people job, who update different parts of the site according to their qualifications and prerogatives. In order to limit users' access only to the part(s) they are assigned to, Blue Chameleon CMS can be divided into columns.

While it comes with the default column `opsmain`, other columns can be easily created.

### 1.4.1 Publisher and objects

All Blue Chameleon CMS objects belong to the column they were created in. Only a few objects like images, downloads and links can exist on a Publisher level. Objects that do behave as if they belonged to all columns of the Publisher.

For instance, a user with deletion rights on images (1.3.4) is able to delete the images of the columns s/he is currently in, as well the images belonging to the Publisher.

### 1.4.2 Creating columns

As featured in Fig.1.6, the creation of a column entails the following :

- a description, which will be the name under which this column will appear when a 'Create in:' or 'Directory:' menu is featured ;
- Development, Production and Scripts directories : see 1.4.4 ;
- allow it to work with editions (4.5) and in this case give a name for the first edition ;
- make it a virtual column (1.4.4.1) ;
- allow backups (1.4.3.3) to be made ;
- select a parent column (this menu show *only* virtual columns) ;
- select a FTP profile (1.6).

Once defined, it is necessary to set the column's access rights (1.3) for each user group.

## [Columns](#) / [Create](#) :


**Manage Columns/Create**  
Enter data on new column:  
Description:   
Development Directory: use existing directory:  |   
or enter a new directory:   
Production Directory: use existing directory:  |   
or enter a new directory:   
Scripts Directory: use  directory.  
If no directory is indicated, the Production directory will be used.  
 works with editions  
Name of first edition:   
 is a virtual column  
 make BackUps  
Parent Column:  |   
FTP Profile:  |   
 

Figure 1.6: Creating a new column.

### 1.4.3 Managing columns

#### 1.4.3.1 Changing columns

As soon as more than one column than the default one (`opsmain`) exist, a [Change Column](#) appears on top of the screen (or via [Change Column](#)) ; the name of the current column is mentioned between parentheses :

**Publisher:** YourPublisher **Column:** [Change Column \(opsmain\)](#)  
**Group:** Administrator **User:** Admin - [Change Login](#)  
**Column/Change**  
Select a column:  
• [Info pages](#)

#### 1.4.3.2 Updating columns

Through [Columns](#) / [Modify](#), a column can be updated (Fig.1.7), featuring some options that were present during column creation (column name, parent column, FTP profile and make Backups options) plus new ones :

- a notice board (4.4) to be associated with the column, with automatic/manual choices for Articles, Headlines, Download and Image objects, to specify whether Blue Chameleon will automatically place all updates made to the object in the notice board, or the user will add them manually ;
- a default object expiration period : if choice is else than 'Never', period of days or months after which a headline is considered as expired (the countdown begins as soon as page has been published). If an expiration period has been specified for the column, Blue Chameleon Content Management System will provide the expiration report (4.6.3); this report lists all objects that have expired or are about to ;
- a default link style.

[Columns](#) / [Modify](#) / [Info pages](#) :

**Manage Columns/Modify**

Enter data on column **Info pages**:

Description:

Directories: Development: **dev**  
 Production: **prod**  
 Scripts: **prod**

Notice board:

Notice board usage: Articles:  |   
 Headlines:  |   
 Download:  |   
 Image:  |

Default object expiration period:  |

Parent Column:  |

FTP Profile:  |

Default Link Style:  |

make BackUps

Figure 1.7: Updating a column.

### 1.4.3.3 Backups

If this option had been checked while creating or updating a column, Blue Chameleon will create a back-up of the currently-published page each time an article or a headline has been published.

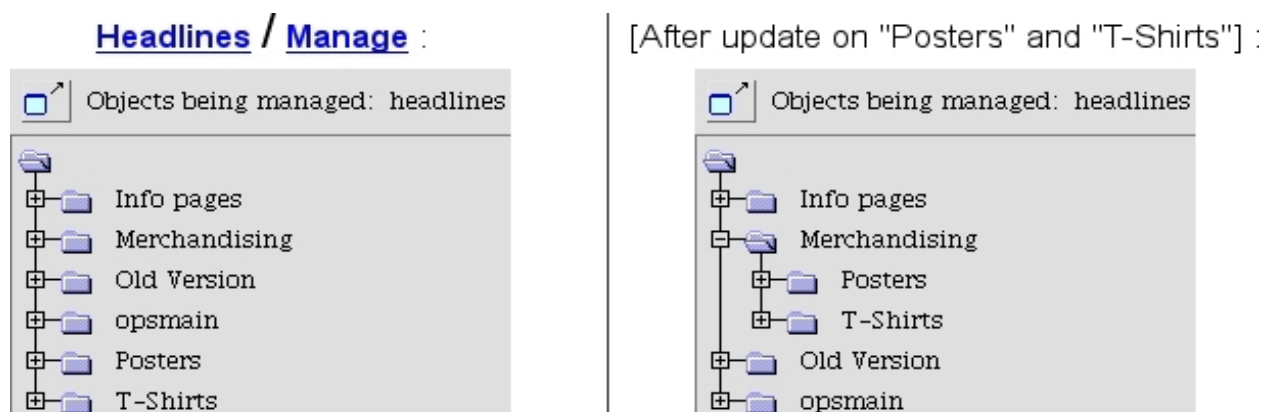
While the back-up mode is activated for a column, the restoring of articles/headlines is also supported, but only applying to the published version of a page (restoring a page does not undo the changes applied to the article/headline).

## 1.4.4 Blue Chameleon directory structure

### 1.4.4.1 Virtual columns

Activating this option for a column will make it contain no object (headline, articles,...) other than columns. Indeed, a virtual column is used to group several columns under one name. It will not appear when changing columns (1.4.3.1) but will nonetheless be available as a 'parent column' when creating or updating an other column, or when browsing for objects.

For instance, if the structure has two "stranded", yet-related columns such as "Posters" and "T-Shirts", it would be interesting to group those under a same, newly-created parent virtual folder called "Merchandising". This is achieved for both of columns via a column update (1.4.3.2) where 'Parent folder' is set to "Merchandising". As a result, the tree structure has become clearer to manage :



## 1.5 Publisher settings

From here, as featured in Fig.1.8, the following is configured :

- its name ;
- how often actions are recorded to the System log (4.6.2) :
  - 'no logging' : actions are never recorded ;
  - 'always log publish, other actions only once a day' : only the publishing of objects is logged as many times it happens ;
  - 'all actions are always logged'.


As for the 'time range' menu, it rules how much time logs are kept before being automatically deleted (options are 1,2 and 3 weeks, 1 and 3 months) ;


- which FTP Profile (1.6) is chosen for the Publisher ;
- an email address and server ;
- what are the normal and secured publisher domains ;
- to which object(s) domain is added to ;
- whether secure downloads and page objects are enabled.


**Publisher / Modify :**

**Publisher/Modify**

Name:


System Log:  

time range:  

FTP Profile:  

Email address:

Email Server:

Scheduled Publishing User:  

Publisher Domains: http:

https:

Add domain to:

- page links
- image urls
- download urls
- links to non OPS objects

Rights:

- Enable Secure Downloads
- Enable Page Objects




Figure 1.8: The properties of your Publisher.

### 1.5.1 File types

This page, accessed via the  button as seen on the *Publisher Settings Page*, allows to define and update which file types are proposed when a "download" object is created (3.4.1) :

[Publisher / Modify / File Types](#) :

**Publisher/File Types**

**Downloads**

File Type

Binary	Modify	Delete
Excel (.xls)	Modify	Delete
Executable	Modify	Delete
Image	Modify	Delete
Miscellaneous	Modify	Delete
PDF (.pdf)	Modify	Delete
Powerpoint (.ppt)	Modify	Delete
Sound	Modify	Delete
Video	Modify	Delete
Word (.doc)	Modify	Delete
ZIP (.zip)	Modify	Delete
<input type="text"/>	Create	

## 1.6 FTP profiles

Blue Chameleon Content Management System handles FTPs.

For them to be used, at least a FTP profile must be created (Fig.1.9), with the following information :

- a type, amongst 'FTP', 'SFTP', 'IIP' and 'Queue only' ;
- a name under which this profile will be referred to as ;
- the server address and port for the FTP server ;
- connection information for it such as user name, password and root.

Afterwards, all of these information can be updated through [Modify](#) and a FTP profile can be removed through [Delete](#).

A FTP profile is assessed to the Publisher, on its Settings page (1.5)

### FTP Profiles / Create :

**FTP profile/Create**

Type:

Profile Name:

Server:

Port:  In general: FTP port=21; IIP port=22.

User Name:

Password:

Root:

---

**FTP profile/Create**  
New FTP profile has been created.

Figure 1.9: Defining a FTP profile.





# Chapter 2

## Composing content

Now that matters related to Blue Chameleon Content Management System configuring and administration have been dealt with at the previous Chapter, it is time to see how webpages themselves are composed.

### 2.1 Principles

Blue Chameleon Content Management System is based upon those simple principles :

- a website is a collection of pages called "**articles**" and "**headlines**" ;
- these articles and headlines are created using *Article models* and *Headlines models* ;
- these templates include *frame templates* ;
- articles and headlines are enriched with objects such as images and links and, when complete, are published on the website.

Frame templates consist in :

- containers, which can contain any type of frame - children frames are in fixed number, decided during modification of the frame template ;
- lists, which can also contain any type of frame - children frames are in variable number, added or subtracted when headline or article is edited ;
- leaves, which cannot contain children frames, but exclusively provide Variables to provide various content input when headline or article is edited.

### 2.2 Headlines models



*User group rights for handling Headline Models are detailed at 1.3.15.*

## 2.2.1 Creating a new headline model

Fig.2.1 shows how a headline model can be created.

[Headline models](#) / [Create](#) :

**Headline Models/Create**

Enter Headline/Frame Model Description:

Model description:

Model image:

Frame Style:

- Frame is displayed
- User may toggle between displayed and hidden state of frame
- When editing a headline, frame is shown in a table with a border
- Frame is a background frame (not editable by user)
- Frame has extended edition mode (with edition script vars only)
- display mode is not applied to children of frame
- user may choose file extension for published file (applies only to headline models)

Edition Script:    
This is only to be defined if a custom form is to be used for the edition of the leaf content.

Layouts:   
Number of alternate layouts available besides the default layout. Alternate layouts are being published in the *layoutn* subdirectory of the column directory (*layout1* for 1st additional layout, *layout2* for 2nd additional layout, etc...). Links to documents with multiple layouts always point to the default layout.

Model type:    
If you are defining a headline or a container model, you must indicate the number of frames used by the model

Number of frames:

use headline model in any column of the current Publisher

Base model on:


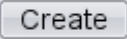


Figure 2.1: Creating a headline model.

This page shows many provides many elements :

- a name for this new headline model, as well as an image (200x150 pixels is ideal) that will represent it ;

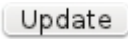
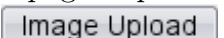

- options for frame style (2.3.4) ;
- an edition script ;
- number of *alternate* layouts (2.5) ;
- the model type : here, "headline" is chosen but other choices include 'container', 'list', 'leaf' (2.3) and 'book' (see 4.3.1 for this particular object) ;
- the number of frames (if any) this headline will contain ;
- if headlines already exist, it is possible to choose one upon which the new headline will be based.

Upon click on the  button at the bottom of this page, one is redirect to a new page where the text of the headline model is inputed, as described as follows.

Most of this data (headline model name, image...) can afterwards be updated through [Headline models](#) / [Properties](#), as explained next.

## 2.2.2 Updating a headline model

As featured on Fig.2.2, it is possible to update a few of a headline model's information, such as its name, model image, frame style, number of layouts and column availability.

Buttons at the bottom of the page respectively allow to  changes done to the form and, if the case, validate . Another button leads to the configuring of the  (2.2.3.2) that headlines created from this model will inherit.

What is more, a link at the bottom of the page allows to select templates that may be used for child frames, as explained in 2.2.3.1.1.

### Headline Models/Properties

Enter Model Description:

Model description:

Model image:

Frame Style:

- Frame is displayed
- User may toggle between displayed and hidden state of frame
- When editing a headline, frame is shown in a table with a border
- Frame is a background frame (not editable by user)
- Frame has extended edition mode (with edition script vars only)
- display mode is not applied to children of frame
- user may choose file extension for published file (applies only to headline models)

Layouts:   
Number of alternate layouts available besides the default layout. Alternate layouts are being published in the layout*n* subdirectory of the column directory (*layout1* for 1st additional layout, *layout2* for 2nd additional layout, etc...). Links to documents with multiple layouts always point to the default layout.

Availability:  use headline model in any column of the current publisher

- available in all columns of the publisher

To limit the usage of the model to certain columns of the publisher, [click here](#).

use headline model only in current column

To select templates that may be used for child frames, [click here](#)

Figure 2.2: Here, a headline model can be updated and its children frame templates be chosen.

### 2.2.3 Contents of a headline model

Just after the validating as previously seen, or anytime through [Headline models / Modify](#), the contents of a headline is inputed/alterd.


The following loads for instance the picture "Banner.gif" that has been previously uploaded (see 4.1.1 on how to upload any file) to the `img/` directory :

## [Headline models](#) / [Modify](#) / [Main Template](#) :

### Headline Models/Modify

Enter the text for the headline model. (For help, see at the end of the page)

```
<HTML>
<BODY>
  <INCLUDE design.phs>
  <IMG SRC="<#SQLWWWHOME#>/img/Banner.gif">
  <!-- Other HTML stuff...-->
</BODY>
</HTML>
```

Update 



*The design.phs script must always be INCLUDE'd in any headline model's contents*

Basically, here is composed, with HTML and a few OSL code the layout of the headlines that will be modelled after this headline model.

Of course, Blue Chameleon Content Management System allows to go further by equipping this headline model with frames.

### 2.2.3.1 Enriching a headline model with frames

The great flexibility of the Content Management System allows to make a headline model contain frames that are chosen amongst the list of defined frame templates. Afterwards, during the editing of the headline itself, those frames will be edited individually.

Code 1 shows how the example headline model's content is made to contain two frames, one besides the other.

There, a system script called `frame.phs` is called with the argument `_FrameIndex` corresponding to the number of current frame to insert. There are therefore as many of those INCLUDEs to be put as the 'Number of frames:' that was defined during headline model creation (Fig.2.1).

This code serves as to provide two frames for the headline ; now have to be defined *which* frame templates will be proposed by the headline model.

---

**Code 1** This headline model will allow two frames :

---

[Headline models](#) / [Modify](#) / [Main Template](#) :

```
<HTML>
(...)
<BODY>

«INCLUDE design.phs»

<IMG SRC="«#SQLWWWHOME#»/img/Banner.gif">

<TABLE>
<TR>
  <TD>
    «INCLUDE frame.phs;_ParentId=«_FSId»;_FrameIndex=1»
  </TD>
</TR>
<TR>
  <TD>
    «INCLUDE frame.phs;_ParentId=«_FSId»;_FrameIndex=2»
  </TD>
</TR>
</TABLE>
(...)
```

---

2.2.3.1.1 **Frame templates used by a headline model** The page displaying a headline model's properties (2.2.2) also allows to define which frame templates it can use :

[Headline models](#) / [Properties](#) / [Main Template](#) :

...

To select templates that may be used for child frames, [click here](#)

**Models available for all frames**

All Frames  
[Add a model](#)

**Additional Models available for specific frames**

1. Frame  
[Add a model](#)

2. Frame  
[Add a model](#)

The [Add a model](#) link then shows a list of frames (containers, lists and leaves). One is selected for each numbered frame, or for 'All Frames', which means any of the numbered frames will be able to use it. Once a selection is done, something like this is obtained :

**Models available for all frames**

All Frames  
[Add a model](#)

**Additional Models available for specific frames**

1. Frame  
Basic text frame [Delete this model](#)  
[Add a model](#)

2. Frame  
Side Menu List [Delete this model](#)  
[Add a model](#)

If a mistake was made, a [Delete this model](#) link allows to remove any frame template from the list of frame templates this headline model can use.



*This selection of frame templates also happen for containers and lists, which likewise accept children frames. .*

**2.2.3.1.2 Updating frame number** During development of a website, it can happen that a headline model has to allow a new frame, or maybe has no use for a specific frame anymore. Such updates can be proceeded with through [Headline models / Frame number](#), where links such as [Add a frame](#) or [Delete frame n°...](#) enable to perform these actions :

### Headline Models/Number of frames

Headline model **Main Template** is being used 1 times.  
Changing the number of frames in the model will affect all the headlines which are using it.

Select Action:

- [Add a frame](#)
- [Delete frame n°1](#)
- [Delete frame n°2](#)

### 2.2.3.2 Meta Tags

These serve as to provide information on a headline (such as the author, the last modify date,...) and will be inserted automatically into the "header" part of HTML code of the headline. They are inherited from the headline model on which the headline is based on.



*For Meta Tags to be inserted, the headline model contents must include the following call, in the page's header :*

[Headline models / Modify / Some headline model](#) :

```
<HTML>
<HEAD>
...
«INCLUDE metatags.phs»
...
</HEAD>
(...)
```

Meta-Tags in use can be configured while at updating a headline model's properties, as featured in Fig.2.3.


There, the information give is twofold :

- the document properties, with two already-existing tags Author and LastUpdate ; they, as well as any other tag that is 'd, can be modified and deleted



**Headline Models/Meta Tags**

**Document properties** (coded `<META NAME="TagLabel" CONTENT="User defined">`)

Tag Label	Value Type		
<input type="text" value="Author"/>	String 80	<input type="button" value="Modify"/>	<input type="button" value="Delete"/>
<input type="text" value="LastUpdate"/>	String 80	<input type="button" value="Modify"/>	<input type="button" value="Delete"/>
<input type="text" value="Page Title"/>	Page Title	<input type="button" value="Create"/>	

**HTTP headers** (coded `<META HTTP-EQUIV="HTTP headers" CONTENT="User defined">`)

HTTP headers	Value Type	
<input type="button" value="-choose a header-"/>	String 80	<input type="button" value="Create"/>

Figure 2.3: Thanks to this, the pages created on this model will have meta-tags called Author, LastUpdate and a Page Title.

through the eponymous buttons. 'Type' as defined for a tag creation must handle the tag information given afterwards, either a String (of maximum length 80, 255 or 512 characters) or a Page Title (the example here) ;

- the HTTP headers, with the same value types as above, giving the following choice for header :
  - Cache-control
  - Content-Encoding
  - Content-Language
  - Content-Type
  - Data
  - Expires
  - Pragma

## 2.3 Frame templates

Now that the way of creating frame-using headline models has been thoroughly explained, it is time to see how frame templates, i.e. container-, list- and leaf models are created.

### 2.3.1 Creating and updating frame templates

Frames are created in the exact same way as for headline models (Fig.2.1) i.e. through [Headline models](#) / [Create](#) except that now 'container', 'list' or 'leaf' is chosen from the 'Model type:' menu.

Updating a frame also happens in a similar way as for headline models, i.e. through [Headline models](#) / [Properties](#).

According to frame template type, management can vary :

- 'container' being the only type of frame that contains a **fixed** number of other frames, a frame number has to be given during creation. The including of children frames (2.2.3.1), their selection (2.2.3.1.1) and the updating of frame number (2.2.3.1.2) happen similarly as for headline models ;
- for 'list' type of frame, only the selection of used children frames must be done ;
- for 'leaf' type, no child frame exist ; but on the other hand, suitable variables must be given.

### 2.3.2 "List" frames

A list frame template is quite particular : indeed, during the editing of a headline (see 2.4.3), it will allow, thanks to certain icons, to add (or remove) as many list elements as needed ; those list elements are frames.

For this, the list frame template's contents are set as shown in Code 2.

---

**Code 2** A simple list frame.

---

[Headline models](#) / [Modify](#) / [Side Menu List](#) :

```
«VAR NEW _ItemCount»
«INCLUDE ListItems.phs;_ListFrameItems="_ItemCount"»
«IF _ItemCount>0»
  «SQLREPEAT» <!-- element «#SQLREPEAT# »->
    «INCLUDE frame.phs;_ParentId=«_FSId»;_FrameIndex=«#SQLREPEAT#»»
  «/SQLREPEAT 1;«_ItemCount»»
«ENDIF»
```

---

There, the number of items (frames) in the list is counted ; then, if there are frames, through the SQLREPEAT loop, each of them is INCLUDE'd.



*It is then necessary to create frames (e.g. 'Simple List Element') that will be supported by the list frame.*

### 2.3.3 "Leaf" frames

A leaf frame is an end frame in itself (like the leaves of a tree) and does not contain any frame.

Nonetheless, it can be richly configured with the help of *variables*, something which does not exist with containers and lists. Indeed, going on [Headline models](#) shows a [Variables](#) link, that, upon click, shows only leaf-type frames.

Variables inputed there will then make appear, when a headline containing this frame is edited, various kind of inputs such as text lines, text fields, images, select menus... upon which information will be given.



*For more details about objects for leaf frames and their variables, see Chapter 3.*

Code 3 shows an example for the contents of a leaf and the related variables.

---

**Code 3** Contents and variables for a simple leaf.

---

[Headline models](#) / [Modify](#) / [Concert photos](#) :

```
<B>«Title»</B> («Date»)<BR>
«Text»

<TABLE>
  <TR>
    <TD>
      «INCLUDE image.phs;pImage="_Pict1";Html="border='0'"»
    </TD>
    <TD>
      «INCLUDE image.phs;pImage="_Pict2";Html="border='0'"»
    </TD>
  </TR>
</TABLE>
```

[Headline models](#) / [Variables](#) / [Concert photos](#) :

```
Title;Title;1;
Date;Date;1;
Text;Text;2;
_Pict1;Picture 1;3;
_Pict2;Picture 2;3;
```


---

There, the leaf contents use several variables («Date», «Title», ...) to output on the final page a title, a descriptive text as well as two pictures. These will be inputted during frame editing using the defined [Variables](#) : see 2.4.2 to view how frame editing will look like.



*For the full list and description of leaf frame variables, please check the Annex (6.1).*

### 2.3.4 Frame style

Frame style options are featured during creation/edit of a headline or frame, plus anytime when a headline's contents themselves are edited (2.4), through the  icon.

For any frame, they are as follows :




**Headline/Modify Frame Style**

Update Frame Style:

Frame Style:  Frame is displayed  
 User may toggle between displayed and hidden state of frame  
 When editing a headline, frame is shown in a table with a border  
 Frame is a background frame (not editable by user)  
 Frame has extended edition mode (with edition script vars only)  
 display mode is not applied to children of frame

Checkboxes there decide whether :

- frame is displayed ;
- user has the possibility to hide/show this frame (ticking this will make a  icon appear for this frame) ;
- whether frame contents are shown with a border for easier view ;
- frame is a background frame ;



*User group rights for handling Background Frames are detailed at 1.3.11.*

- frame has extended edition mode (concerning frames that handle 'edit script' variables) ;
- display mode is not applied to children of frame (this is the default choice)

## 2.4 Headlines : creation and editing



*User group rights for Headlines to be handled are detailed at 1.3.1.*

Once headline models have been created, a headline is simply created as featured on Fig.2.4.

[Headlines](#) / [Create](#) / [List of H. models] [Main Template](#) :

**Headlines/Create**

Page Title:

Publication   
name: Besides the system name, the page will be published under the name indicated here.

Folder:

**Headlines/Create**

Headlines: Home Page  
Model: Main Template

Headline **Home Page** created. Click here to [modify](#) it.


Figure 2.4: Creating a new headline.

After the selection of the headline model it will be based on, a headline is given a title (under which it will be referred to as when browsing and modifying headlines) and a name under which it will be published (2.7.1).

### 2.4.1 Editing a headline : a basic example

Through [Headlines](#) / [Modify](#), the list of headlines in the current column is displayed for each to be edited.

When editing a headline for the first time, Blue Chameleon Content Management System displays what is featured at Fig.2.5 :

- HTML code as inputted in the headline model contents is interpreted : here, the display of image "Banner.gif" (cf. Code 1) ;
- frames that have to be displayed are shown yet unfilled ; then, clicking then on the  icon either fills the frame with the assigned frame template or shows the available frame templates for this frame (2.2.3.1.1).

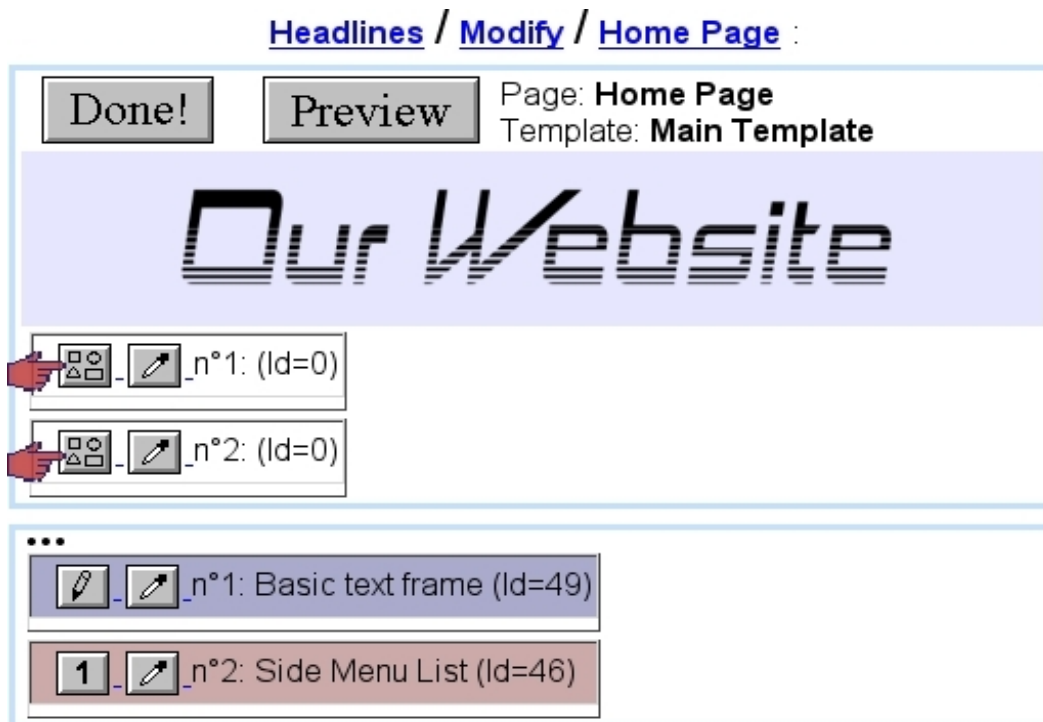



Figure 2.5: First edit ; as both frames use only one template ("Basic text frame" and "Side Menu List"), so clicking on the icon shows them immediately.

The buttons **Done!** and **Preview** as seen at the top of the page allow respectively to validate any modification on the headline and see how page would be rendered in real (without the frame menus and icons).

It is to note that these buttons have been created thanks to the inclusion of the `design.phs` script (Code 1), hence the importance of always including it for a headline model.

 *Outside the headline editing context, the preview of a headline can also be done through [Headlines](#) / [Preview](#).*

Any editing to a frame is then performed through its own  icon.

## 2.4.2 Editing a leaf frame

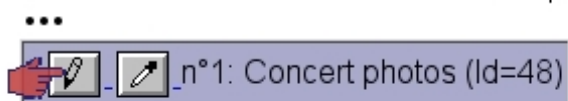
The output for the example Code 3 (a simple leaf frame template contents and variables) is featured at Fig.2.6. There, user fills the different variables that had been defined such as text fields, text zone (as using the Document Editor) and uploaded images (3.1).

**Done!**

**Preview**

Page: **Josh Jones Tulsa 2010**

Template: **Concert template**



### Headlines/Modify/Frame Data

Page: **Josh Jones Tulsa 2010**

Template: **Concert photos**



Title :

Date :

Text :

It was a cold evening when Josh took the stage at...|

Picture 1 : Josh (Tulsa 2010)

Picture 2 : Bassist (Tulsa 2010)

Figure 2.6: Filling in with information the leaf's various variables.

When everything is done, page is firstly validated via  ; it can then be  ed (Fig.2.7), and, if modifications done are fine to finally validated,  is hit.

### 2.4.3 Editing a list frame

Lists frames are slightly more complicated, as they can yield an indefinite number of frames. In the following example, a list frame template "Side Menu List" has been created as in Code 2 and is using (2.2.3.1.1) leaf model "Simple List Element" (see 2.4.3.1



Figure 2.7: A preview of the page.

for more details about it).

As featured on Fig.2.8 :

- first, the button of the list frame is clicked, which adds a new sub-frame, yet undefined ;
- the of this sub-frame is clicked, which already fills it with a "Simple List Element" leaf (as it is the only frame Side Menu List uses) ;
- the now-appearing Simple List Element frame then offers several icons, amongst which and : clicking on the latter adds another Simple List Element on the bottom (the former would have added it on the top) ;
- and so on, until all needed list elements are here.

Each list element features of course the icon, which allow to edit it ; also, the , allow to place the list element before the previous one (if not on the top of the list) or after the next one (if not on the bottom of the list).

A list element can be removed through .

As usual, the icon allows to edit list element's frame style (2.3.4).

### 2.4.3.1 Example for a leaf frame as used by a list

The leaf model "Simple List Element" as used above has the following contents :

```
<A HREF='«_Link »'>«_Text »</A>
```



[Headlines](#) / [Modify](#) / [Home Page](#) :

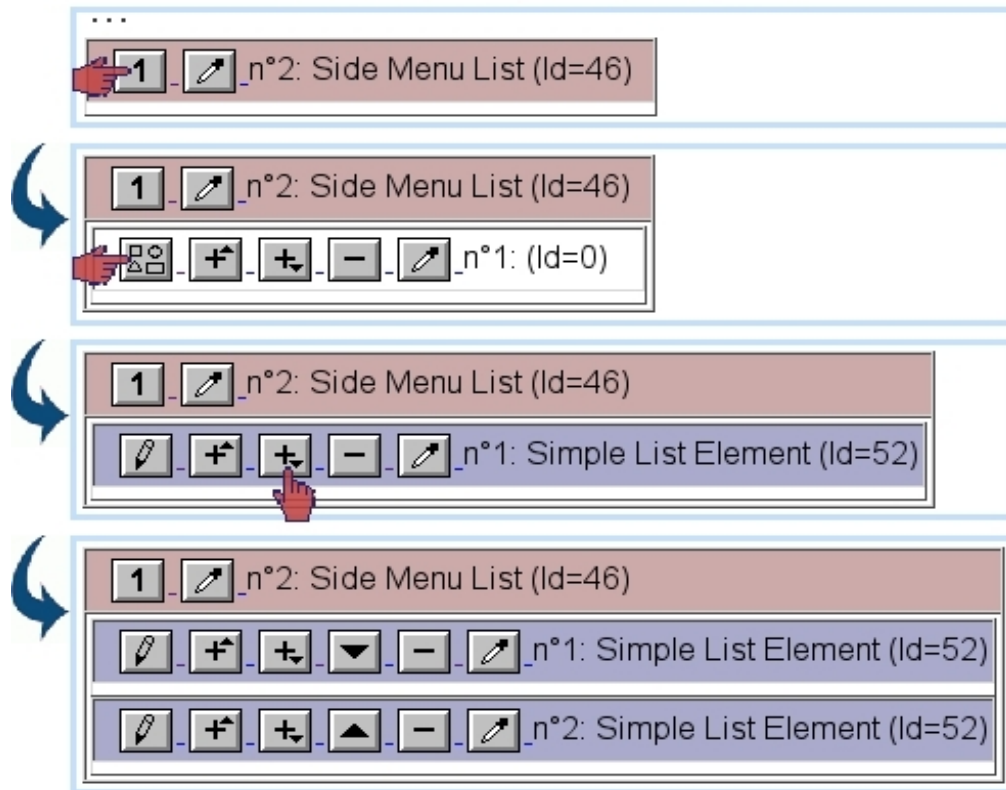
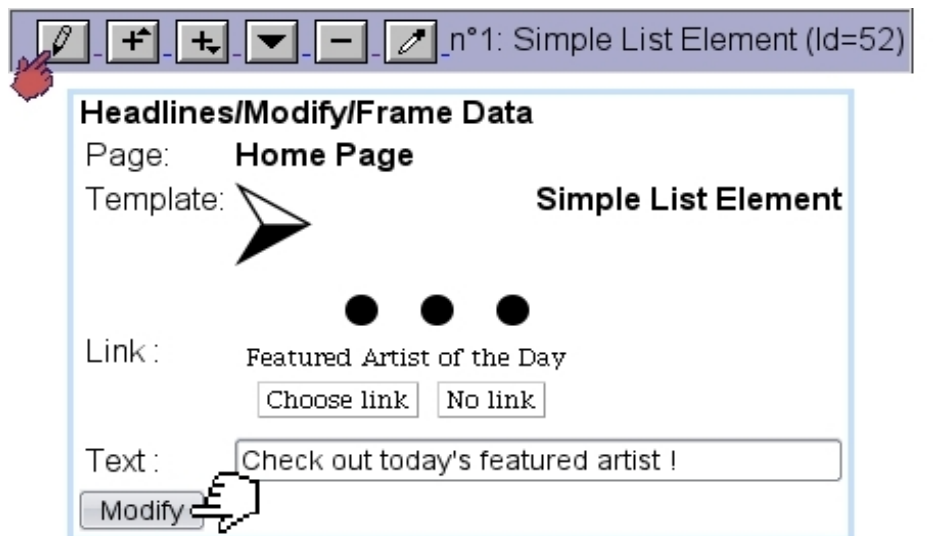


Figure 2.8: Populating a list frame.

and the corresponding [Variable](#) are defined as :

```
_Link;Link;4;  
_Text;Text;1;
```

When a list element will be edited, the following output will be shown :



## 2.5 Layouts

## 2.6 Articles and article models



*User group rights for Articles and Article Models to be handled are respectively detailed at 1.3.2 and 1.3.14.*

Articles and article models are in fact a deprecated version of Headlines, and therefore are currently not used anymore.

## 2.7 Managing pages

How headlines are managed through time is described next.

### 2.7.1 Publishing pages



*User group rights for publishing pages are detailed at 1.3.8.*

As a matter of fact, while most objects in Blue Chameleon Content Management System can be published at the time of their creation/modification, a page that is edited is not available right now on the website : it needs to be explicitly published first.

Publishing is done through [Headlines](#) / [Publish](#), where a list of headlines in the current column - including its virtual columns (1.4.4.1) - is shown, with their publish status, related .htm file in the current folder, and the name under which it is published. For instance, the following :

**[Headlines](#) / [Publish](#) :**

```
...
o Home Page (Modified)
  (20110323143529_16608.htm, Home_Page.htm)
...
```

would show that headline "Home Page", which is published under Home\_Page.htm using file 20110323143529\_16608.htm (see 'Properties of headlines' (2.7.2) for more details about publication name and related file) has been edited but not published yet : simply clicking on [Home Page](#) would then publish it, and a (Published) status would replace (Modify).

As a column could contain a quite sizable number of pages, it is possible to publish multiple pages with a single click.

### 2.7.1.1 Multiple publish

On the [Headlines / Publish](#) page, a link [Click here for multiple headline publish](#) (also available for articles) redirects to a page as featured in Fig.2.9.

[Click here for multiple headline publish](#) :

#### Detailed Publish

<input type="checkbox"/> articles	Current	opsmain
<input checked="" type="checkbox"/> headlines		
<input type="checkbox"/> images	Current	opsmain
<input type="checkbox"/> downloads	Current	opsmain

Check Objects:  modified  published  all  none

#### Headlines

...

- Featured Artist of the Day (Modified - 20110318103742\_18668.htm, Featured.htm)
- Festivals Main (Published - 20110330115149\_24394.htm, Festivals\_Main.htm)
- Home Page (Modified - 20110323143529\_16608.htm, Home\_Page.htm)

...

#### Detailed Publish

Are you sure you want to publish the following objects?

#### Headlines

[List of ticked headlines]

#### Published objects 1 to 8 from 8

Publication succeeded.

Figure 2.9: Publishing several headlines in one single time.

There :

- a framed table with checkboxes (articles, headlines,... ) stands as to pick objects of interest ; menus with options 'Current', '+Subfolder', 'Section' let then choose where the publishing task takes place, to where specified by the right right-hand menus, detailing the current column and its virtual columns ;
- a possibility to check objects, whatever their publication status is ;

- ...for the above choices, a **Display** button used to refresh the...
- list of headlines, each accompanied with a checkbox deciding if the publishing applies on it ;
- upon click on the **Publish**, a confirmation screen then appears, featuring the number of ticked pages to be published ;
- finally, upon validation, confirmation screen shows the result of the mass-publication.

Headlines that belong to edition-enabled columns (Fig.1.6) have dedicated publishing functionalities (4.5.2).

### 2.7.1.2 Scheduled publishing

Scheduled publication allows - in the multiple publish context - to program the automatic publication a selection of headlines on a specified date and time ; for it to be proposed, the Publisher settings (1.5) must have a user selected for 'Scheduled Publishing User'. Then, menus allowing to choose date and time will be provided :

[Click here for multiple headline publish](#) :

**Detailed Publish**  
•••

**Schedule publication**  
 Please check to schedule publication for specified date:  
 Day: Month: Year: Hour: Minute:  
 14 / 4 / 2011 2 : 00

**Headlines**  
 [List of headlines to select for publishing]

**Publish**

---

**Detailed Publish**  
 Are you sure you want to publish the following objects?

**Headlines**  
 [List of ticked headlines]

**Publishing Scheduled for:**  
 dd/mm/yyyy hh:mm: 14/4/2011 2:00

**Validate**

---

**Published objects 1 to 2 from 2**  
 Publication has been scheduled for the specified date.

## 2.7.2 Properties of headlines

[Headlines](#) / [Properties](#) / [Home Page](#) :

### Headlines/Properties

#### Properties

Page Title:

Publication date: 1/4/2011

Publication name: **YourPublisher**/opsmain/

File name: **YourPublisher**/opsmain/20110323143529\_16608.htm

Expiration period:

Associated Form: none selected

Link page as:

#### Meta Tags

Author:

LastUpdate:

Page Title: Home Page

#### Reports

[Click here](#), to view activity related to this headline.

#### Crossreferences

objects referenced by this object     references to this object

Home Page

featuredartist.htm

Figure 2.10: Configuring and viewing data related to headline "Home Page".

This page is useful to check any activity and useful data related to a headline. As featured on Fig.2.10, information and data configuring is multifarious :

- the headline's name, which can be modified ;
- when the last publication was done ;
- under which name this page is published.



*After having created a headline, it is necessary, for linking purposes, to change this name to something URL-compliant, e.g. Home\_Page.htm.*

*The page can then be added as a link, or accessed, via the path displayed on the screen (here, YourPublisher/opsmain/Home\_Page.htm)*

- by which physical file of the server (e.g. 20110323143529\_16608.htm) it is represented (name is mainly composed of year, month, day of when it was first created) ;
- whether this page will expire (by default set as in 1.4.3.2) ;
- whether it has an associated form (3.6) ;
- whether it is linked as http or https ;
- the Meta Tags, where the information configured in 2.2.3.2 is finally filled ; as a result, the Home\_Page.htm page will contain the following :

```
...
<HEAD>
<meta name="Author" content="Robby K" />
<meta name="LastUpdate" content="05/04/2011" />
<meta name="Page Title" content="Home Page" />
</HEAD>
...
```

- a direct click-link to check the reports (4.6.2) related to this headline ;
- where it refers to, as well as where it is referenced from, as a tree structure that can be browsed.

Any modification in the fields and/or drop-down menus must be validated through the  button.

### 2.7.3 Other management options for pages

A tree-structure view aimed at managing headlines directory-wise is accessible via [Headlines / Manage](#), as illustrated in Fig.2.11.

Through a right-click on a headline's name, it is possible to copy a headline in the same column or to another and also - if headline is in the current column, to rename it. Previewing and refreshing are also available.

When right-click is done on a folder's name, the only options are refreshing and creating a new folder inside it (if clicked folder corresponds to the current column).

### Headlines / Manage :

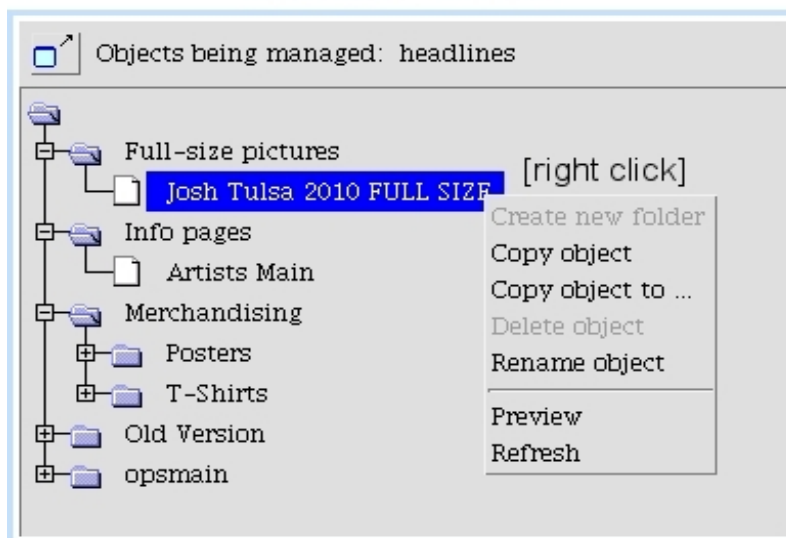


Figure 2.11: Managing headlines in a file system way.






# 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  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 inputted :

- 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](#).

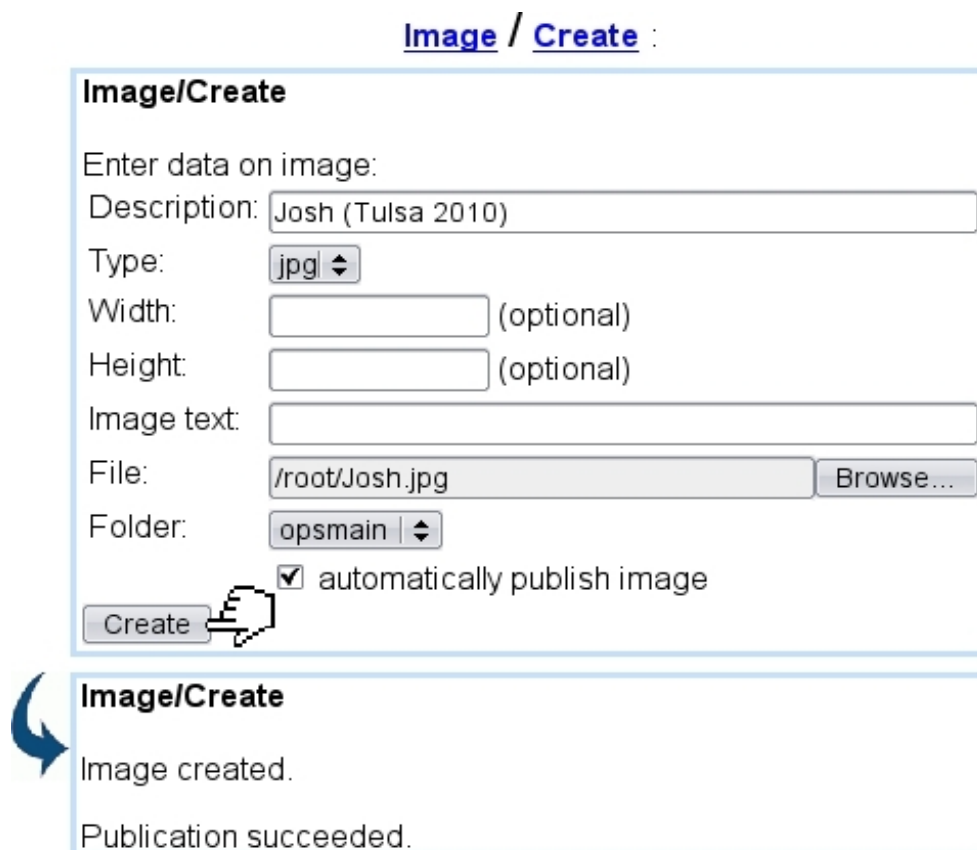


Figure 3.1: Uploading an image.

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.

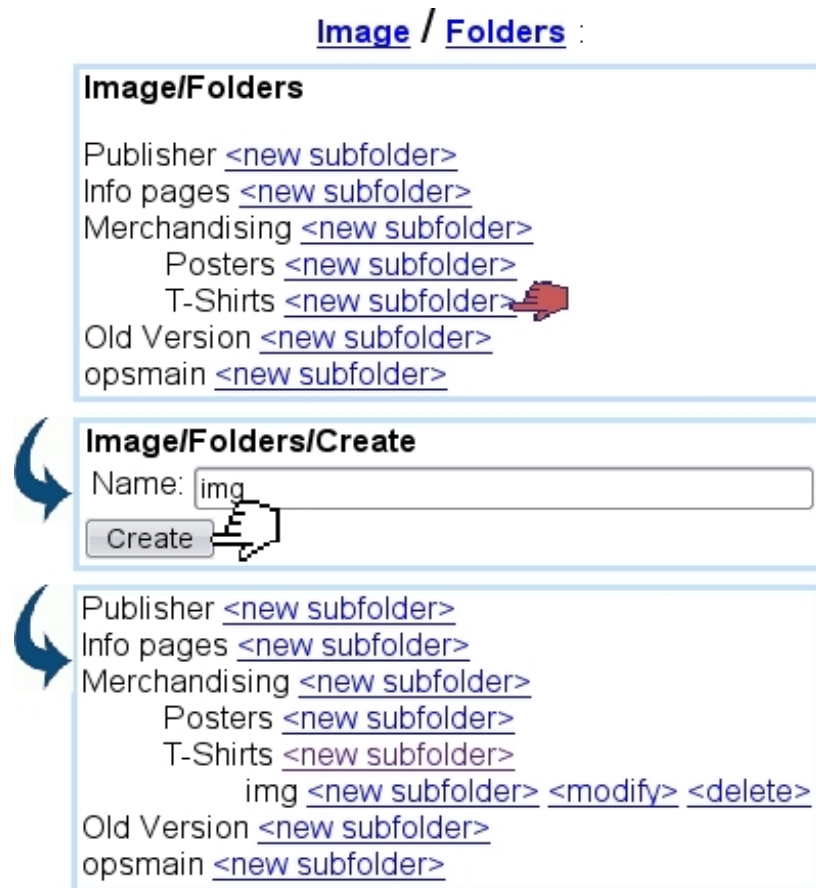



Figure 3.2: Creating a folder for images.

### 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  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

## Image / Manage :

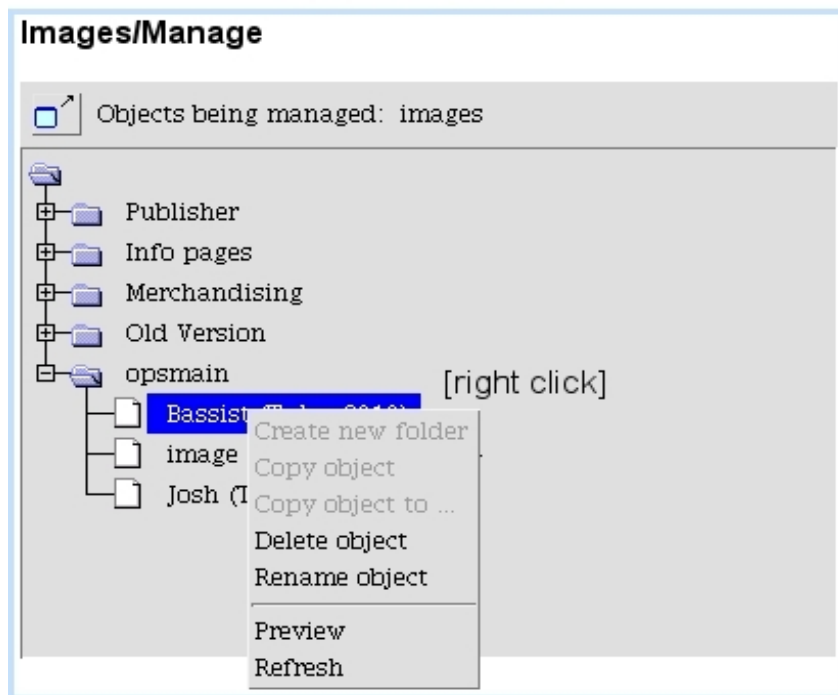


Figure 3.3: From here, all images and folders can be globally managed.

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**.


{ [Leaf Frame : Image variable edit]  :  
 { [Leaf Frame : Text variable edit]  :



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.

## Link / Create :



The image shows two parts of a web interface. The top part is a form titled "Link/Create" with the instruction "Enter data on link:". It contains two input fields: "Description:" with the value "Josh's homepage" and "Address:" with the value "http://www.myspace.com/Josh[...]". Below these are examples of link types: "external link:" with "http://www.domain.com" and "https://www.domain.com", "local link:" with "category/product.htm", and "mailto link:" with "mailto:info@domain.com". There is a checkbox labeled "use in any column of the current publisher" which is unchecked. A "Create" button with a hand cursor icon is at the bottom left. The bottom part of the image shows a confirmation message "Link created." with a blue arrow pointing from the "Create" button to it.


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 :

{ [Leaf Frame : Link variable edit]  :

{ [Leaf Frame : Text variable edit - highlighted text]  :

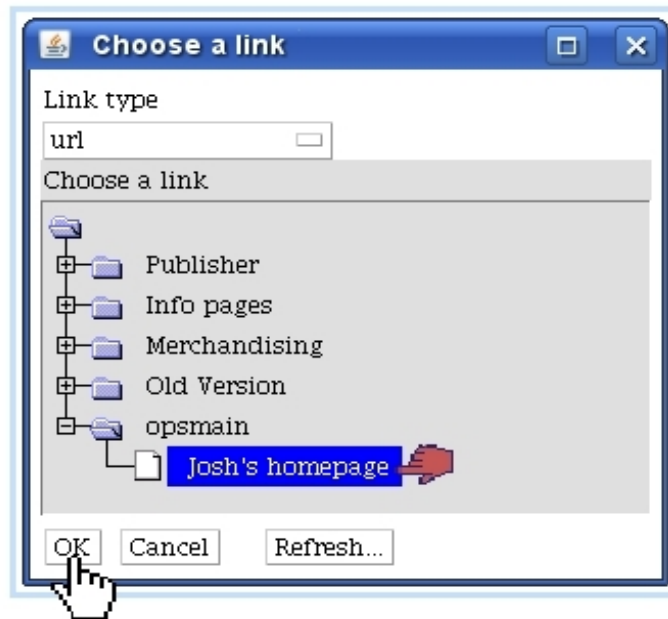
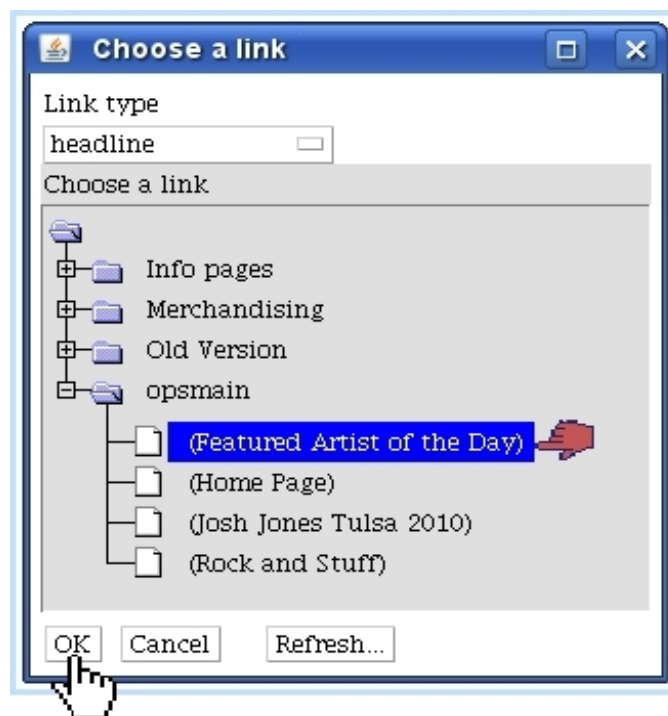


Figure 3.6: Putting a previously-defined URL link.



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](#)).



## [Scripts / Create](#) :

**Scripts/Create**

Description:

use script in any column of the current Publisher

Type:

**Scripts/Create**

Frame Edit/View Script: **Login Page**

```
<FORM NAME="ServiceLogin" METHOD="POST" ACTION="/Scripts/sql.exe">
<input type="hidden" name="Sql" value="LoginClient_Login.phs">
<input type="hidden" name="SqlDB" value="YourShopName">
<input type="hidden" name="LoginScript" value="LoginClient.phs">
<input type="hidden" name="NextScript" value="DisplayAccount.phs">
(...)

Username : <input type="text" name="_Username">
Password : <input type="password" name="_Password">
<input type="submit" value="Login">

</FORM>
```

**Scripts/Create**

Script has been created

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](#) :

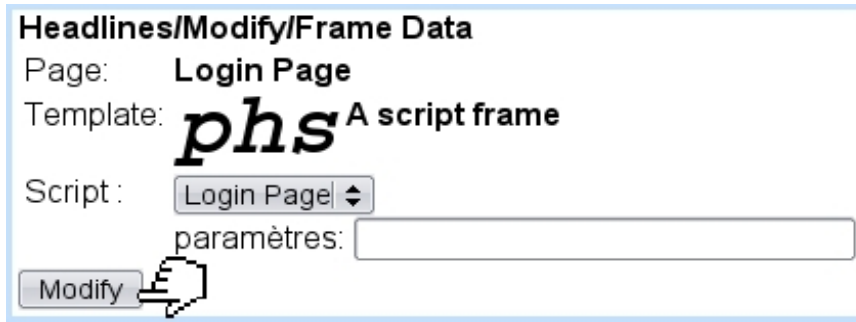
```
«INCLUDE scriptfile.phs»
```

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.

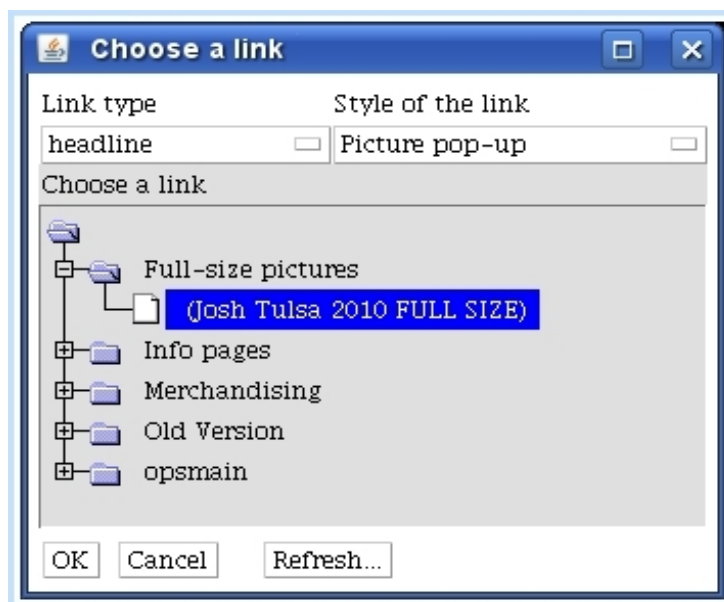
For that matter, a script 'Picture pop-up' is created/defined as follows :

[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.

**Download / Create :**

**Download/Create**

Download **TShirt\_Catalogue.pdf** has been created on the site.

Publication succeeded.

**Download/Create**

File:

Rename filename:  (optional)

Description:

Type:  |

Folder:  |

automatically publish download




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 :

[Download](#) / [Properties](#) / [Tshirt catalogue \(2011\)](#) :

**Download/Properties**

Description:

Type:

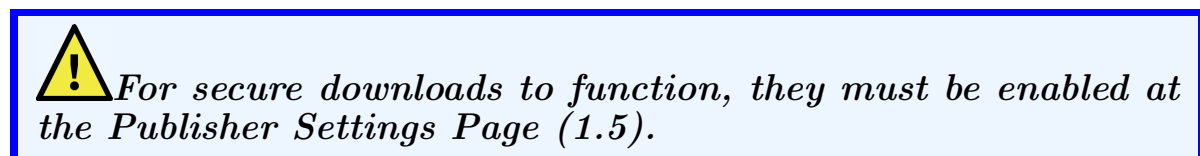
Publication Path:

Code URL as:

Creation: 28/3/2011 16:5:12

Last Modification: 28/3/2011 16:5:13

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.



### 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.

**Menu models / Create** :

**Menu Models/Create**

Name:

Rights


use model in any column of the current publisher

**Methods**

Create:

Edit Menu Level 1:

View:  Top Menu script



**Menu Models/Create**

Menu model has been created

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.

### Menu / Create :

**Menu/Create**  
Select a model:  
• [Top Menu](#)

**Menu/Create**  
Model: **Top Menu**  
Name:   
 use menu in any column of the current publisher  
Data:

**Menu/Create**  
Menu has been created

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 :

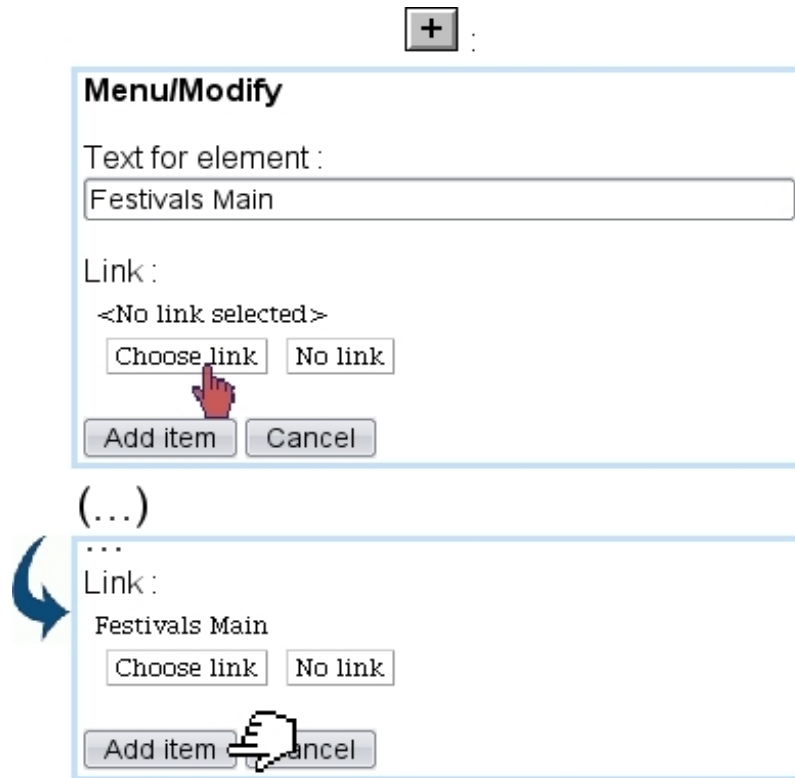
### Menu / Modify / Basic top menu :

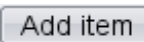
**Menu/Modify**  
**Basic top menu /**








Menu name	Url	
		<input data-bbox="1189 1702 1236 1747" type="button" value="+"/>

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 , menu item is finally added, and now a whole range of icons is available :


Menu name	Url	
Festivals	/YourPublisher/opsmain/Festivals_Main.htm	      

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 :

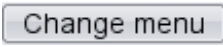
[Headline models](#) / [Modify](#) / [Menu frame](#) :

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

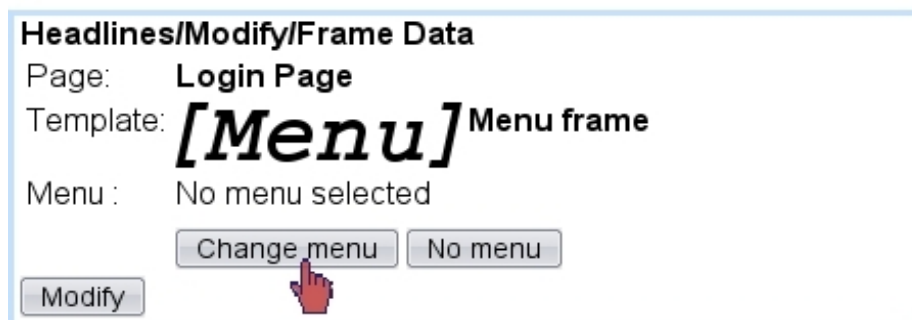
This system script handles menus, with the variable `_Menu` set as follows,

[Headline models](#) / [Variables](#) / [A script frame](#) :

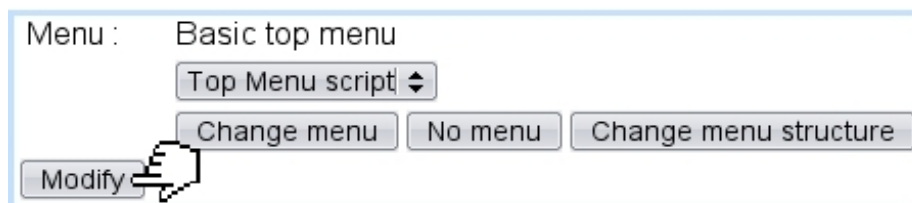
```
_Menu;Menu;12;
```

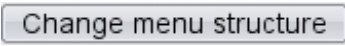
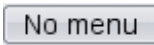
As a result, when editing this frame, a  will allow to choose a menu, here "Basic top menu" :

[Editing frame "Menu frame"] :



[After menu is selected] :



After selection of the menu, button  redirects to a page where menu items are managed, similarly as in 3.5.3.1. The  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 `SQLOUTPUTs`, 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>&nbsp;
  <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>
```

[Menu item : "Our Magazine"]  :

**Menu/Modify**

Text for element :

Link :  
 ...

**Menu/Modify**

[Menu \(extended\)](#) / *Our magazine/*

Menu name	Url	
Previous issues	...	

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 :

[Menu \(extended\)](#) / *Our magazine/*

Menu name	Url	
Previous issues	...	
Current issue	...	
Suscribe !	...	

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

[Menu \(extended\)](#) :

**Menu (extended) /**

Menu name	Url	
Home	...	
<u><a href="#">Our magazine</a></u>	...	

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.

#### 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.

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.

## Form model / Create :

**Form model / Create**

Model name:  Alignment preview :

Script type:

Action:

Label alignment:

Buttons alignment:

use model in any column of the current publisher


Alignment preview :

text text :

text :

text text :  text  text

text :



**Form model / Create**

Form model created.

Figure 3.12: Creating a form model.

## Form model / Modify / Standard form model :

...

**Parameters :**

Name	Value	
<input type="text" value="_Param1"/>	<input type="text" value="0"/>	<input type="checkbox"/> read only <input type="button" value="Create"/>

...

<input type="button" value="pencil"/> <input type="button" value="-"/>	<input type="text" value="_Param1"/>	<input type="text" value="0"/>	
	<input type="text" value="not defined"/>	<input type="text"/>	<input checked="" type="checkbox"/> read only <input type="button" value="Create"/>


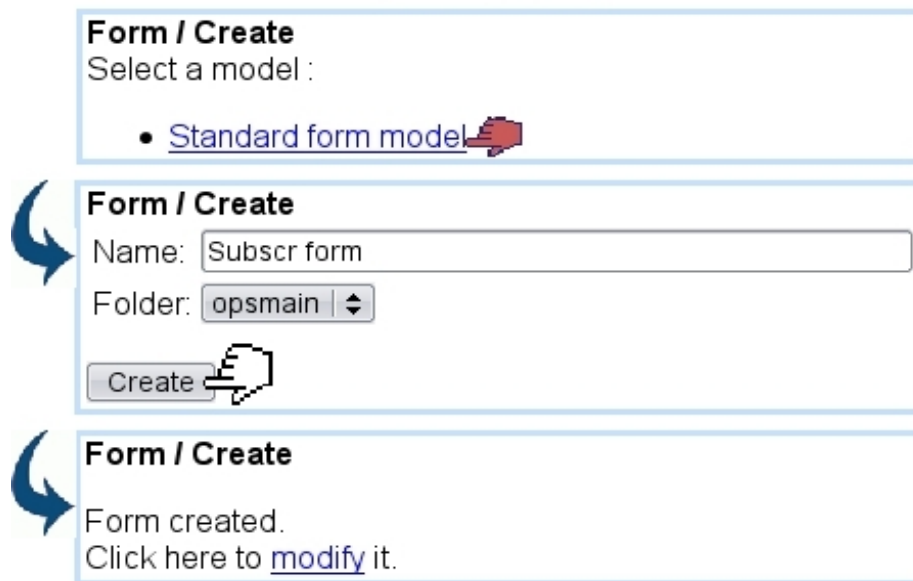


Figure 3.13: Adding parameters to a form model.

### 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 :

## Form / Create :



**Form / Create**  
Select a model :

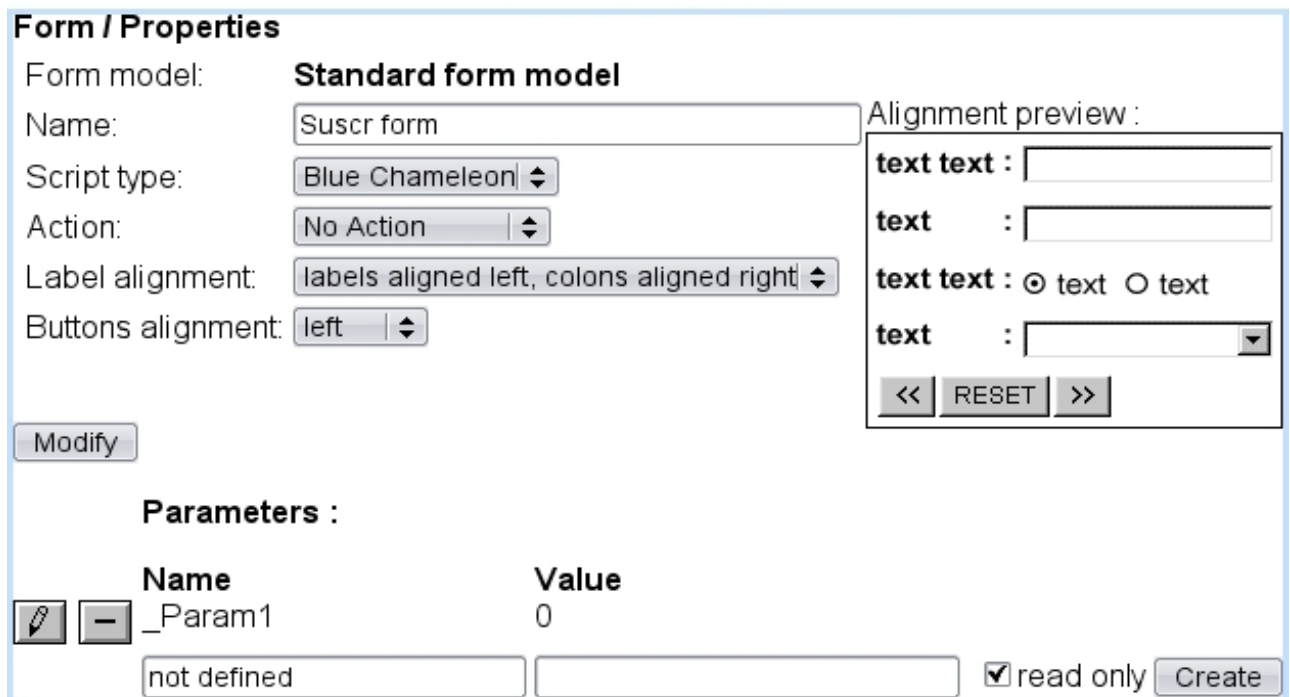
- [Standard form model](#)

**Form / Create**  
Name:   
Folder:

**Form / Create**  
Form created.  
Click here to [modify](#) it.

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

## Form / Properties :



**Form / Properties**

Form model: **Standard form model**

Name:

Script type:

Action:

Label alignment:

Buttons alignment:

**Alignment preview :**

text text :

text :

text text :  text  text

text :

**Parameters :**

Name	Value
<input type="text" value="_Param1"/>	<input type="text" value="0"/>
<input type="text" value="not defined"/>	<input type="text"/>

read only

### 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 :

**Form / Modify / Subscr form :**





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 :






Page 1  :








Done!PreviewForm : Subscr form (page 1)ResetThanks





Model : Standard form model




     

Name:

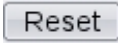
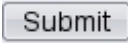
      



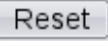
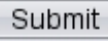





This headline-editing-style environment features, along with the usual  and  buttons, the name of the current form as well as the current page, in a greyed-out style, . If form contains several pages (3.6.3.6), as many buttons as pages would be available, each clickable to switch pages.

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  and  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 :



<b>Name</b>	<input type="text" value="element7"/>
<b>Type</b>	Radio/checkbox : <input type="radio"/> <i>radio</i> <input type="radio"/> <i>checkbox</i>
	Listbox : <input type="radio"/> <i>single selection</i> <input type="radio"/> <i>multiple selection</i> <input type="radio"/> <i>chained selection</i>
	Input field : <input type="radio"/> <i>normal text</i> <input type="radio"/> <i>password text</i>
	Text area : <input type="radio"/> <i>40 x 5</i> <input type="radio"/> <i>40 x 10</i> <input type="radio"/> <i>40 x 20</i> <input type="radio"/> <i>60 x 5</i> <input type="radio"/> <i>60 x 10</i> <input type="radio"/> <i>60 x 20</i> <input type="radio"/> <i>80 x 5</i> <input type="radio"/> <i>80 x 10</i> <input type="radio"/> <i>80 x 20</i>
	Hidden field : <input type="radio"/> <i>hidden field</i>
	Image : <input type="radio"/> <i>single image</i>
	Label : <input type="radio"/> <i>normal</i> <input type="radio"/> <i>with checkbox</i>
	Separator : <input type="radio"/> <i>10 %</i> <input type="radio"/> <i>20 %</i> <input type="radio"/> <i>30 %</i> <input type="radio"/> <i>40 %</i> <input type="radio"/> <i>50 %</i> <input type="radio"/> <i>60 %</i> <input type="radio"/> <i>70 %</i> <input type="radio"/> <i>80 %</i> <input type="radio"/> <i>90 %</i> <input type="radio"/> <i>100 %</i>
<b>Required</b>	<input type="checkbox"/>
<input type="button" value="Update"/> <input type="button" value="Cancel"/>	

Types include the following :

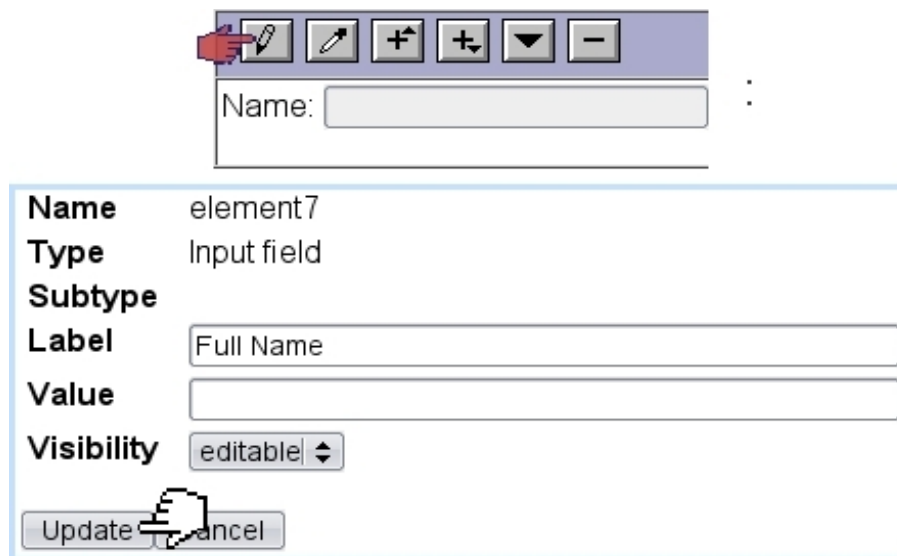
- radiobutton ;
- checkbox ;
- input text ;
- text area ;
- a hidden field ;
- an image ;
- a label ;
- a separator.

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 :

**Preview** :

Full Name (\*) :

Address (\*) :

Email :

How you discovered us :  Internet  Friend  Flier

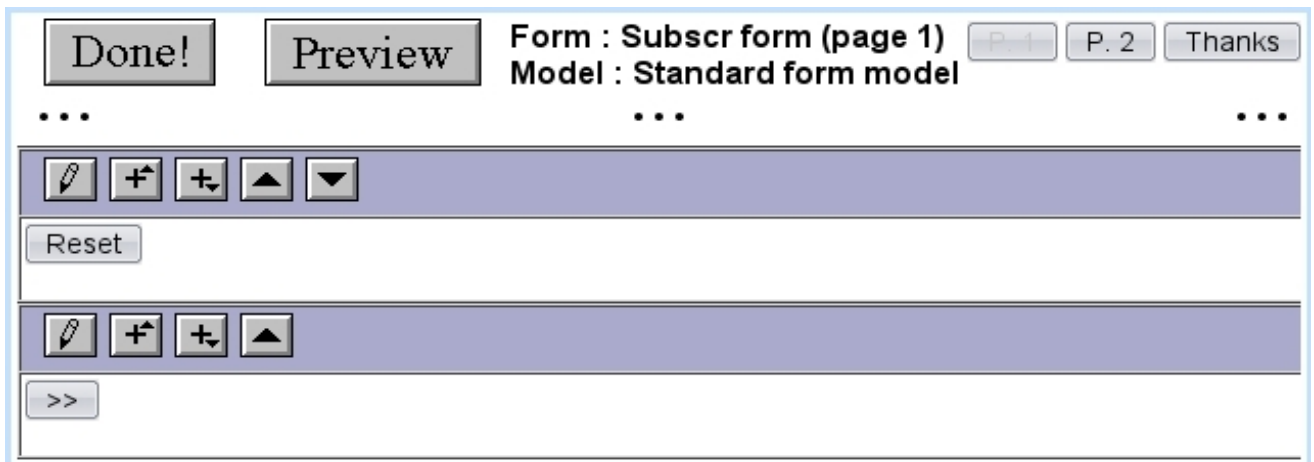
Instrument played :  None  
 Guitar  
 Bass  
 Drums  
 Kazoo  
 Other

### 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.

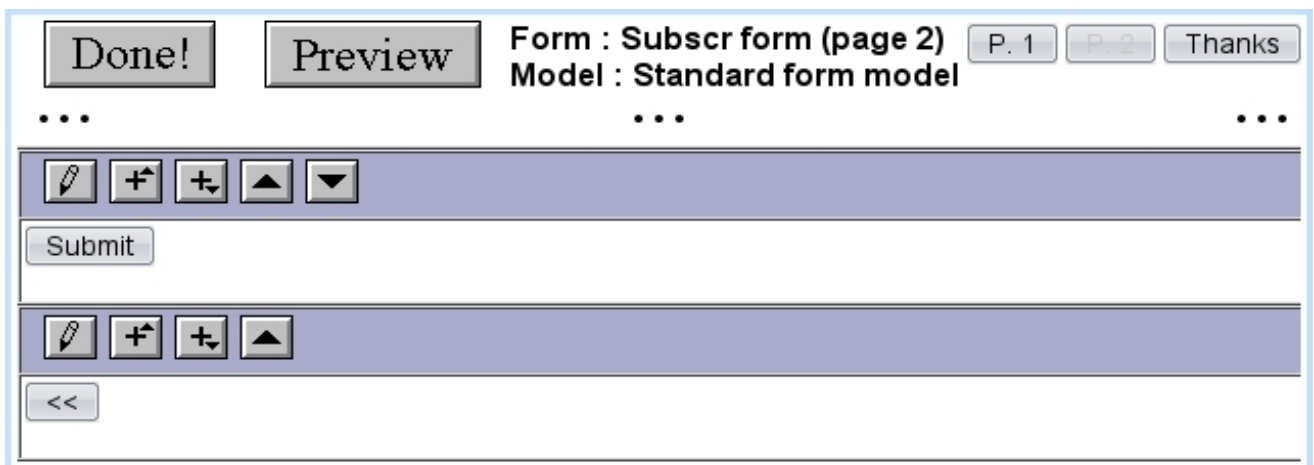
Now, need may rise of a form - with usually a large quantity of information - that features several steps (for instance, each terminated by some  button) until final submission is done. Blue Chameleon Content Management System allows to do this, by adding as many steps ("pages") as needed, thanks to 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 inputted 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 `NAMES` and `VALUES` 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).

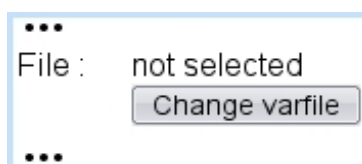
### 3.7.2 Varfiles as variables

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 :





# Chapter 4

## Other elements

This Chapter introduces components that, while quite secondary to Blue Chameleon Content Management System use, are nevertheless useful.

### 4.1 Site management



*User group rights for Site Management are detailed at 1.3.13.*

The previous ways of uploading data - images (3.1), links (3.2) and downloads (3.4) - showed a dedicated interface with the uploaded data being integrated on a leaf frame, through a variable.

Now, it is time to focus on files that can easily be used when composing a page's content, such as a Banner image (as in Code 1), a `css` style... in other words, files to be used in **contents** as accessed by [Headlines](#) / [Modify](#). They need to be easily linked to, with HTML code.

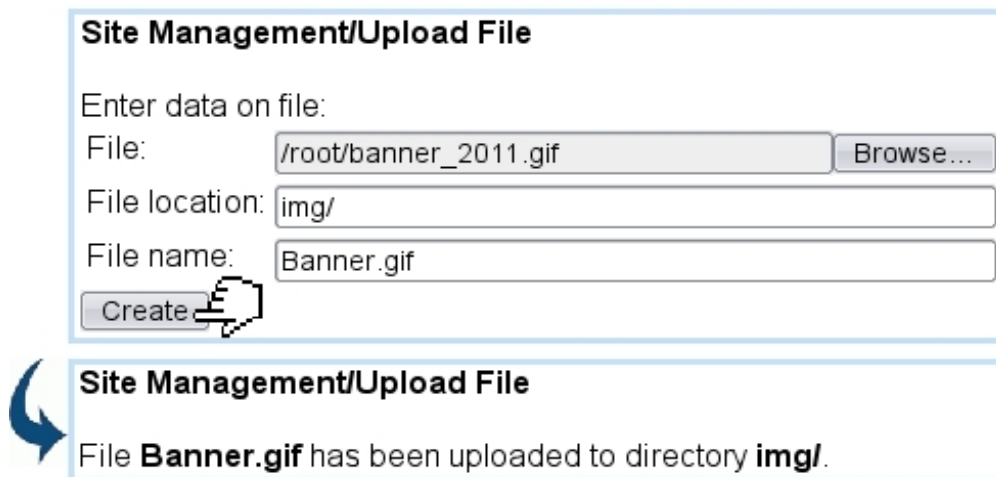
The paths to upload files, in the ways previously described, were consisting in columns, mainly `opsmain`. The uploading of files through [Site management](#) happens at the root described by the system variable `<#SQLWWWHOME#>`. From here, folders can also be created (4.1.2).

#### 4.1.1 Uploading files

Fig.4.1 shows how the image `Banner.gif` is uploaded onto the already-existing `img/` folder. There are given :

- the directory under which it will be uploaded e.g. `img/`, `css/`,... (this directory must already exist, otherwise it has to be created - see below) ;
- the full name under which it will be used (which can be different from the original file name) ;

## [Site management](#) / [Upload file](#) :



The image shows two screenshots of a web interface. The top screenshot is a form titled "Site Management/Upload File". It contains the following fields and buttons: "File:" with a text input containing "/root/banner\_2011.gif" and a "Browse..." button; "File location:" with a text input containing "img/"; "File name:" with a text input containing "Banner.gif"; and a "Create" button with a mouse cursor hovering over it. The bottom screenshot is a confirmation message titled "Site Management/Upload File" with a blue arrow pointing to the text: "File **Banner.gif** has been uploaded to directory **img/**."

Figure 4.1: Uploading a file to be used in headline contents.

When uploaded, files can then be used as in the following headline (or frame) sample, never forgetting the «#SQLWWWHOME#» root :

### [Headline models](#) / [Modify](#) / [Some headline](#) :

```
<HTML>
<HEAD>
<LINK HREF="«#SQLWWWHOME#»/css/MyStyle.css" REL="stylesheet" TYPE="text/css"
MEDIA="screen">
</HEAD>
<BODY>

«INCLUDE design.phs»

<IMG SRC="«#SQLWWWHOME#»/img/Banner.gif">

(...) <SCRIPT type="text/javascript"
src="«#SQLWWWHOME#»/js/Calculation.js"></SCRIPT>
(...)
```

## 4.1.2 Creating directories

Directories can be created to sort better pictures, styles,... to be included in headline and frame templates' contents. Fig.4.2 shows how a new directory is created.

Blue Chameleon Content Management System already has the directories `img/`, `js/` and `css/`. To create a sub-directory under an already-existing directory, the full path must be given :

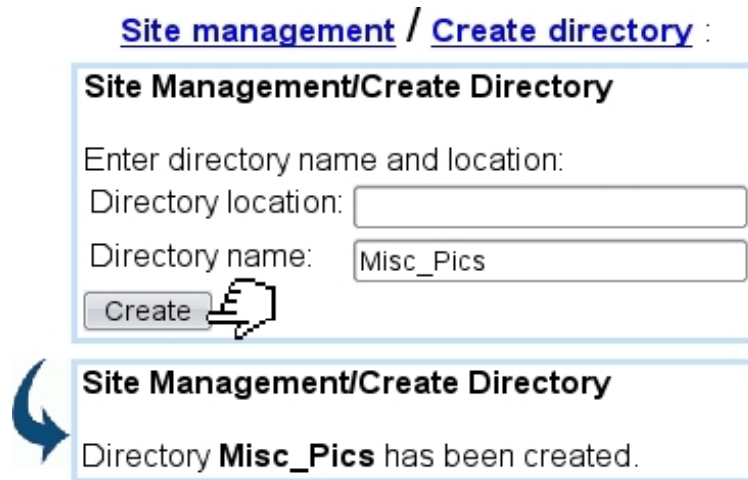
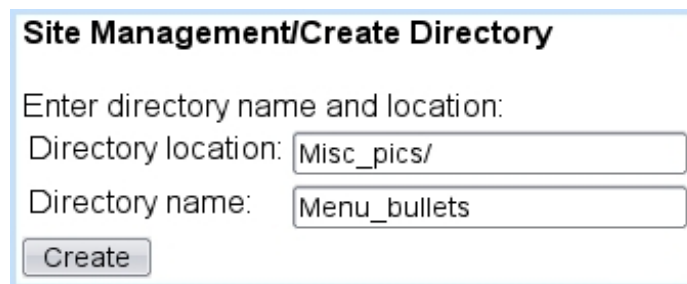


Figure 4.2: Creating a new directory, directly under the «#SQLWWWHOME#» root.



### 4.1.3 Managing files and directories

Text-based files such as scripts, css,... can be edited and saved as featured in Fig.4.3.

It is also possible to rename files with the eponymous link : file will be identified as in Fig.4.3 and a new name will be inputed for it.

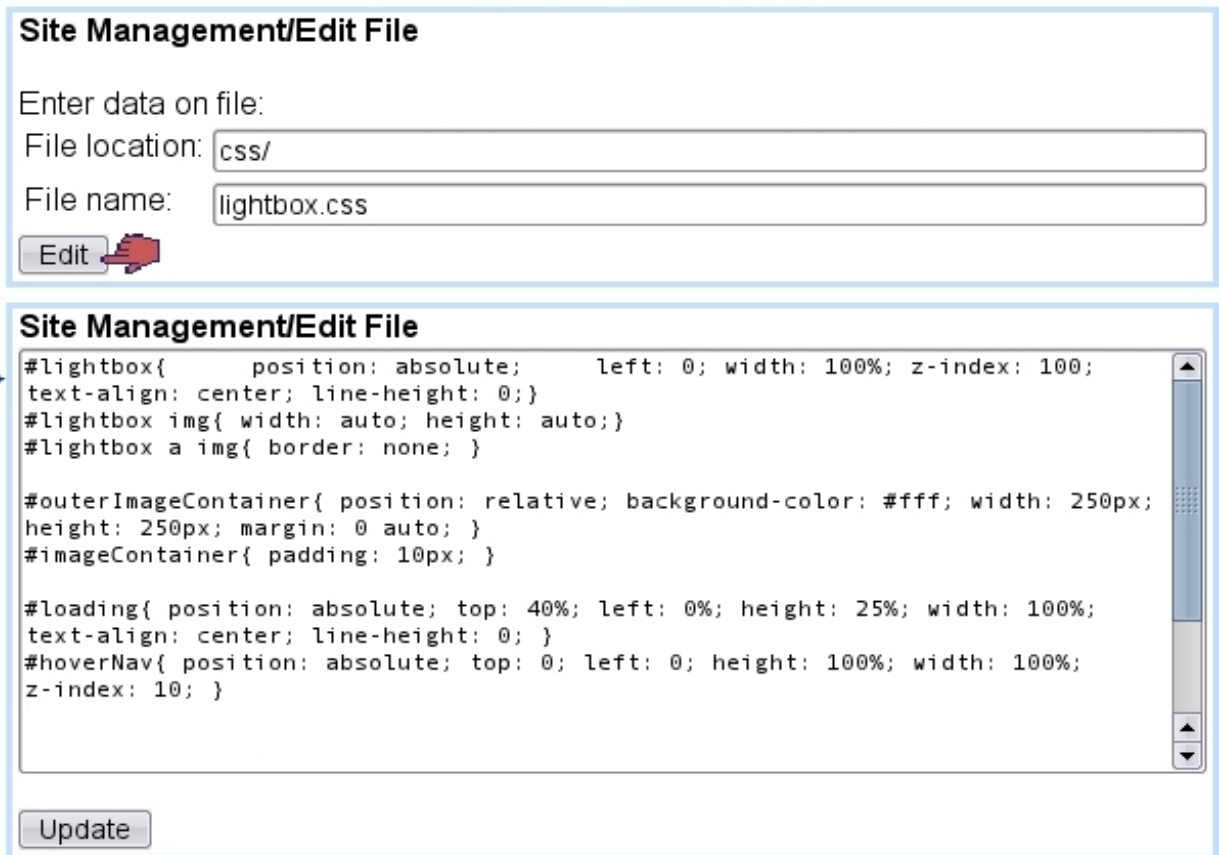
Deletions of files and directories are processed with the relevant links, with the same identification needed (location and name of file or directory).

### 4.1.4 Executing SQL

Executing SQL queries, an example of which is shown at Fig.4.4, is mainly important for Packages (5), where you have created your own tables (5.1.1). According to the nature of the query (SELECT or an other), a different button is clicked.

An example of table creation :

[Site management](#) / [Edit file](#) :



**Site Management/Edit File**

Enter data on file:

File location:

File name:

---

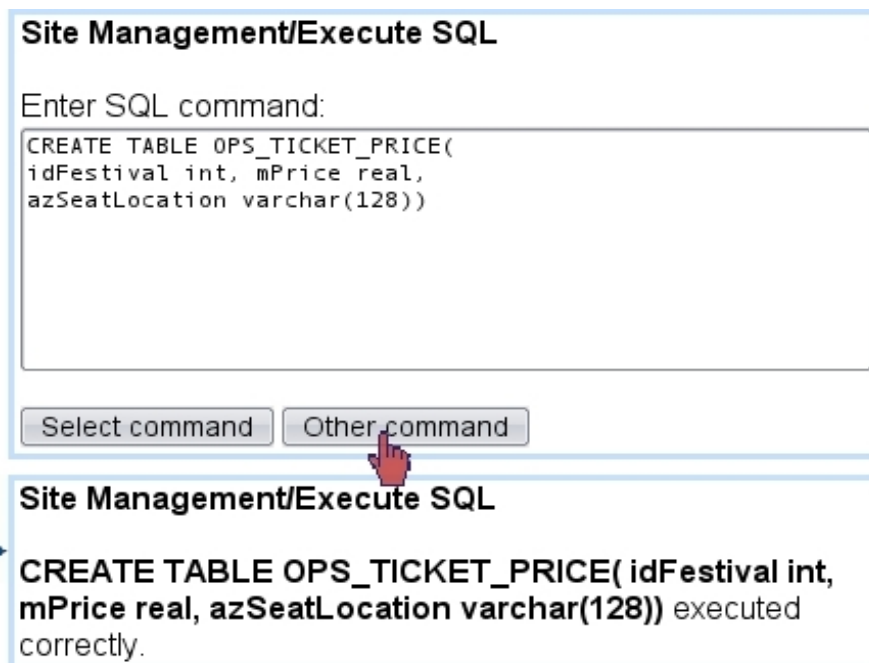
**Site Management/Edit File**

```
#lightbox{ position: absolute; left: 0; width: 100%; z-index: 100;
text-align: center; line-height: 0;}
#lightbox img{ width: auto; height: auto;}
#lightbox a img{ border: none; }

#outerImageContainer{ position: relative; background-color: #fff; width: 250px;
height: 250px; margin: 0 auto; }
#imageContainer{ padding: 10px; }

#loading{ position: absolute; top: 40%; left: 0%; height: 25%; width: 100%;
text-align: center; line-height: 0; }
#hoverNav{ position: absolute; top: 0; left: 0; height: 100%; width: 100%;
z-index: 10; }
```

Figure 4.3: Editing a text-based file.



**Site Management/Execute SQL**

Enter SQL command:

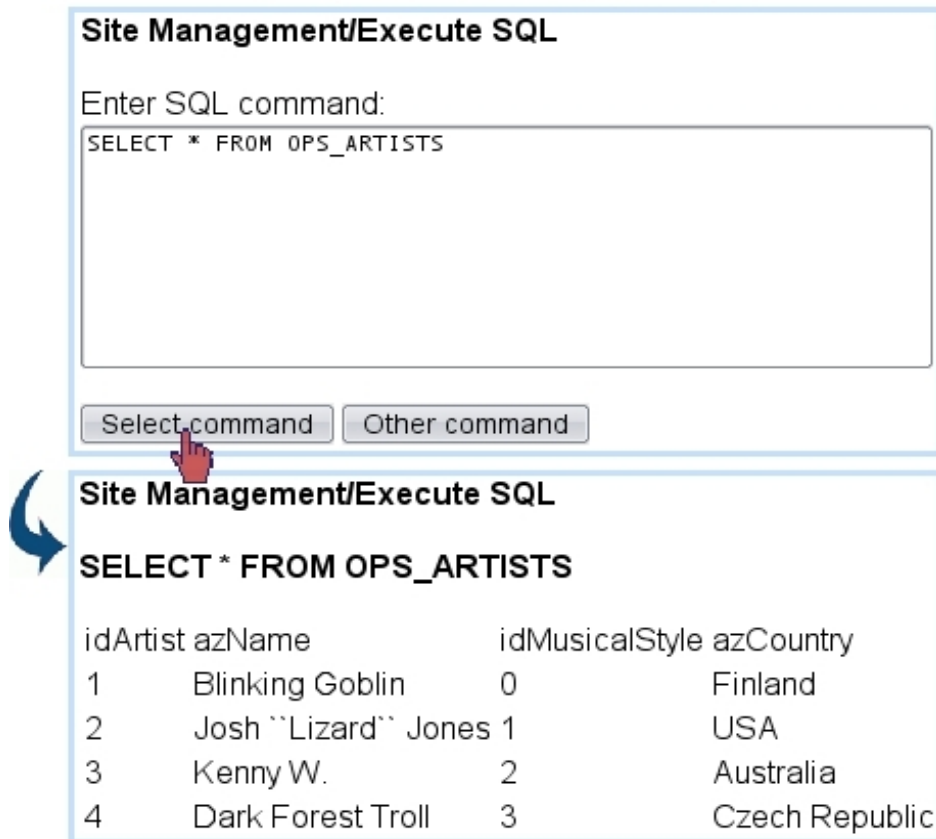
```
CREATE TABLE OPS_TICKET_PRICE(
idFestival int, mPrice real,
azSeatLocation varchar(128))
```

---

**Site Management/Execute SQL**

**CREATE TABLE OPS\_TICKET\_PRICE( idFestival int, mPrice real, azSeatLocation varchar(128)) executed correctly.**

## Site management / Execute SQL :



The screenshot shows two panels of the 'Execute SQL' interface. The top panel shows the input area where the SQL command 'SELECT \* FROM OPS\_ARTISTS' has been entered. Below the input area are two buttons: 'Select command' and 'Other command'. A red hand cursor is pointing at the 'Select command' button. A blue arrow points from this button to the bottom panel. The bottom panel shows the same interface but with the SQL command 'SELECT \* FROM OPS\_ARTISTS' displayed in bold. Below the command is a table with the following data:

idArtist	azName	idMusicalStyle	azCountry
1	Blinking Goblin	0	Finland
2	Josh ``Lizard`` Jones	1	USA
3	Kenny W.	2	Australia
4	Dark Forest Troll	3	Czech Republic

Figure 4.4: Outputting the contents of a table.

### 4.1.5 Export and Import

The Publisher in general and its constituting columns in particular can be summed up in a formatted file that describe their contents with the help of opening and closing tags, holding with them data.

The aim of those files (which are `.txt` files) is to easily **export** the data contained in a column in a form that can be **imported** by another publisher, or even from a column of the publisher to another column.

#### 4.1.5.1 Generating export files - importing them

Fig.4.5 shows how an `export.txt` file is generated.

There, for export purposes, two eponymous buttons serve as to either export the current column or the whole publisher. The file can then be seen through Blue Chameleon Content Management system via the [Edit file](#) link (4.1.3), leaving the 'File location:' blank as it is situated in the Publisher directory.

Similarly, an import of a well-formatted file can be done through the dedicated button, provided that file had been uploaded (4.1.1) under the Publisher root, i.e. with leaving



Figure 4.5: Exporting the current column.

again the 'File location:' blank.

#### 4.1.5.2 Export file contents

Fig.4.6 shows an excerpt of the contents of an `export.txt` file.

There, it can be seen that it is organized along those rules :

- `<HEADLINE>...</HEADLINE>` tags embed the whole contents of a headline, of which basic information (its identifier, its related `COLUMN`, its `NAME`) is first given ;
- contents then consist in the file information this headline is based on (2.7.2), between `<FILE>...</FILE>`: name of the `.htm` file (`DEFFILE`) as well as the publication name (`FIXEDFILE`) ;
- then, more information on the headline itself :
  - the published status (`PUBLISHED_FLAG`) ;
  - if it belongs to an `EDITION` (4.5) ;
  - its `STYLE` ;
  - the identifier of the headline `MODEL` is it based on ;
  - the date of last publication (`PUBLISHED_DATE`) given as a Unix c-day ;
  - whether it has been given an `EXPIRY` date (2.7.2).
- `<FRAMESET>...</FRAMESET>` tags then embrace the headline's contents, i.e. the frame(s) the headline is made of, each between `<FRAME>...</FRAME>` tags ;
- within those, general information upon the frame is given :
  - its `INDEX`, i.e. the value of the `_FrameIndex` variable it is called by (2.2.3.1) ;
  - what its style (2.3.4) is, i.e. whether it is `SHOWN`, has a `BORDER` and has extended edition (`EXTEDITION`) ;
  - the identifier of the frame `MODEL`.



export.txt :

```
...
<HEADLINE>
NEW=1
COLUMN=1
ID=15
NAME=Home Page
<FILE>
DEFFILE=20110323143529_16608.htm
FIXEDFILE=Home_Page.htm
</FILE>
PUBLISHED_FLAG=0
EDITION=0
STYLE=0000
MODEL=44
PUBLISHED_DATE=40638
EXPIRY=0
<FRAMESET>
<FRAME>
INDEX=1
SHOWN=1
BORDER=1
EXTEDITION=1
MODEL=49
<VARFILE>
NAME=20110323145244_16737.var
<VAR>
NAME=_Text
VALUE=Blah
</VAR>
</VARFILE>
</FRAME>
...
<FRAME>
...
</FRAME>
</FRAMESET>
</HEADLINE>
...
```

Figure 4.6: An example of export file contents.

- then, in between `<VARFILE>...</VARFILE>` tags, the various variables for this frame are to be given, after the `NAME` of the related varfile is given. Variables are given as `NAME` and `VALUE` couples, in between `<VAR>...</VAR>` tags (see 3.7.1 for a detailed example of a varfile) ;
- as a frame contains only one varfile, the closing `</VARFILE>` tag is followed by the `</FRAME>` tag ;
- as a headline may contain several frames, the last of the closing `</FRAME>` tag(s) is followed by the closing `</FRAMESET>` tag, which is followed by the closing `</HEADLINE>` tag ;
- there are as many `<HEADLINE>...</HEADLINE>` tags as there are headlines in the

current column or Publisher.

## 4.2 Model defaults

A model default can be defined for Form models (3.6.1) and headline models (2.2).

At Fig.4.7, the creation of a model default aimed at headline model "Bk background" is illustrated.

**Model Defaults / Create :**

**Model Defaults/Create**

Enter Data on Model Defaults:

Description:

Image:

You must select an image if you are defining a headline model default.  
You cannot select an image if you are defining a form model default.

Model: You must must choose one model, either a headline or a form model.

Form:  ▾

Headline:  ▾




Figure 4.7: Creating a model default for a headline model.

### 4.2.1 Model defaults and headlines

This model default can be afterwards associated with a headline, picked amongst a list of headlines based upon the headline model that was used.


Here, for model default "Bk model default bkgnd", using headline model "Bk background", we use a headline ("bk mod def headline") that was defined using that headline model. This setting can be done right after model default creation, or modification :

## Model Defaults / Modify :

### Model Defaults/Modify

Description:

Image:

Headline:  

[Click here](#) to limit the usage of the model default to certain columns of the publisher.



## 4.3 Books

### 4.3.1 Book models



*User group rights for managing Book Models are detailed at 1.3.23.*



*To be successfully created, book models necessitate the following to be defined :*

- *'book' headline models (2.2.1) ;*
- *and fully-configured model defaults (4.2) for those headline models.*

Creating a new book model is featured at Fig.4.8.

There, apart from the name, picture of the book model and whether it is to be used in any column, book-type headline models are selected for the background and foreground.


After creation, a book model can be updated (name, picture, background/foreground models) or removed via [Model Defaults / Modify Book Model](#), [Delete Book Model](#).

### 4.3.2 Creating books



*User group rights for managing Books are detailed at 1.3.22.*

## Books / Create Book Model :



**Book/Create Book Model**  
Enter Data on Book Model:  
Description:   
Image:    
Background Model:   
Foreground Model:   
 use book model in any column of the current Publisher  
 

**Book/Create Book Model**  
Book model has been created.

Figure 4.8: Creating a book model.

Once at least a book model has been created as described above, books can be created ; an example is featured at Fig.4.9.

## Books / Create :

**Book/Create**  
Enter data on book:  
Description   
Model   Basic book model  
 

**Book/Create**  
Book has been created.

Figure 4.9: Creating a book.

This action also creates, in the current column two new headlines "Book - Main Book - Background" and "Book - Main Book - Foreground".

In this example, background and foreground headline models have been defined as follows :

[Headline models](#) / [Modify](#) / [Bk background](#) :

```
<style type="text/css">
  body {background-color:#AEABA0;}
</style>
<i>[This is Headline Model "Bk background"]</i>
```

[Headline models](#) / [Modify](#) / [Bk foreground](#) :

```
[Contents for Headline Model "Bk foreground"...]<br>
<b>[blah blah blah]</b>
```

The results of those simple book background and foreground models can be seen on the Headlines page :

[Headlines](#) / [Preview](#) :

[Book - Main Book - Background](#) :

*[This is Headline Model "Bk background"]*




[Book - Main Book - Foreground](#) :

[Contents for Headline Model "Bk foreground"...]  
**[blah blah blah]**

### 4.3.3 Modifying a book

Just after it is created, a book only contains a background and foreground, as associated to its model.

Via [Modify](#), it becomes then possible to perform actions with sets of various icons (Fig.4.10). On this page, elements (including background and foreground) are called "menus".

-  : displays a preview of the element, just as the previews for Background and Foreground as displayed above. For "Background", it is the only icon available ;
-  ,  : allow to add a menu or submenu element before the previous element or after the next ;

## Books / Modify / Main book :










Book/Modify			
Book: Main book			
Options	Menu	SubMenu	Menu Name
	0	0	Background
  	1	0	Foreground - ( <a href="#">change name</a> )
			

Figure 4.10: From here, book elements can be managed.

-  : allows to add a submenu under the menu (not available for submenus) ;
- ,  : with more than one menu (Background and Foreground excepted - icons not available for those) or submenu, allows to displace element before the previous one (if not the first in the list) or after the next one (if not the last in the list) ;
-  : removes any element that is not Background or Foreground.

## 4.4 Notice boards



*User group rights for handling Notice Boards are detailed at 1.3.16.*

Notice boards gather a collection of modifications that have been brought a site ; a notice board is associated with a column (1.4) and can be published on a headline as a list of links to every object that has been updated.

### 4.4.1 Creating and managing notice boards

An example creation of a notice board is featured at Fig.4.11 ; the maximum number of days modifications remain in the notice board is given. Then, it is assessed, amongst a list of headlines in all columns, a specific headline or none at all.

After their creation, notice boards can be modified or deleted thanks to the eponymous links.

### 4.4.2 Association with a column

For a notice board to be used in the context of a column, the latter must be updated to specify it, in the following way :

[Notice board / Add entry](#) :

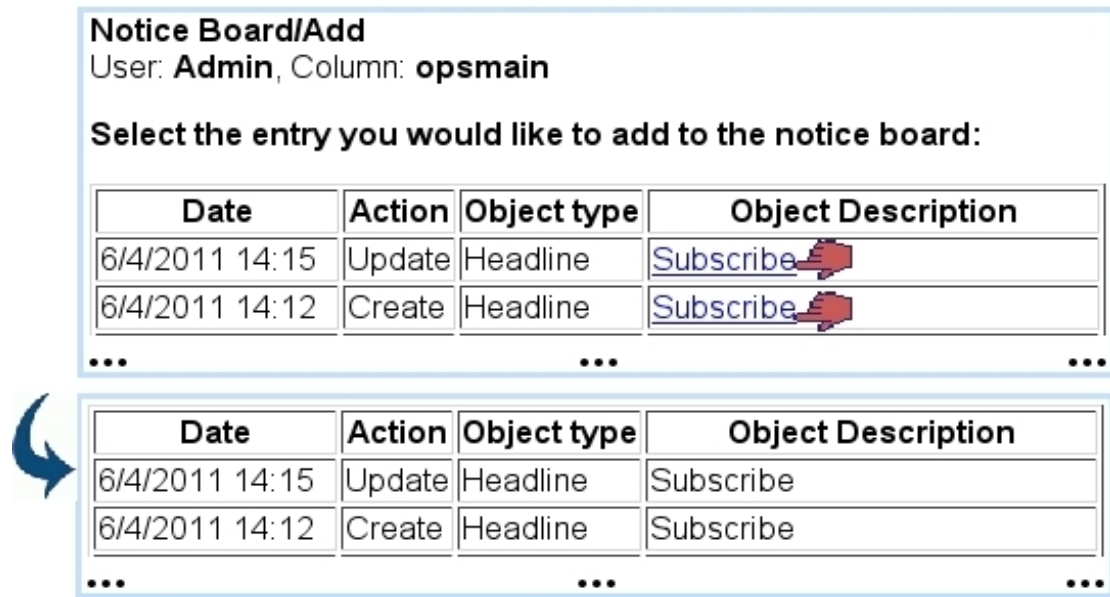
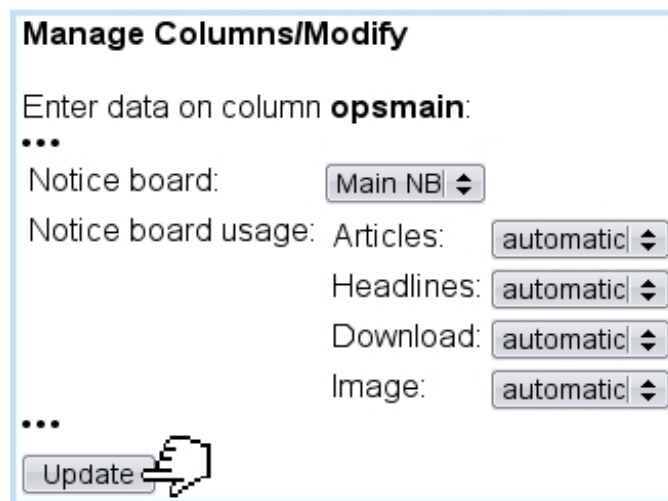


Figure 4.11: Creating a general notice board.

[Columns / Modify / opsmain](#) :



Menus there also allow to define individually, for articles, headlines, downloads and images, whether entries are done manually or automatically.

### 4.4.3 Notice board entries

If the notice board usage menus as seen in the previous images have been set 'automatic', this nonetheless does not prevent user from adding entries manually, as featured in Fig.4.12.

There, a table with all the latest modifications on the associated column's headline(s) is shown ; these are click-links, that, when clicked, add the modification to the notice

board, rendering it in normal font.

[Notice board / Add entry](#) :

**Notice Board/Add**  
User: **Admin**, Column: **opsmain**

**Select the entry you would like to add to the notice board:**

Date	Action	Object type	Object Description
6/4/2011 14:15	Update	Headline	<a href="#">Subscribe</a> 
6/4/2011 14:12	Create	Headline	<a href="#">Subscribe</a> 
...		...	...



Date	Action	Object type	Object Description
6/4/2011 14:15	Update	Headline	Subscribe
6/4/2011 14:12	Create	Headline	Subscribe
...		...	...

Figure 4.12: Manually adding two entries to this notice board.

Entries can be removed from the notice board via [Delete entry](#).

## 4.5 Editions



*User group rights for Edition handling are detailed at 1.3.10.*



***When creating a column (Fig.1.6), the Administrator can indicate whether the new column will be working in edition mode. To activate this mode for an already-existing column, s/he has to create a default edition for the column in the Blue Chameleon database.***

The Edition mode is dedicated to publishers who wish to update their website and a periodical basis (e.g. a magazine that publishes every month an extract of their articles from the current paper edition).

In this mode, Blue Chameleon CMS supports different versions of a set of pages in the same column. These sets are stored in Editions (e.g. the index page leading to the article extracts will exist in editions January, February, etc.). When opening up a new



edition, all the headlines of the current column are copied to the new column.

Only documents from the active edition can be published. In order to replace the current website with the new edition, the new edition must be marked as 'active edition' and then be published.

For instance, let us suppose a column has been created with "work with editions" ticked with "April" as the name of the first edition ; with this column being set as the current one, the following can be seen :

#### [Edition](#) :

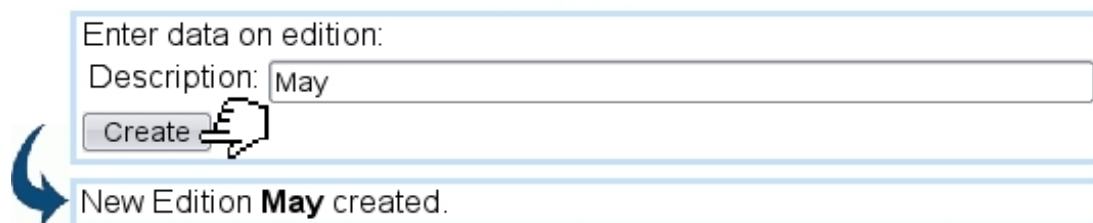
**Edition:** [Modify](#) - [New](#) - [Set active](#) - [Delete](#) (Current: April) (Active: April)


This current edition name, as well as any other, can afterwards be updated through [Edition](#) / [Modify](#).

### 4.5.1 General edition management

The process of creating a new edition is simple :

#### [Edition](#) / [New](#) :



Enter data on edition:  
Description:   
 

New Edition **May** created.

Once more than one edition exist in the system, any edition can be set to 'current' viz. 'active' via [Edition](#) / [Change](#) viz. [Set active](#).

The [Delete](#) link also allows to remove any edition that is not current or active.

On the *Headline Modify Page*, the following display will be shown :

#### [Headlines](#) / [Modify](#) :



Select a page:  
..... **April** ..... [May](#)

While the current edition is displayed in bold, with its headlines further down, any other edition (e.g. [May](#)) can be clicked to alter its contents.

## 4.5.2 Publishing editions

Through the [Edition](#) / [Publish](#) path, the following is available :

### **Edition Publish**

Click on 'Edition Publish' to publish all objects that have been modified.

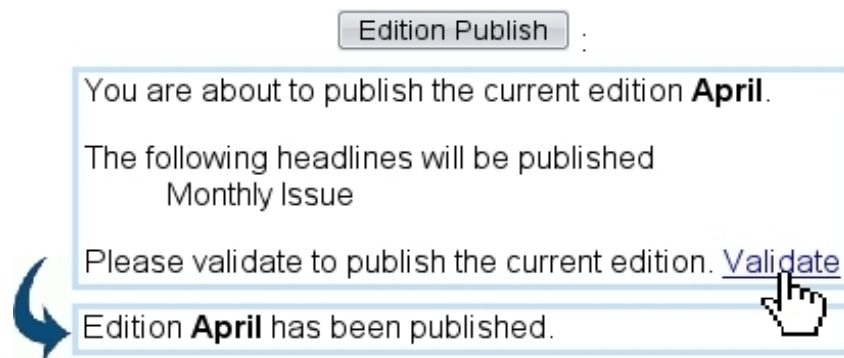
### **Detailed Publish**

Click on 'Detailed Publish' to have the possibility to individually check the objects to be published.

There :

- clicking on the  button will display a simple list of all objects that have been modified, with the only choice to validate their publication ;
- clicking on the  button, on the other hand, will provide check-boxes to select headlines, articles,... and to check objects, exactly as in Fig.2.9.

The following shows for instance (as current and active editions are set to "April") the former process :



## 4.6 Options

This part of Blue Chameleon Content Management System, as accessed via [Options](#) is aimed at giving various information.

### 4.6.1 Version

This link allows to check the current versions of Blue Chameleon Content Management System and OSL, user's IP address as well as the versions (5.2.2) of all packages in use :

## Options / Version :

**Options/Version**  
**Online Publishing System Version 2.27**  
Scripting Language Version 390  
Your IP address is '10.1.1.37'.  
**Packages**  
Agenda version 1  
Documents version 1  
Festivals version 2  
News version 1  
Photo album version 2

### 4.6.2 System Logs



*How actions are recorded and how long records are kept are defined at the Publisher settings (1.5).*

Blue Chameleon Content Management System logs in the activity of its users. The system log contains information on the following actions :

- the modification, creation, deletion and publication of articles headlines, images and downloads ;
- the setting of ready-for-publication flag and the launching of scheduled publications ;
- the FTP deletion and FTP publication of articles, headlines, images and downloads ;
- the cancellation of FTP deletion and FTP publication ;
- the creation, modification, and deletion of users and user groups ;
- the users that have logged to Blue Chameleon.

As featured in Fig.4.13, this page provides several menus to narrow searches :

- user(s) of whom activity will be displayed : one's own log only, one's current group, logs of all groups and everything.



*User group rights for displaying System Logs user-wise are detailed at 1.3.17.*

- on which column the activity took place (or all columns) ;

- when this activity took place (the current day, or for the 2, 7 or 30 days) ;
- the kind of activity, related to content management or system (i.e. the managing of users, groups...) - or any.

### Options / System Log :

**Options/System Log**

**User**                      **Column**                      **Period**                      **Type**  
 User's own log ▾   All available columns ▾   Last 7 days ▾   Any type ▾  

**YourPublisher**

Date	Column	User	Action	Object type	Object Description
4/4/2011 9:32	Publisher	Admin	Update	Group	Administrator
4/4/2011 9:01	opsmain	Admin	Logged in		
...					
1/4/2011 11:49	opsmain	Admin	Publish	Headline	Rock and Stuff
1/4/2011 11:49	opsmain	Admin	Publish	Headline	Login Page
1/4/2011 11:49	opsmain	Admin	Publish	Headline	Josh Jones Tulsa 2010
...					
1/4/2011 8:43	opsmain	Admin	Create	Headline	bk mod def headline
...					

Figure 4.13: Checking one's own activity for last week.

### 4.6.3 Expiration report

The eponymous link leads to a page where objects that have expired or are going to are listed ; by default, what decides if objects related to a column will expire is the setting as done when updating this column (1.4.3.2), which can then be fine-tuned for any headline on its Properties page (2.7.2).

## 4.7 Searching for content



*User group rights for performing searches are detailed at 1.3.5.*

Blue Chameleon Content Management System also offer a search tool that allows, as pictured in Fig.4.14, to perform searches in four different ways :

- by the name contents are published under ;

- by the publication folder (search then returns results by categories - headlines, images, etc.) ;
- by the object description ;
- by cumulating criteria object type, column and to/from dates (for which choice is creation, update and publish).

[Search / Search :](#)

**Publisher - Search**

**Search by Publication Name**  
 Publication Name:

**Search by Publication Path**  
 Publication Path:

**Search by Object Description**  
 Object Description:

**Search by Multiple Criteria**  
 Object type:    
 Section:    
 Date: from  /  /  to  /  /  Type

**Search by Publication Name**

**Headlines:**

Rubrique	Folder Name	Description	Last Publication	Last Modification
opsmain	Josh Jones Tulsa 2010	Josh Jones Tulsa 2010	1/4/2011	24/3/2011
Full-size pictures	Josh Tulsa 2010 FULL SIZE	Josh Tulsa 2010 FULL SIZE	4/4/2011	No date available

Figure 4.14: Performing a search on objects which publishing name begin by 'Josh'.

## 4.8 Management of objects

This page, as featured in Fig.4.15, offers the same tree-view structure and management functionalities than were already seen for images (3.1.3), downloads (3.4.2), headlines (2.7.3),... but for *any object*, as selected in the menu on the top.

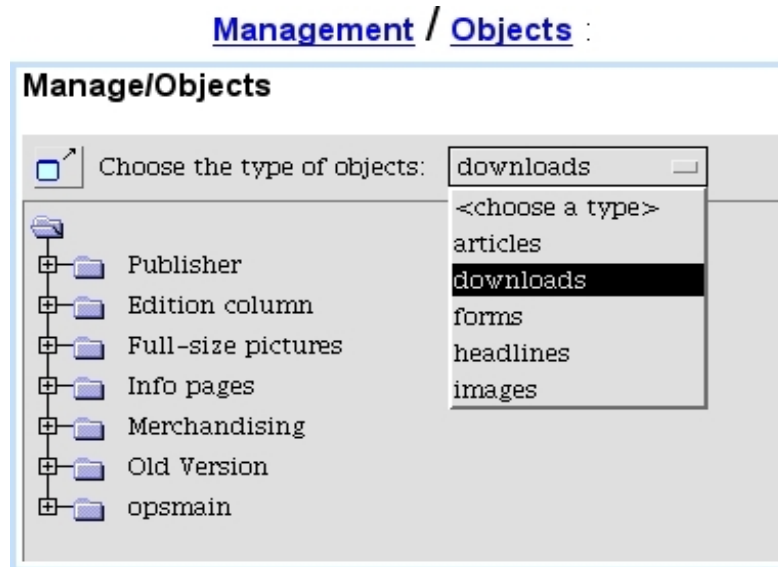


Figure 4.15: From here, any object can be managed.

# Chapter 5

## Packages

Through use of Blue Chameleon Content Management System, the need for extended functions might arise sooner or later, requiring data to be uploaded, processed and managed. This implies the enabling of database capabilities as well as more complex script features.

A package is composed of several `.phs` scripts, which make extended use of Blue Chameleon's own language, OSL ; if you are unfamiliar with it, please check OSL's User Guide.

Amongst the scripts that constitute a package, most of them aim at managing the package itself and its objects inside the system. They have to be integrated for the package to properly function.



*User group rights for managing Packages are detailed at 1.3.28.*

### 5.1 Mandatory scripts

What follows describes the scripts that **must** be included into your package for it to function properly. A package is, as an end result, a library compiled from those scripts, as similarly described in **Developing Your Blue Chameleon Add-On**.

This library will then be uploaded in your system and its features will be available for page composing.

#### 5.1.1 PackageInstall.phs

This script contains the necessary instructions to register and update the package inside the system, according to a version variable. It contains the SQL table creation instructions to execute the first time and, afterwards, table updates.

An example is given at Code 6. There :

- the `_PkgeId` variable as given at the beginning identifies your package and must be unique for each package ;
- things are done according to variable `piPackageVersion` (which value is known whenever this script is called) and, according to their success (as given by variable `_Ok`, which can be set through evaluations of `<<#SQLSTATUS#>>es`), the value of `piPackageVersion` is incremented. See "Updating a package" (5.2.2) for more details.

## 5.1.2 PackageGroupModify.phs, PackageGroupModify1.phs

These scripts aim at respectively configuring the group access rights (1.3) for the package and modifying them. They are featured at Code 7 and 8.

In the first, a FORM is generated : one of the CGI variables, `_Script`, is set to `YourPackageName:PackageGroupModify1.phs` and a group right is selected through a menu (which options can be expanded if needed). In the second, the update is propagated to the OPS table that manages user groups.

### 5.1.2.1 Result : package group rights

By accessing the group user rights for a package, what is shown in Fig.5.1 will be then available : a list of uploaded packages with for each a possibility to [Modify rights](#), in the layout as defined by script `PackageGroupModify.phs`.

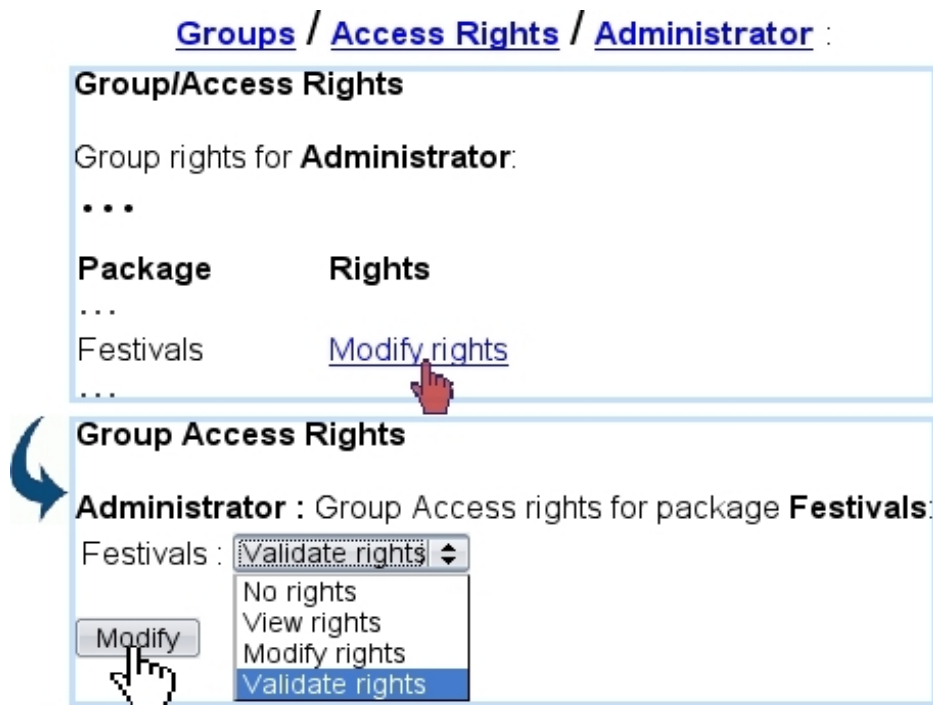


Figure 5.1: Modifying this package's rights for a particular user group.



---

**Code 6** A template for PackageInstall.phs.

---

```
< * Parameters:
  * piPackageVersion, currently installed version of the package
  * iPackagePublisherId, Id of publisher for which package is to be installed
  * >

<LET _PkgeId=150>
<VAR NEW _CptError=0>
<VAR NEW _Ok=1>

<IF <piPackageVersion><1>  <* Create tables+indexes *>
  <SQLSTATUS TRANSACTION BEGIN>

    <SQL CREATE TABLE OPS_FESTIVALS(
      idFestival int, azName varchar(128),
      azCity varchar(64), iDate int, idArtist int,
      iOrder int)>

    <SQL CREATE TABLE OPS_ARTISTS(
      idArtist int, azName varchar(128), idMusicalStyle int, azCountry
      varchar(64))>

    (Create some other tables...)

    <SQLSTATUS TRANSACTION END>
  <ENDIF>

  <DIRECTORY CREATE images/artists>
  (Create some other directories...)

  <IF <piPackageVersion><2>
    <* Upgrade package to version 1*>
  <ENDIF>

  <IF _Ok==1>
    <LET _PackageVersion=1>
  <ENDIF>
```

---

---

**Code 7** A template for PackageGroupModify.phs.

---

```
«SQLEXEC STRING _GName=select azGroupName from «#SQLWPA#»OPSP_GROUPS where
iGroupId=«iPackageGroupId»
<BR><STRONG>«_GName»:</STRONG> Group Access rights for package
<STRONG>«azPackageName»</STRONG>:

<FORM METHOD="post" ACTION="/Scripts/sql.exe">
  <INPUT TYPE="hidden" NAME="SqlDB" VALUE="«SqlDB»">
  <INPUT TYPE="hidden" NAME="Sql" VALUE="opsPackage.phs">
  <INPUT TYPE="hidden" NAME="_Script" VALUE="Festivals:PackageGroupModify1.phs">
  <INPUT TYPE="hidden" NAME="xid" VALUE="«xid»">
  <INPUT TYPE="hidden" NAME="iPackageGroupId" VALUE="«iPackageGroupId»">
  <INPUT TYPE="hidden" NAME="iPackageId" VALUE="«iPackageId»">
  <INPUT TYPE="hidden" NAME="_OPSMenu" VALUE="«_OPSMenu»">

  «VAR NEW _GroupRights»
  «SQLEXEC STRING _GroupRights=SELECT azGroupRights FROM «#SQLWPA#»OPSP_GROUPS
WHERE iPublisherId=«iPackagePublisherId» AND iGroupId=«iPackageGroupId» AND
iPackageId=«iPackageId»»

  <TABLE>
  <TR>
  <TD>Festivals :</TD>
  <TD>
  <SELECT NAME="pos0">
  <OPTION VALUE="48"«IF _GroupRights[0]==48»SELECTED«ENDIF»>>No rights</OPTION>
  <OPTION VALUE="49"«IF _GroupRights[0]==49»SELECTED«ENDIF»>>View rights</OPTION>
  <OPTION VALUE="50"«IF _GroupRights[0]==50»SELECTED«ENDIF»>>Modify rights</OPTION>
  <OPTION VALUE="51"«IF _GroupRights[0]>=51»SELECTED«ENDIF»>>Validate
rights</OPTION>
  </SELECT>
  </TD>
  </TR>
  </TABLE>

  <BR><INPUT TYPE="submit" VALUE="Modify">
</FORM>
```

---

---

**Code 8** The script PackageGroupModify1.phs.

---

```
«VAR NEW _GroupRights="00000000000000000000"»

«SQLEXEC STRING _GroupRights=SELECT azGroupRights FROM «#SQLWPA#»OPSP_GROUPS
WHERE iPublisherId=«iPackagePublisherId» AND iGroupId=«iPackageGroupId»»
«SQLEXEC STRING _GName=SELECT azGroupName FROM «#SQLWPA#»OPSP_GROUPS WHERE
iGroupId=«iPackageGroupId»»
«LET _GroupRights[0]=@int(pos0)»




«SQL UPDATE «#SQLWPA#»OPSP_GROUPS SET azGroupRights='«_GroupRights»' WHERE
iPackageId=«iPackageId» AND iPublisherId=«iPackagePublisherId» AND
iGroupId=«iPackageGroupId»»
«IF #SQLSTATUS#<>1»
  <BR>ERROR: Command 'UPDATE «#SQLWPA#»OPSP_GROUPS SET
  azGroupRights='«_GroupRights»' WHERE iPackageId=«iPackageId» AND
  iPublisherId=«iPackagePublisherId» AND iGroupId=«iPackageGroupId»' failed.<BR>
«ELSE»
  <BR>«_GName» Group rights for package <STRONG>«azPackageName»</STRONG>
  have been successfully updated.
«ENDIF»
```

---

### 5.1.3 PackageMain.phs

When, from the main page, the package name is clicked, a content as generated by the PackageMain.phs script is featured. It is used to manage the main data as used in the package. Therefore, it is mostly free-form apart from the loading of group rights which can be used to allow certain actions, such as adding a new element, deleting another...

Codes 9 and 10 feature an example of what can be done. There :

- the content of one the main tables (as defined in the PackageInstall.phs script, 5.1.1) is shown on the screen ;
- for each line, a  icon "launches" an EditFestival.phs script aimed at for instance modifying the data related to that festival ;
- a  icon "launches" yet another script (AddAnArtist.phs) aimed at for instance adding a new artist to the festival ;
- at last, at the bottom of the page, a  icon could be added, doing the same for a script aimed at adding a new element to the table.

It is to note that, while the bulk content of that package's main page is totally free, there are several things to comply to :

- INPUTs of the form as featured in Code 9 in black must be present ;

- a form always redirects to OPS's opsPackage.phs script. The script that has to be executed is stored in the CGI variable \_Script, always in the YourPackageName:ScriptToExecute.phs fashion.

### 5.1.3.1 Result : package main page

By accessing a package from the main page, the output of script PackageMain.phs is shown. The current example is featured at Fig.5.2.

**Festivals :**

<b>Festival List</b>				
<b>Name</b>	<b>Date</b>	<b>City</b>	<b>Artists</b>	
Lapin Kulta Heavy Metal Fest	22/6/2011	Rovaniemi	5	 
Mittelalter und Goth Fest	13/8/2011	Munich	4	 
Rock and Stuff	3/10/2011	Budapest	6	 
				

Figure 5.2: The main page for package Festivals.

---

**Code 9** An example for PackageMain.phs (I).

---

```
«IF azPackageRights[0] < 49»<B>You haven't got the right to use this
package</B>«ELSE»
```

```
<B>Festival List</B><P>
```

```
<FORM NAME="category" ACTION="/Scripts/sql.exe" METHOD="post">
  <INPUT TYPE="hidden" NAME="SqlDB" VALUE="«SqlDB»">
  <INPUT TYPE="hidden" NAME="xid" VALUE="«xid»">
  <INPUT TYPE="hidden" NAME="Sql" VALUE="opsPackage.phs">
  <INPUT TYPE="hidden" NAME="_Script" VALUE="">
  <INPUT TYPE="hidden" NAME="_Context" VALUE="«_Context»">
  <INPUT TYPE="hidden" NAME="iPackageId" VALUE="«iPackageId»">
  <INPUT TYPE="hidden" NAME="_Back" VALUE="">
  <INPUT TYPE="hidden" NAME="iPContext" VALUE="1">

  <INPUT TYPE="hidden" NAME="_FestivalId" VALUE="0">

  <TABLE CELLPADDING="2" CELLSPACING="0" BORDER="1">
    <TR>
      <TH BGCOLOR="#CCCCCC" nowrap>Name</TH>
      <TH BGCOLOR="#CCCCCC" nowrap>Date</TH>
      <TH BGCOLOR="#CCCCCC" nowrap>City</TH>
      <TH BGCOLOR="#CCCCCC" nowrap>Artists</TH>
      <TH BGCOLOR="#CCCCCC" nowrap>&nbsp;</TH>
    </TR>
    «SQLOUTPUT»
    <TR>
      <TD>«azName»</TD>
      <TD>«=@daytimeday(iDate)»/«=@daytimemonth(iMonth)»/«=@daytimeyear(iDate)»</TD>
      <TD>«azCity»</TD>
      «SQLEXEC INT nbArtists = SELECT COUNT(*) FROM OPS_FESTIVALS WHERE
      idFestival=«idFestival»»
      <TD align="center">«nbArtists»</TD>
      <TD>
        «IF azPackageRights[0]>49»
        <IMG SRC="«#SQLWVHOME#»/img/data.gif"
        STYLE="cursor:pointer" onClick="onEditFestival(«idFestival»)"/>
        &nbsp;<IMG SRC="«#SQLWVHOME#»/img/data_ext.gif" STYLE="cursor:pointer"
        onClick="onAddArtist(«idFestival»)"/>
        «ENDIF»
      </TD>
    </TR>
    «/SQLOUTPUT SELECT DISTINCT(azName), idFestival, iDate, azCity FROM
    OPS_FESTIVALS ORDER BY iDate»
  </TABLE>
```

```
(...)
```

```
</FORM>
```

```
«ENDIF»
```

---

---

**Code 10** An example for PackageMain.phs (II : javascript part).

---

```
<SCRIPT LANGUAGE="javascript">

function onEditFestival(i)
{
  document.category._FestivalId.value=i;
  document.category._Script.value="Festivals:EditFestival.phs";
  document.category.submit();
}

function onAddArtist(i)
{
  document.category._FestivalId.value=i;
  document.category._Script.value="Festivals:AddAnArtist.phs";
  document.category.submit();
}

(...)

</SCRIPT>
```

---

#### 5.1.4 PackageUninstall.phs

This script is launched when an uninstall of a package (5.2.3) is done. It can be used, as featured in Code 11, to erase all table data used by the package.

---

**Code 11** An example for PackageUninstall.phs.

---

```
«SQL DROP TABLE OSL_FESTIVALS»
...
(Drop all package's tables)

«DIRECTORY DELETE images/artists»
...
(Delete all created folders)
```

---

## 5.2 Package general management

### 5.2.1 Installing a package

Once at least all mandatory files as described in 5.1 have been written and compiled with the `sqllib` compiler and the compiled library has been uploaded to the Publisher's SysLibHome, the package is ready to be installed, as featured on Fig.5.3.

**Package / Install :**

**Package Install**

Select the package to install:

Package Library:  Please enter the root of the package library.

Package Name:

Administrator Group:

**Package Install**

The package version 1 has been installed.

Figure 5.3: The first install of the package "Festivals".

### 5.2.2 Updating a package

Through time, a package's contents may be updated : new scripts can be added, others modified, some SQL updates have to be performed... In any case, the library will have to be compiled and uploaded again.

If the case of SQL updates and/or actions that necessitate OSL code to be executed once (for instance, `DIRECTORY CREATE`), they have to be featured in the `PackageInstall.phs` script, updated as shown at Code 12.

In the case of an update of `PackageInstall.phs`, the package will have to be re-installed as described above ; on success, the screen will confirm that package has been upgraded :

**Package Install**

The package has been upgraded to version 2.

---

**Code 12** An append to `PackageInstall.phs` to upgrade the package to Version 2.

---

*[Instructions for Version 1]*

```
«IF «piPackageVersion»<3»
  «* Upgrade package to version 2*»
  «SQL UPDATE (...)»
  «LET _Ok=#SQLSTATUS#»
«ENDIF»

«IF _Ok==1»
  «LET _PackageVersion=2»
«ENDIF»
```

---



*After tables are created, queries on them can be executed in a simpler, quicker way through [Site management](#) / [Execute SQL](#) (4.1.4).*

The current versions of all installed packages can easily be checked thanks in [Options](#) (4.6.1).

### 5.2.3 Uninstalling a package

Through [Packages](#) / [Uninstall](#), a list of all packages is displayed ; clicking on a package will execute the `PackageUninstall.phs` script that had been defined for this package (5.1.4) and will remove it from the main screen. According to this script's contents, data may be erased irreversibly.

## 5.3 Using object types and objects to compose headlines

Packages, with their possibility to execute `.phs` scripts, offer vast advantages for headline creation (2.4). More precisely, one single script can be used to produce several different headlines, according to variables that are called object types and objects.

In Blue Chameleon Content Management System, these are managed in the way shown at Fig.5.4.

There :

- as leaf frame (2.4.2) is edited within headline modify context, a script `PackageTypeObjList.phs` displays a choice of several object types, one of which is



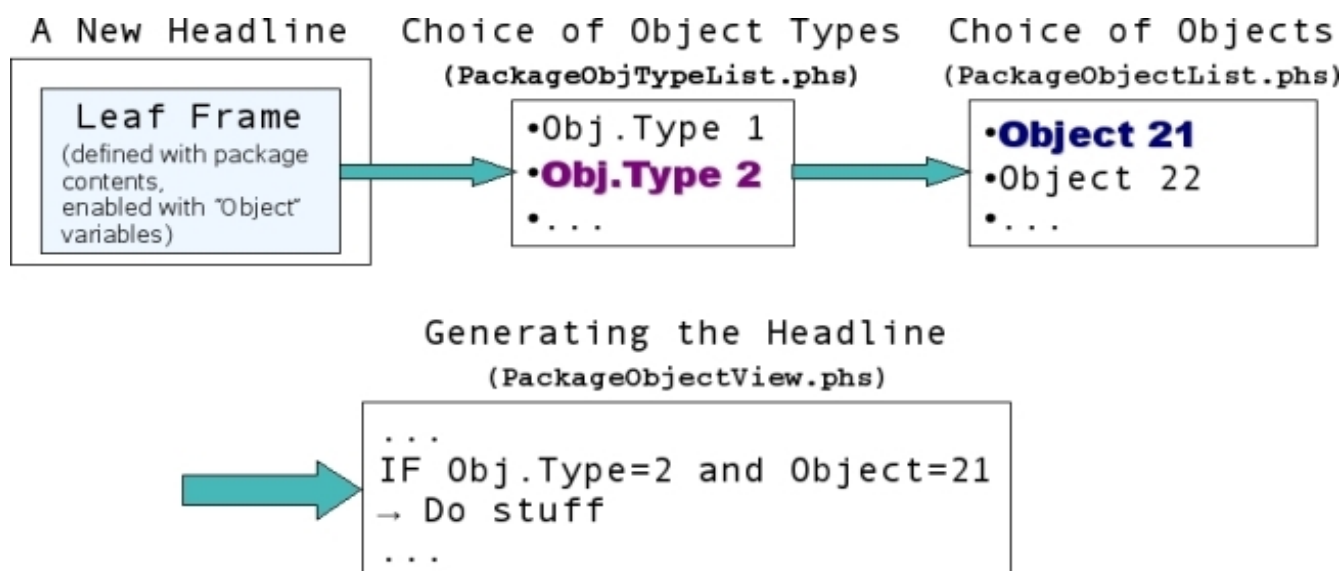


Figure 5.4: The process of selecting an object type and an object for a leaf frame that is package-enabled.

chosen ;

- then, according to the chosen value, a choice of objects is shown through PackageObjectList.phs ;
- upon selection of object, the couple of values (Object Type, Object) steers to the right place of PackageObjectView.phs in order to generate, upon validation, an information-specific headline that reflects the choices made.

In the current example, in order to make information pages for festivals and artists, a practical application would be :

- make object types as "Festivals" and "Artists" ;
- for each, make objects as individual festivals and individual artists.

The following shows how this practical example is implemented.

### 5.3.1 Enabling a leaf frame to handle packages and an object

A leaf frame "Info Page" is created (2.3.1) to handle packages. Code 13 then shows what has to be inputted for the frame's content (during creation or modification).

It is to note that Code 13 is the same whatever the leaf frame or the package are. Now, for the leaf frame's variables, as shown in Code 14, the contents depend on what should be handled in the frame's layout.

In this code, "PO" stands for *package object* and "14" is the identifier for that type; in between, a title for the object (which can be here either a festival or an artist) is given.

---

**Code 13** A leaf frame's contents to handle packages, for a single object :

---

[Headline models](#) / [Modify](#) / [Info Page](#) :

```
«INCLUDE PackageObjectView.phs;_PackageId=«POPackageId»;
_ObjectTypeId=«POObjectType»;_ObjectId=«POObjectId»;
_ObjectAttrType=«POStyleAttrType»;_ObjectAttrValue=«POStyleAttrValue»»
```

---

---

**Code 14** A package-oriented frame's variable for handling an object :

---

[Headline models](#) / [Variables](#) / [Info Page](#) :

```
PO;Festival/Artist;14;
```

---

### 5.3.2 PackageObjTypeList.phs

Code 15 shows how the code that proposes the choice between object types can be written.

---

**Code 15** An example for PackageObjTypeList.phs :

---

```
<TABLE cellpadding="2" cellspacing="0" border="0">
<TR>
  <TD><A HREF="javascript:selectObjectType(150,1)">Festivals</A></TD>
</TR>
<TR>
  <TD><A HREF="javascript:selectObjectType(150,2)">Artists</A></TD>
</TR>
</TABLE>
```

---

There, a choice between object types is laid out through a table ; each object type must use function `selectObjectType`, with for arguments the identifier of the package and a distinct number that will identify the object type.

Fig.5.5 shows the result of this script. On the *Headline Modify Page* for headline "Rock and Stuff" - which template accepts leaf frame Info Page - as frame is edited, the "Festival/Artist" variable previously defined for the frame is visible, along with a  button.

On click, the list of packages is shown and, upon selection of the "Festivals" one, choice is then offered between the two defined object types.

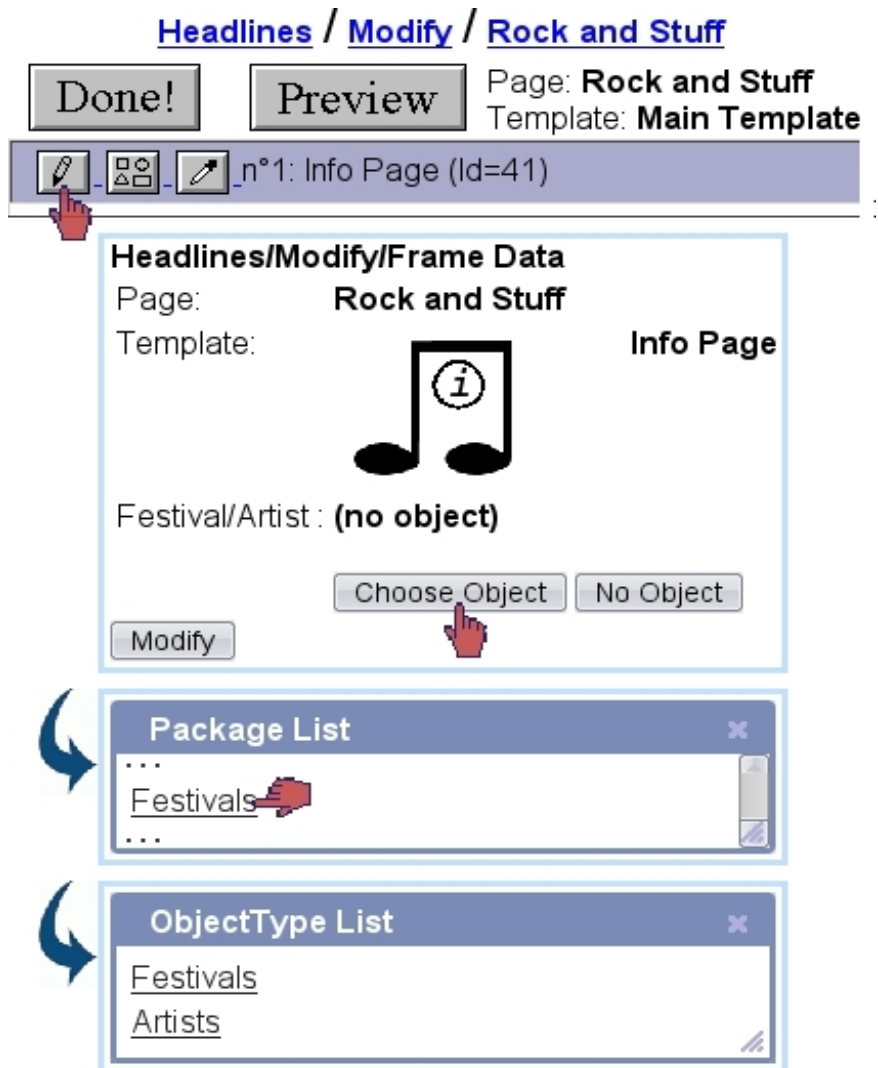


Figure 5.5: A selection of object types.

### 5.3.3 PackageObjectList.phs

This script must use the variable `iObjectType` as set by the previous script to select the right object list. As Code 16 shows, each object line has to redirect to the `selectObject` function, which arguments are package identifier, object type identifier, and a distinct number that will identify the object.

---

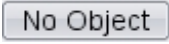
**Code 16** An example for `PackageObjectList.phs` :

---

```
<TABLE cellpadding="2" cellspacing="0" border="0">
  <IF @toi(iObjectType)==1> <*<Festivals*>
    <TR>
      <TD><A HREF="javascript:selectObject(150,1,0)">All festivals</A></TD>
    </TR>
    <SQLOUTPUT>
    <TR>
      <TD><A HREF="javascript:selectObject(150,1,<idFestival>)"><azName></A></TD>
    </TR>
  </SQLOUTPUT SELECT DISTINCT(azName), idFestival FROM OPS_FESTIVALS ORDER BY
  azName>
  <ELSEIF @toi(iObjectType)==2> <*<Artists*>
    <TR>
      <TD><A HREF="javascript:selectObject(150,2,0)">All artists</A></TD>
    </TR>
    <SQLOUTPUT>
    <TR>
      <TD><A HREF="javascript:selectObject(150,2,<idArtist>)"><azName></A></TD>
    </TR>
  </SQLOUTPUT azName FROM OPS_ARTISTS ORDER BY azName>
  <ENDIF>
</TABLE>
```

---

The result of this script is featured at Fig.5.6, where, after "Festivals" from the object type list is clicked, the list of festivals is proposed. Upon final selection, the page shows well that "Festival/Artist" has been set to "Rock and Stuff".

If the wrong object or object type was selected, it is possible to reinitialize the Object variable with the  button.

The object name as featured on the final screen is set by a small script, `PackageObjectName.phs`, described as follows.

### 5.3.4 PackageObjectName.phs

As shown in Code 17, following a similar structure as the previous one, this script makes use of the now-existing `iObjectId` value (referring to the selected object) to simply fetch

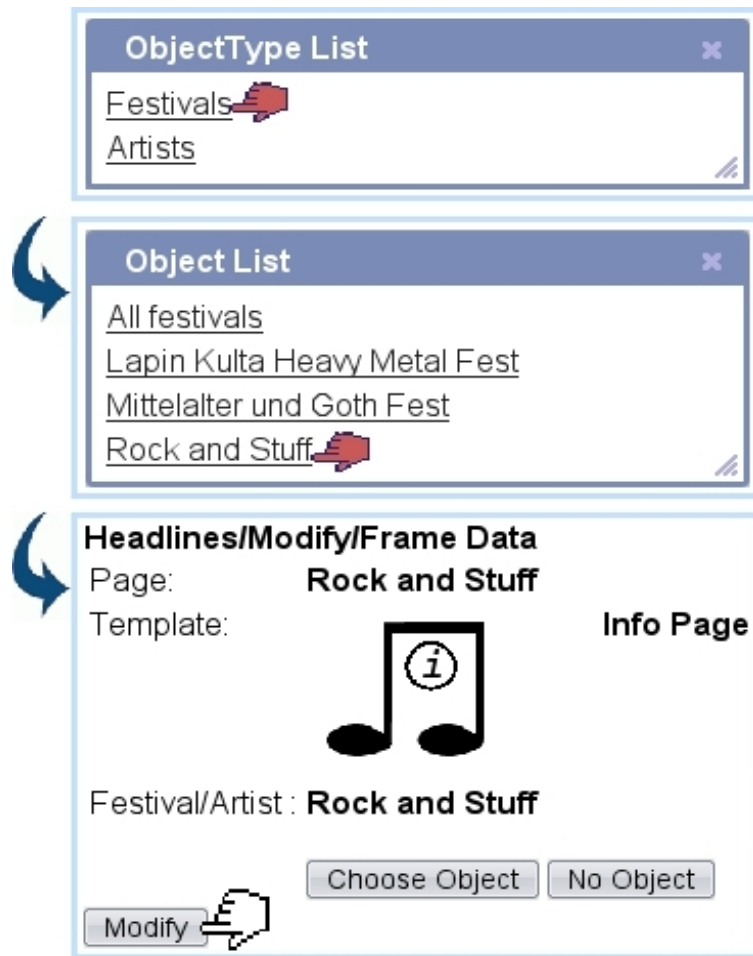


Figure 5.6: Finally choosing the object.

its name.

---

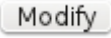
**Code 17** An example for PackageObjectName.phs :

---

```
<VAR NEW _Name="">
<IF @toi(iObjectType)==1> <*<Festivals*>
  <SQLEXEC STRING _Name=SELECT azName FROM OPS_FESTIVALS WHERE
    idFestival=<=@toi(iObjectId)>>
<ELSEIF @toi(iObjectType)==2> <*<Artists*>
  <SQLEXEC STRING _Name=SELECT azName FROM OPS_ARTISTS WHERE
    idArtist=<=@toi(iObjectId)>>
<ENDIF>
<<B>_Name</B>>
```

---

### 5.3.5 PackageObjectView.phs

This script, as executed when the  button is clicked after object was set, will finally generate the headline.

Variables available there are `_ObjectId` and `_ObjectType`, respectively identifying the selected object type and object.

Code 18 then shows a template of how `PackageObjectView.phs` can be composed : a first test on `_ObjectType` steers to the right part of the script, where an other test on `_ObjectId` decides which festival data has to be shown.

It is to note that this script, thanks to the object concept, works with any festival or artist for which an information page has to be done : **a single script to make them all.**

---

**Code 18** An example for PackageObjectView.phs :

---

```
(...)  
  
«IF @toi(_ObjectId)==1» «*Festivals*»  
  
  «VAR NEW _Where=""»  
  «IF @toi(_ObjectId)<>0»  
    «LET _Where=" WHERE idFestival=@toi(_ObjectId) "»  
  «ENDIF»  
  
  «SQLOUTPUT»  
  
    (Display formatted festival information)  
  
  «/SQLOUTPUT SELECT * FROM OPS_FESTIVALS «_Where» ORDER BY azName»  
  
«ELSEIF @toi(_ObjectId)==2» «*Artists*»  
  
(...)  
  
«ENDIF»
```

---

## 5.4 Using attributes

The above described how to produce specific pages with the help of an "Object" variable. Need may arise though to particularize a page even more ; Blue Chameleon Content Management allows it, with the help of an "Attribute".

Basically, an attribute functions works as an object ; it is defined in the first place as a leaf frame's variable, then dedicated scripts (following the same pattern as the Object ones, with different variable names) are written to handle attribute types and subsequent attributes.

Finally, when a headline's package-enabled leaf frame is edited, the attribute variable will be featured below the object as new configuring option.

### 5.4.1 Enabling a frame to handle attributes

An attribute variable is added in the frame's variables ; it is named "POStyle" and with the identifier "15". In this example, it will bear the name "Style/Mode" :

[Headline models](#) / [Variables](#) / [Info Page](#) :

Table 5.1: Scripts, functions and variables as used for attribute setting

<b>PackageAttTypeList.phs</b> Shows a list of attribute types : ... <TR><TD><A HREF="javascript:AttributeType(1)">Style</A></TD></TR> <TR><TD><A HREF="javascript:AttributeType(2)">Mode</A></TD></TR> ...
<b>PackageAttVallist.phs</b> Shows the list of attributes, corresponding to the selected type : ... «IF @toi(iAttrType)==2» <TR><TD><A HREF="javascript:AttributeValue(1)">View only</A></TD></TR> <TR><TD><A HREF="javascript:AttributeValue(2)">Ticket book</A></TD></TR> «ENDIF»
<b>PackageAttrName.phs</b> Outputs the name of the selected attribute : ... «ELSEIF @toi(iObjectType)==2» «*Mode*» «IF @toi(iAttrValue)==1» <B>View only</B> «ELSEIF @toi(iAttrValue)==2» «*Mode*» <B>Ticket book</B> «ENDIF» «ENDIF»

PO;Festival/Artist;14

POStyle;Style/Mode;15

## 5.4.2 Scripts for attribute

Table 5.1 shows how what scripts are attribute's own, along with code snippets.

When the `PackageAttTypeList.phs`, `PackageAttVallist.phs` and `PackageAttrName.phs` scripts have been written, the `_ObjectType` and `_AttributeValue` variables can then be used within the `PackageObjectView.phs` script along with the object variables. Here is an example on how they can be integrated :

```
(...)  
«IF @toi(_ObjectId)==1» «*Festivals*»
```



```

«VAR NEW _Where=""»
«IF @toi(_ObjectId)<>0»
  «LET _Where=" WHERE idFestival=@toi(_ObjectId) "»
«ENDIF»

«SQLOUTPUT»
  (Display formatted festival information)
  «IF @toi(_ObjectAttrType)==2 && @toi(_ObjectAttrValue)==2 »«*Mode : Book
ticket*»
  «IMG SRC="«#SQLWWWHOME#»/img/BookNow.gif" STYLE="cursor:pointer"
onClick="onBookNow(«idFestival»)"/>
  «ENDIF»
«/SQLOUTPUT SELECT * FROM OPS_FESTIVALS «_Where» ORDER BY azName»

«ELSEIF @toi(_ObjectTypeId)==2» «*Artists*»
(...)

```

## 5.5 Featuring multiple objects on the same headline

The previous sections aimed at describing how a headline containing a single object could be composed ; now, the flexibility of Blue Chameleon Content Management System's object and attribute system allows, as a matter of fact, more than one object to be featured on the *same* headline.

Indeed, a need for such might arise for instance when multiple elements (as gathered from the database) should be displayed on the screen, but along *different* styles ; and administrator should be able to manage them easily.

Fig.5.7 shows an instance of that need, in the current example we followed. A headline called "Featured Artist of the Day" aims to feature in an extensive manner a particular artist (i.e. displaying all possible available information on her/him), while *also* featuring, at the bottom the page (and prefaced by such phrase as 'Other artists of interest...') two other artists, but in a far more minimalistic fashion.

Each artist to display will be treated as an object, but each will have a *different attribute* : for instance the attribute "HUGE showcase" for the main one, and "Small showcase left", "Small showcase right" for the two others. Thus, to implement it : a leaf frame "3-artist frame" will be defined as follows, handling 3 objects and 3 attributes.

# Headline “Featured Artist of the Day”

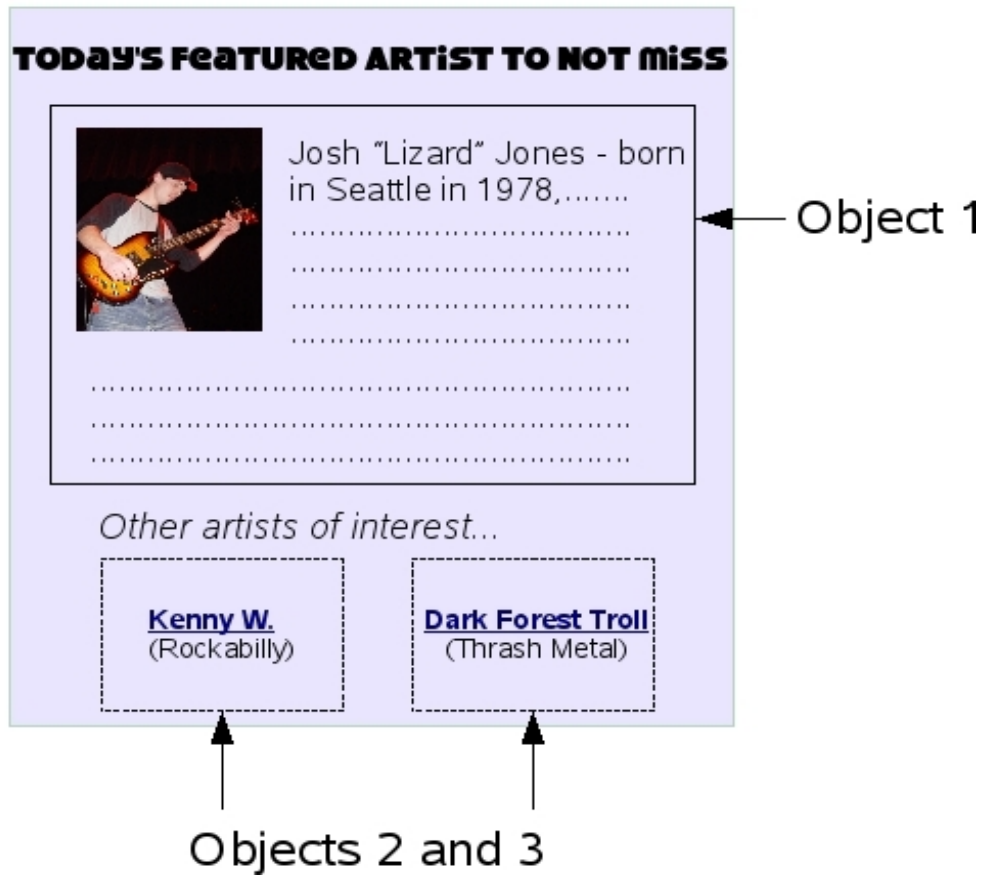


Figure 5.7: A page design that calls for multiple objects to be featured.

## 5.5.1 A leaf frame that handles multiple objects

In Code 13, it can be seen that a single call to system function `PackageObjectView.phs` was made, with a single family of arguments (`P0ObjectType, P0ObjectId,...`).

Now, in order to display the multiple objects that we need, it is no simpler than to call the same function, but as many times as we need it (3 times here), and each time with a new set of variables : Code 19 then shows the result. It can be seen that variables are named `P01{...}`, `P0Style1{...}`, `P02{...}`, `P0Style2{...}`, etc. for each to be unique.

The contents must feature the page's HTML formatting in advance ; for instance, here, the `TABLE` formatting to display the 'lesser' artists one besides the other.

As for the frame's variables, they will thus be defined as shown in the same Code, at the bottom.

---

**Code 19** A leaf frame's contents and variables to handle packages, for 3 objects :

---

[Headline models](#) / [Modify](#) / [3-artist frame](#) :

```
<!-- Main Artist -->
«INCLUDE PackageObjectView.phs;_PackageId=«P01PackageId»;
_ObjectTypeId=«P01ObjectType»;_ObjectId=«P01ObjectId»;
_ObjectAttrType=«POStyle1AttrType»;_ObjectAttrValue=«POStyle1AttrValue»»

<TABLE>
<TR>
<TD> <!-- Artist on the bottom, left -->
  «INCLUDE PackageObjectView.phs;_PackageId=«P02PackageId»;
  _ObjectId=«P02ObjectId»;_ObjectAttrType=«POStyle2AttrType»;_ObjectAttrValue=«POStyle2AttrValue»»
</TD>
<TD> <!-- Artist on the bottom, right -->
  «INCLUDE PackageObjectView.phs;_PackageId=«P03PackageId»;
  _ObjectId=«P03ObjectId»;_ObjectAttrType=«POStyle3AttrType»;_ObjectAttrValue=«POStyle3AttrValue»»
</TD>
<TR>
</TABLE>
```

[Headline models](#) / [Variables](#) / [3-artist frame](#) :

```
P01;Main Artist;14
POStyle1;Display style 1;15

P02;Artist, Left;14
POStyle2;Display style 2;15

P03;Artist, Right;14
POStyle3;Display style 3;15
```

---



*From the unicity of the Object and Attribute scripts, a new package could be defined so as to keep various functionalities in the same publisher. Indeed, as a `_PackageId` variable is used for each `PackageObjectView.phs` call (Code 19), **a single frame can use different packages.***

*As a matter of fact, as Fig.5.5 well showed, the selection of an object always starts by a list of available packages. .*

## 5.5.2 Packages : defining more of them

In the current example, a "Festivals" package had been previously defined, with its Object and Attribute scripts. Now, for the needs of the page "Featured Artist of the Day" we aim to build, we need to define new scripts. In order to not delete the previous ones, **it is possible to define a new package (for instance called 'Artist News')** where we will define our new scripts.

In the new `PackageInstall.phs` script, no SQL table would have to be created if our new needs are covered by the previously-created tables (`OPS_ARTISTS`, for instance) : **any table defined in a package context is accessible by any package.**

Nonetheless, a new package is a great occasion to expand available information : for instance, this new package could entail the creation of a `OPS_ARTISTS_MORE`, aimed at containing far more detailed information about artists (biographies, member(s), instruments played,...).

## 5.5.3 Object and attribute scripts for this case (guidelines)

The Object scripts (not `PackageView.phs`), in the current example, would be aimed at artist selection only. It could be then quite practical to define object types as musical styles and thus offer an easier, quicker way to select which artists to be featured on the headline :

PackageObjTypeList.phs :

```
(...)  
<TD><A HREF="javascript:selectObjectType(200,«idMusicalStyle»)">  
«azMusicalStyle»</A></TD>  
(...)
```

PackageObjectList.phs :

```
(...)  
<TD><A HREF="javascript:selectObject(200,«idMusicalStyle»,«idArtist»)">  
«azArtist»</A></TD>  
(...)
```

For attributes, there is no need to define several types of them : one default type is enough, holding three attributes, one for each display style :

PackageAttTypeList.phs :

```
<TD><A HREF="javascript:javascript:AttributeType(1)">Default</A></TD>
```

PackageAttValList.phs :

```
«IF @toi(iAttrType)==1»  
<TABLE>  
  <TR><TD><A HREF="javascript:AttributeValue(1)"></A>HUGE showcase</TD></TR>  
  <TR><TD><A HREF="javascript:AttributeValue(2)"></A>Small showcase left</TD></TR>  
  <TR><TD><A HREF="javascript:AttributeValue(3)"></A>Small showcase right</TD></TR>  
</TABLE>  
«ENDIF»
```

As for the scripts used for displaying names, `PackageObjectName.phs` and `PackageAttrName.phs` are built in accordance with `PackageObjectList.phs` and `PackageAttValList.phs`.

The result of those, on the *Headline Modify Page* is featured at Fig.5.8.

[Headlines](#) / [Modify](#) / [Featured Artist of the Day](#)

Page: **Featured Artist of the Day**  
 Template: **Main Template**

n°1: 3-artist frame (Id=43)

**Headlines/Modify/Frame Data**

Page: **Featured Artist of the Day**

Template: **3-artist frame**

ARTIST

(artist)
(artist)

Main Artist : **Josh ``Lizard`` Jones**

Display style 1 : **HUGE showcase**

Artist, Left : **Kenny W.**

Display style 2 : **Small showcase left**

Artist, Right : **Dark Forest Troll**

Display style 3 : **Small showcase right**

Figure 5.8: The 3-object frame after all relevant choices of objects and attributes have been made.

### 5.5.3.1 The PackageObjectView.phs script for this example

Finally, the `PackageObjectView.phs` script for which a sketch is shown at Code 20, will generate the headline ; it will be executed three times (i.e., as many times as there are `INCLUDE`'s in the frame's code) by the system upon click of the  button. For each time, the system will set the right variables for the current object type/object and the attribute type/attribute value.

---

**Code 20** An example for PackageObjectView.phs for the 'Artist News' package :

---

```
(...)  
  
«IF @toi(_ObjectAttrValue)==1» «*Main Artist Display*»  
  
  Today's featured Artist to NOT miss  
  «*Fetch artist information using _ObjectId *»  
  (...)  
  
«ELSEIF @toi(_ObjectAttrValue)==2 || @toi(_ObjectAttrValue)==3» «*Display for  
the two other artists*»  
  
  «*Fetch artist information using _ObjectId *»  
  (...)  
  <A HREF="#" onClick="onViewArtistDetails(«=@toi(_ObjectId)»)»>«_Name»</A><BR>  
  («_MusicalStyle»)  
  
«ENDIF»
```

---

More precisely, at the first turn, (mentioning only the variables of interest) the object value will correspond to Josh "Lizard" Jones' identifier and an attribute value `_ObjectAttrValue` of 1, thus producing the 'Today's featured Artist to NOT miss' part. On the next two turns, identifiers will correspond to the two other artists, and the attribute values of 2 and 3 will produce the smaller display.

It is to note that the table disposition of those two last objects was already done when frame was defined (Code 19), so there is no need to do it here.





# Chapter 6

## Annex

### 6.1 Leaf frame variables

Variables of a leaf frame must be given in the following manner,

*Name;Label;Type; [Data]*

where :

- *Name* is the name of the variable, that will be used in the leaf frame model's contents;
- *Label* is the label under which, in the context of leaf frame editing, the variable will be prompted ;
- *Type* is an identifier describing the type of variable (see Tables 6.1, 6.2 below) ;
- *Data* is a string of parameters : it is only given for 'select'- or 'SQL select'- type variables (6.1.2, 6.1.3) and empty for all of the others.

Table 6.1: Leaf frame variables

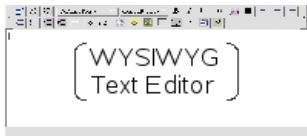
Type id.	Type	Example	Leaf frame edit
1	Text edit	<code>_Text;City;1;</code>	City : <input type="text"/>
2	Text area	<code>_Comment;Comments;2;</code>	Comments : 
3	Image	<code>_Pic;Photo;3;</code>	Photo : <No image selected> <input type="button" value="Choose image"/> <input type="button" value="No image"/> <input type="button" value="Edit image map links"/>
4	Link	<code>_Link;Link;4;</code>	Link : <No link selected> <input type="button" value="Choose link"/> <input type="button" value="No link"/>
5	Script (view)	<code>_Skr;Script;5;</code>	Script : <input type="text" value="-----"/> <input type="button" value="v"/> paramètres: <input type="text"/>
6	Shared varfile	<code>_Varf;File;6;</code>	File : not selected <input type="button" value="Change varfile"/>
7	Table	see 6.1.1	Table : <input checked="" type="radio"/> internal format (;) <input type="radio"/> Tabs <input type="radio"/> Spaces <input type="text"/>
8	Script (edit)	Deprecated	Deprecated
9	Select	see 6.1.2	Country : <input type="button" value="Canada"/> <input type="button" value="v"/>
10	Download	<code>_Dwnl;Download;10;</code>	Download : No download has been selected. <input type="button" value="Change download"/> <input type="button" value="No download"/>

Table 6.2: Leaf frame variables (continued)

11	Anchor	<code>_Ank;Anchor;11;</code>	Anchor : <input type="text"/>
12	Menu	<code>_Menu;Menu;12;</code>	Menu : No menu selected <input type="button" value="Change menu"/> <input type="button" value="No menu"/>
13	SQL select	see 6.1.3	Festival : <input type="text" value="Mittelalter und Goth Fest"/> ▾
14	Package object	<code>P01;Main Artist;14;</code>	Main Artist : <b>Josh "Lizard" Jones</b> <input type="button" value="Choose Object"/> <input type="button" value="No Object"/>
15	Attribute	<code>POStyle2;Display style 2;15;</code>	Display style 2 : <b>Small showcase left</b> <input type="button" value="Choose Attribute"/> <input type="button" value="No Attribute"/>



*These variables are then practically integrated in leaf frames thanks to system scripts as detailed at Tables 6.4 and 6.5.*

### 6.1.1 "Table" variables

### 6.1.2 "Select" variables

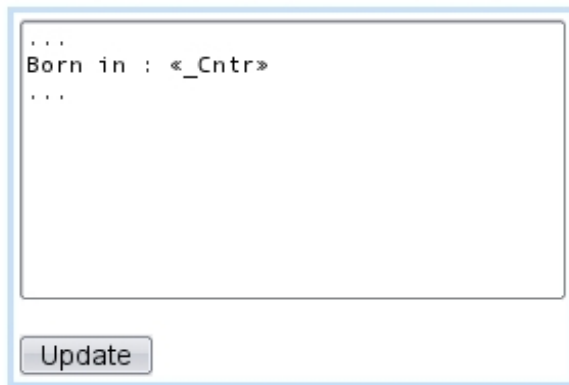
They ultimately serve as to provide, as featured on Table 6.1, a drop-down menu filled with elements.

The variable is set for instance as follows :

```
_Cntr;Country;9;USA/Canada/UK/...
```

Here, the [Data] string lists values, separated by slashes, that will be featured in the drop-down menu while editing the leaf frame. Upon previewing (and after, publication) of the headline, the variable will be set to the very value that was chosen, for instance :

## Headline models / Modify / a leaf frame mdl :



```
...  
Born in : <_Cntr>  
...
```

Update

[Frame edit] :



Country : Canada

USA  
Canada  
UK  
...

Preview

Born in : Canada

The [Data] string can be defined in a more elaborate manner with value=label couples, also separated by slashes, for instance as follows :

```
_Rezur;Reservation;9;-1=N.A./0=Non necessary/1=Mandatory
```

During editing of the frame, the drop-down menu will display the labels "N.A.", "Non necessary"... and, upon validation of the headline, the variable <\_Rezur> will be assessed the corresponding value (-1,0,...).

### 6.1.3 "SQL select" variables

Those variables work in a similar manner as the previous ones, i.e. a drop-down menu is provided, with elements corresponding to values. Except here, the couples value=label are generated by a SQL query, a SELECT, done so as **two columns** of a table are the result of the query :

```
_Price;Festival;13;SELECT DISTINCT(azName),mPrice FROM OPS_FESTIVALS  
a,OPS_TICKET_PRICE b WHERE a.idFestival=b.idFestival
```

This query will generate two columns with the festival name and its corresponding price, and, upon validation of the headline, the variable <\_Price> will amount to the chosen festival.

## 6.2 System script glossary

### 6.2.1 Headline model scripts

During the composing of a headline model's contents (and also for list and container models), some system scripts are called. Table 6.3 sums them all.



*For 'Text edit', 'Text area', 'Select' and 'SQL select' variables, they are simply inserted via <\_Variable>, e.g. <\_Text>, <\_Comment>, <\_Cntr>, <\_Price>.*

Table 6.3: System scripts for headline, list and container models

Name	Call & Function
design.phs	<pre>«INCLUDE design.phs»</pre> <p>Enables the <b>Preview</b> and <b>Done!</b> buttons to appear when a headline created from this model is edited. It is therefore a <b>mandatory</b> script.</p>
frame.phs	<pre>«INCLUDE frame.phs;_ParentId=«_FSId»;_FrameIndex={1,2,...}»</pre> <p>For models that can have children (headlines, containers, lists), loads the frame number 1, 2,... This script must be called as many times as model has children frames, and those calls must be disposed, HTML-wise, in a manner that mirrors the desired layout of the page.</p>
metatags.phs	<pre>«INCLUDE metatags.phs»</pre> <p>Placed in the &lt;HEAD&gt;...&lt;/HEAD&gt; section of the headline model's contents, inserts into the same section of the rendered .htm page the meta-tags that were filled on headline's <a href="#">Manage</a> page.</p>
ListItems.phs	<pre>«INCLUDE ListItems.phs;_ListFrameItems="_ItemCount"»</pre> <p>In the case of a List model, puts into the variable <code>_ItemCount</code> (that has to be initialized before call) the number of list items that the list currently holds. It is then used to call <code>frame.phs</code> this very number of times.</p>
form.phs	<pre>«INCLUDE form.phs»</pre> <p>Will enable the headlines generated from this model to use forms.</p>

## 6.2.2 Leaf frame model scripts

Leaf frame models have also their system scripts, aimed at loading their variables, as detailed in Tables 6.4 and 6.5.

Table 6.4: System scripts for leaves

image.phs	<pre>«INCLUDE image.phs;pImage="_Pict";Html="border='0'..."»</pre> <p>Loads the image as referred to by the variable <code>_Pict</code>; the <code>Html="..."</code> string can be used to give any HTML additional parameter. As a result, the following tag will be generated :</p> <pre>&lt;IMG WIDTH=.. HEIGHT=.. ALT="..." SRC="..." <i>additional parameters</i>&gt;</pre> <pre>«INCLUDE image.phs;pImage="_Pict";_URLOnly=1;_Return="_Url"»»</pre> <p>This calls will store in variable <code>_Url</code> the address of the image.</p>
href.phs	<pre>«INCLUDE href.phs;_Link="_Link";Parameters»</pre> <p>Inserts a link as referred to by variable <code>_Link</code>, in different manners as specified by <i>Parameters</i> :</p> <ul style="list-style-type: none"> <li>● <b>StartTag</b> : generates the tag <code>&lt;A HREF="..."&gt;</code> if any link has been selected by the user ;</li> <li>● <b>EndTag</b> : generates the closing tag <code>&lt;/A&gt;</code> if necessary ;</li> <li>● <b>NoText</b> : the generated <code>&lt;A HREF="..."&gt;</code> tag is trimmed so that it does not contain any redundant blanks or lines ;</li> <li>● <b>Target</b> : will add the specified target to the <code>&lt;A HREF="..." TARGET="Target"&gt;</code> tag ;</li> <li>● <b>JustUrl</b> : (as a unique parameter) returns only the address of the link ;</li> <li>● <b>_Return=_ReturnVAR</b> : makes <code>href.phs</code> not give out HTML tags, but specific information as stored in a <code>_ReturnVAR</code> variable that must be initialized before call. The various <code>_ReturnVARs</code> specify the information : <ul style="list-style-type: none"> <li>◇ <code>_ReturnAddress</code> contains the URL of the link</li> <li>◇ <code>_ReturnType</code> contains the type of the link. Possible values are article, headline, URL, download, anchor and book</li> <li>◇ <code>_ReturnStyle</code> contains the name of the file containing the link style</li> <li>◇ <code>_ReturnStyleId</code> contains the Id of the link style.</li> </ul> </li> <li>● <b>AddDomain</b> : forces the coding of the domain specified for the publisher in the URL ;</li> <li>● <b>_OPSLink</b> : forces the link to be made to the OPS name of the article/headline/page object even if the user has indicated an publication name.</li> </ul>
scriptfile.phs	<pre>«INCLUDE scriptfile.phs»</pre> <p>Enables the leaf frame to load scripts. If the published page is meant to be used dynamically, the commands <code>«VAR NEW _Dynamic»</code> <code>«_Dynamic="yes"»</code> must be placed before the call.</p>
varphile.phs	<pre>«INCLUDE varfile.phs;_Action={1,2};_File="_MyFile"»</pre> <p>Loads (<code>_Action=1</code>) or unloads (<code>_Action=2</code>) the variables defined in the varfile as referred to by <code>_MyFile</code>. Between these two commands, the variables defined by the selected varfile can be used as if they were variables defined for the current leaf.</p>

Table 6.5: System scripts for leaves (continued)

table.phs	«INCLUDE table.phs;_Table="_Tbl";_Align="Align"» Inserts a table as referred to by variable <code>_Tbl</code> with the <code>Align</code> parameter being any of the alignments used for TD tags (left, right...)
download.phs	«INCLUDE download.phs;_Download="_Dwnl"» Generates the <code>&lt;A HREF="..."&gt;</code> , <code>&lt;/A&gt;</code> tags around the download description as relevant to the <code>_Dwnl</code> variable.
anchor.phs	«INCLUDE anchor.phs;_Anchor="_Ank"» Generates the <code>&lt;A NAME="..."&gt;</code> tag for the anchor as referred to by variable <code>_Ank</code> .
showmenu.phs	«INCLUDE showmenu.phs;_Menu=_Menu» Loads the menu as referred to by the variable <code>_Menu</code> .
PackageObjectView.phs	see 5.3.5, 5.4

## 6.3 Icon glossary

Tables 6.6 and 6.7 sum up what is the role icons as used during the editing of pages and in other contexts.

Table 6.6: Blue Chameleon Content Management icons


















Icon	[Context] and Action
 (Layout)	<p><b>[Headline edit]</b></p> <ul style="list-style-type: none"> <li>•If a headline, list, container can use several types of frames (2.2.3.1.1), this icon leads to a page where the frame to use there is selected.</li> </ul>
 (Edit)	<p><b>[Headline edit]</b></p> <ul style="list-style-type: none"> <li>•Edits a leaf frame, i.e. leads to a screen where its variables (text areas,...) are filled.</li> </ul> <p><b>[Menu item (3.5.3.1)]</b></p> <ul style="list-style-type: none"> <li>•Edits the menu item's text and link.</li> </ul> <p><b>[Form model (3.6.1.1)]</b></p> <ul style="list-style-type: none"> <li>•Edits that parameter of a form model.</li> </ul> <p><b>[Modify Form Page (3.6.3)]</b></p> <ul style="list-style-type: none"> <li>•Edits a specific page of a form.</li> </ul> <p><b>[A page of a form (3.6.3.3)]</b></p> <ul style="list-style-type: none"> <li>•Edits a form element.</li> </ul>
 (Extd. edit)	<p><b>[Headline edit]</b></p> <ul style="list-style-type: none"> <li>•Changes the parameters of the scripts, as well as the scripts that are selected for frames that handle them (in extended mode only, appearance set in 2.3.4)</li> </ul> <p><b>[Menu item (3.5.3.1)]</b></p> <ul style="list-style-type: none"> <li>•Displays the output of the script related to this menu item.</li> </ul>
 (Style)	<p><b>[Headline edit]</b></p> <ul style="list-style-type: none"> <li>•Accesses the frame style settings (2.3.4)</li> </ul> <p><b>[A page of a form (3.6.3.3)]</b></p> <ul style="list-style-type: none"> <li>•Leads to a screen where the style of the form element (checkbox, listbox,...) is selected (3.6.3.2).</li> </ul>
 (Hide/Display)	<p><b>[Headline edit]</b></p> <ul style="list-style-type: none"> <li>•Hides/shows a frame (appearance set in 2.3.4)</li> </ul> <p><b>[Menu item (3.5.3.1)]</b></p> <ul style="list-style-type: none"> <li>•Hides/shows a menu item</li> </ul>
 (Add elt.)	<p><b>[Headline edit]</b></p> <ul style="list-style-type: none"> <li>•For list frames, adds a single list element downwards.</li> </ul>
 (Append)	<p><b>[Menu item (3.5.3.1)]</b></p> <ul style="list-style-type: none"> <li>•Appends a child element for a menu item (3.5.4)</li> </ul>



Table 6.7: Blue Chameleon Content Management icons (continued)

Icon	[Context] and Action
  (Add elt up, down)	<b>[Various]</b> Respectively allow to add an element before (if not the first) or after (if not the last) current element. Occur for : <ul style="list-style-type: none"> <li>•list elements ;</li> <li>•form pages ;</li> <li>•form elements ;</li> <li>•menu/submenu elements.</li> </ul>
  (Displace elt up, down)	<b>[Various]</b> Allow to place element before the previous one (if not on the top) or after the next one (if not on the bottom). Occur in the same places as   except form pages.
 (Remove)	<b>[Various]</b> Deletes an element. Occurs in the same context as for   .

## 6.4 Form page elements : detailed

The following shows how the various form page elements (3.6.3.3) such as checkboxes, listboxes,... are composed. Whatever the kind of element, this is always achieved via the  icon.

### 6.4.1 "Active" elements


The following deals with elements on which, on the final form, actions such as checking, filling, selecting... will be done.

#### 6.4.1.1 Checkboxes

Fig.6.1 is how the checkbox choice as seen in "Form preview" (3.6.3.5) has been composed.

There, apart from the label that will appear on the final form :

- the range of values are given in a `<val>=<label>` manner ; the chosen `<val>` is the value that will be assessed to the CGI variable as given by 'Name', here `element20` ; a default value (checked when form is accessed) can be chosen ;
- how the checkboxes will be disposed is chosen, either on a single line or one below another ;
- the visibility for the form element is selected, from 'hidden'/'visible'/'editable'.

[Form element]  :

<b>Name</b>	element20
<b>Type</b>	Checkbox
<b>Subtype</b>	
<b>Label</b>	<input type="text" value="Instrument played"/>
<b>Value</b>	<input type="text" value="0=NONE&lt;br/&gt;1=Guitar&lt;br/&gt;2=Bass&lt;br/&gt;3=Drums&lt;br/&gt;4=Kazoo&lt;br/&gt;5=Other"/>
<b>Default</b>	<input type="text" value="0"/>
<b>Disposition</b>	<input type="text" value="one choice per line"/>
<b>Visibility</b>	<input type="text" value="editable"/>

Syntax for 'Value' is the following :  
<val>=<label>

**val** : the value that will be returned by the CGI. It can be anything (integer, string) but cannot contain the equal sign (=).  
**label** : text displayed for the choice.  
Choices are separated by ENTER. Empty lines are ignored.

'Default' contains the value (val) of the element which should be used as default.

Figure 6.1: Composing a checkbox selection for a form.

#### 6.4.1.2 Radiobuttons

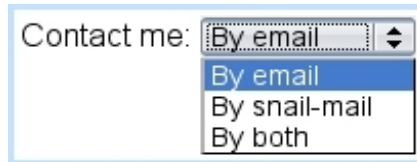
They are configured in the exact same way as for checkboxes, except that on the final form, of course one only item will be allowed to be selected.

#### 6.4.1.3 Listboxes

Listboxes, which provide three different styles, allows the same configuration method (<val>=<label>) and choices as for checkboxes, except there is no disposition choice.

The three styles are :

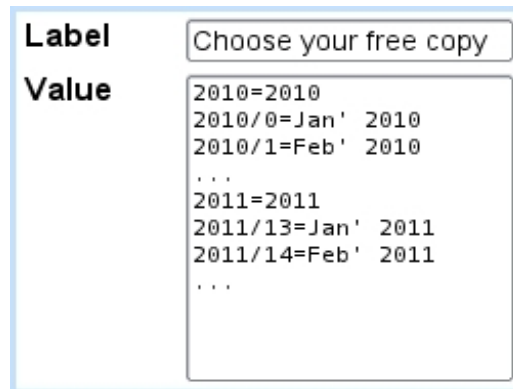
- 'single selection' : a standard drop-down menu, from which only one item can be chosen :



- 'multiple selection' : several items can be chosen, by clicking and holding **Ctrl** :



- 'chained selection' : two drop-down menus will be available, with the choices of the second (secondary list) depending on what was chosen for the first (main list). Main list elements are labelled `<valA>=<labelA>`, while secondary list elements `<valA>/<valB>=<labelB>` if this element B belongs to main list element A. For instance :



As a result, if for example '2011' is chosen in the first menu, only the corresponding choices for the secondary list (Jan' 2011, Feb' 2011,...) will be available :



#### 6.4.1.4 Input field and text area

The only configuring possibility for these two are :

- choosing whether input field will be 'normal' or 'password' (in that case hiding the typed content) ;
- choosing the *width x height* of the text areas, amongst various values (given in *number of characters x number of lines*).

#### 6.4.1.5 Form label - 'with check box'

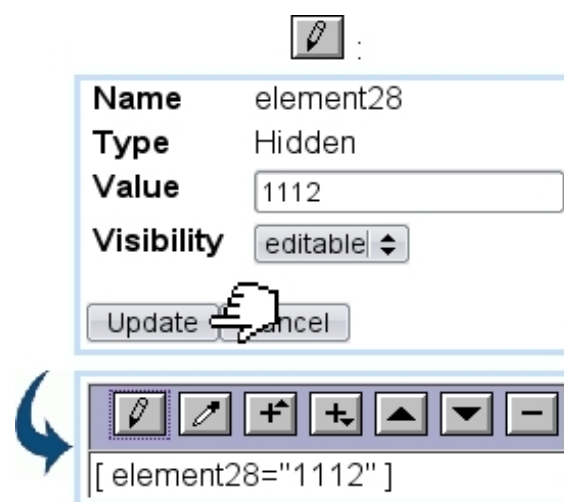
This choice for 'Label' allows to display a read-only text with a single checkbox before it.

### 6.4.2 "Passive" elements

These elements consist on graphical elements that will not be acted on the front-end form.

#### 6.4.2.1 Hidden field

A hidden field form object is simply assessed a value, which is then remembered during form edit :



#### 6.4.2.2 Form image

The Image :  *single image* choice for a form element allows to include an image into the form. Image selection is then similar to anywhere else in the Publisher :



#### 6.4.2.3 Form label - 'normal'

This choice for 'Label' type allows to display just a read-only text on the final version of the form.

#### **6.4.2.4 Form separator**

A form separator input is a centered horizontal line (corresponding to HTML's `<HR>`), which width is chosen amongst a range of percentages relative to page width.