

---

# Bitrix Site Manager 8.x

Integration Guide

---

# Introduction

---

This integration guide is for reading by the developers of web sites built using Bitrix Site Manager. The document discusses on the considers the integration of the software with a new or an existing web site. The guide presumes that the reader has knowledge of site development technologies like **HTML**, **CSS**, **PHP**.

The purpose of the document is to explain the main principles of the design integration and the key points of site template creation.

The guide does not cover all possible or exotic integration techniques. Beside this guide, we also recommend that you study the free online course: **Integration (BX-DEV001)**. Having studied the course and having passed the test, a specialist gets a certificate which confirms the specialist's knowledge in this field.

If you have any questions, you can send them to the [technical support](#) service, or ask in the Bitrix users [forum](#).

# Chapter 1. Creating and Deleting a Site

The demo version contains only one site whose main purpose is to present the software features. For more detailed study of the integration process, do not modify the existing demo site; instead, create a new one. Then, create a design template for your site. You can manage the sites in **Control Panel**, open the page *Settings > System settings > Sites > List of sites* (fig. 1.1).

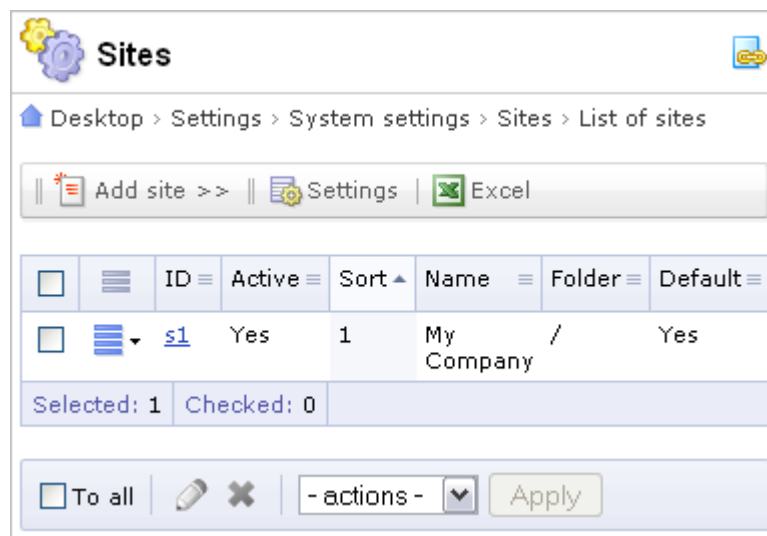


Fig. 1.1 A page showing existing sites

The process of creation and deleting sites is described in detail in the [System administration](#) training course.

---

## Chapter 2. Common Integration Principles

---

Bitrix Site Manager offers a wide range of handy features to help you integrate the system with an existing design. Active templates (the ones having executable code) provide flexibility and allow to implement simple, trivial templates as well as templates with arbitrary logic and individual design.

All this became possible due to separation of visual aspect (design) from information. The web site design is defined by the site templates and CSS files, whereas the site content is created using the means of the system modules and components.

The system components are based on similar principles. A component has a graphic constituent in the form of a template and a CSS file, and a logic constituent that renders information. Such separation is used in Components 2.0. The use of Components 1.0 is not recommended due to complexity of their adaptation to the site design.

The system allows to create as many templates as required.

**Note!** The essence of conversion of an HTML template to a PHP version is the substitution of HTML code with the PHP function calls and software components.

The sequence of actions that are to be performed to create a fully functional template is as follows:

- § Create a blank template in *Settings > System settings > Sites > Site templates*;
- § Add your **HTML** design to the blank template;
- § Insert the **#WORK\_AREA#** macro;
- § Replace **HTML** code with the **PHP** calls;
- § Apply the template to the site.

It is recommended to develop a site template on a local demo version. Then, export the ready template as files to a remote server and fill the site with text content.

The following features will make the template creation convenient:

- § the visual **HTML** editor;
- § the use of **Drag and Drop** technology to add visual components;
- § quick access to the component and object properties.

## The Structure of a Site Template

The main thing that define the site appearance is a design template.

A **design template** define the location of visual elements, the art style and the page rendering method. The template includes HTML code, images, CSS styles and any other additional files used to render the content. It can also contain the component templates and code snippets.

The site design usually consists of the three main ingredients (fig. 2.1):

- § The web page **header**.
- § The page **work area** holding the site content, components and any other code.
- § The web page **footer**.

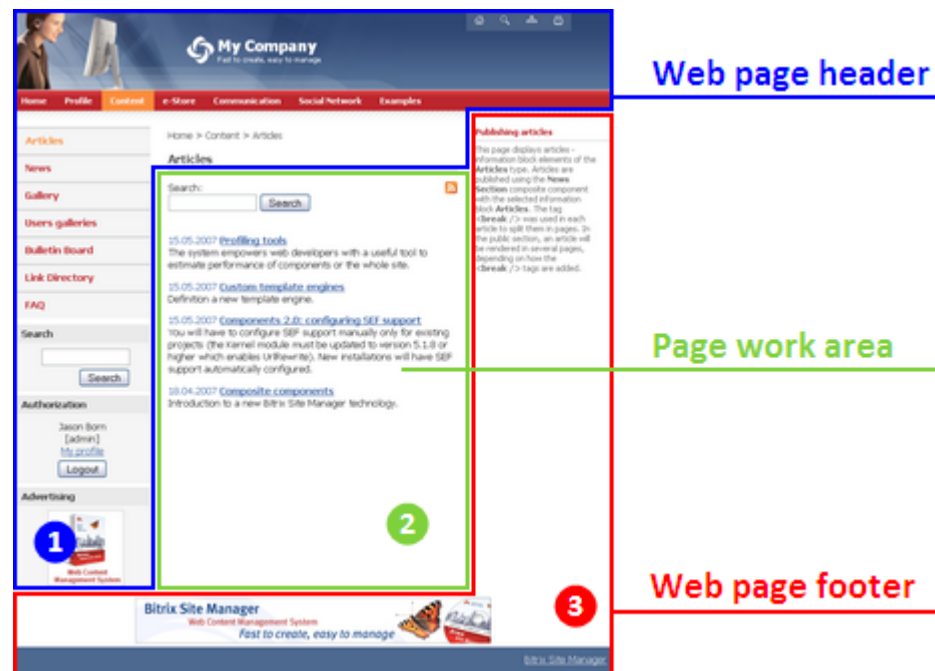


Fig. 2.1 Main design areas

The **Header** usually includes: the upper and the left part of a web page with static information (logo, motto etc.); the top horizontal menu and the left vertical menu (if these exist in the design). The header can also include dynamic information. The header is stored in an individual file `.../<template ID>/header.php`.

The **work area** is the place where the main content resides.

The **footer** contains: static information (usually contacts, copyright, the site owner etc.); the lower horizontal menu and the right menu (if these exist in the design). The footer can also include any information content. The footer is stored in an individual file `.../<template ID>/footer.php`.

Let us consider how these design parts interact in a template by the example of the **Books shop** template included in the installation package.

The design templates are managed in **Control Panel**:

- § Open the templates page: *Settings > System settings > Sites > Site templates*. This form allows to view and edit existing templates or add new ones.
- § Open the **Books shop** template by selecting **Edit** in the action menu.

The **Template** tab shows the appearance of the site design template. Since version 6.0, the site design templates can be created in the visual editor (fig. 2.2) using Components 2.0.

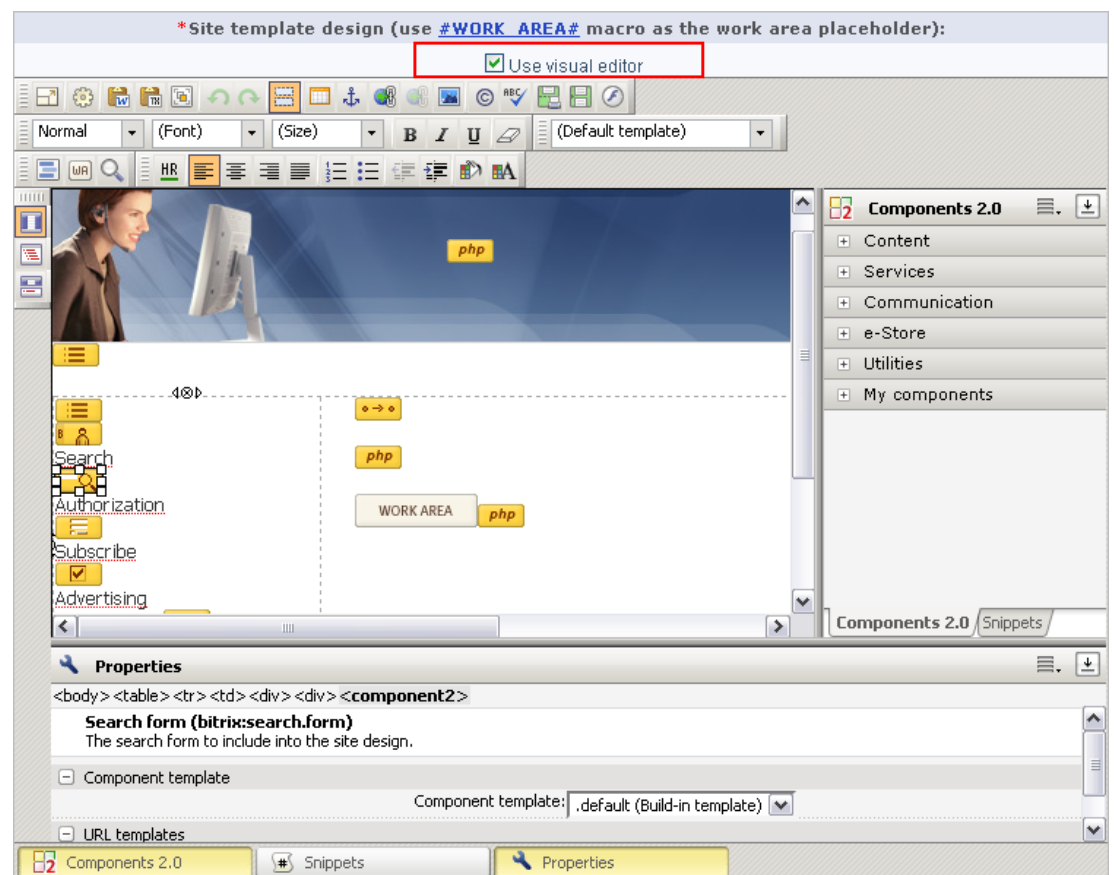


Fig. 2.2 Editing the template in the visual editor

Mainly, all components required that you would need to create a template are in the **Service** section. Other components can be found in the appropriate groups (**Content**, **Services** etc.)

You can toggle between visual and text template in the **Kernel** module settings (fig. 2.3).

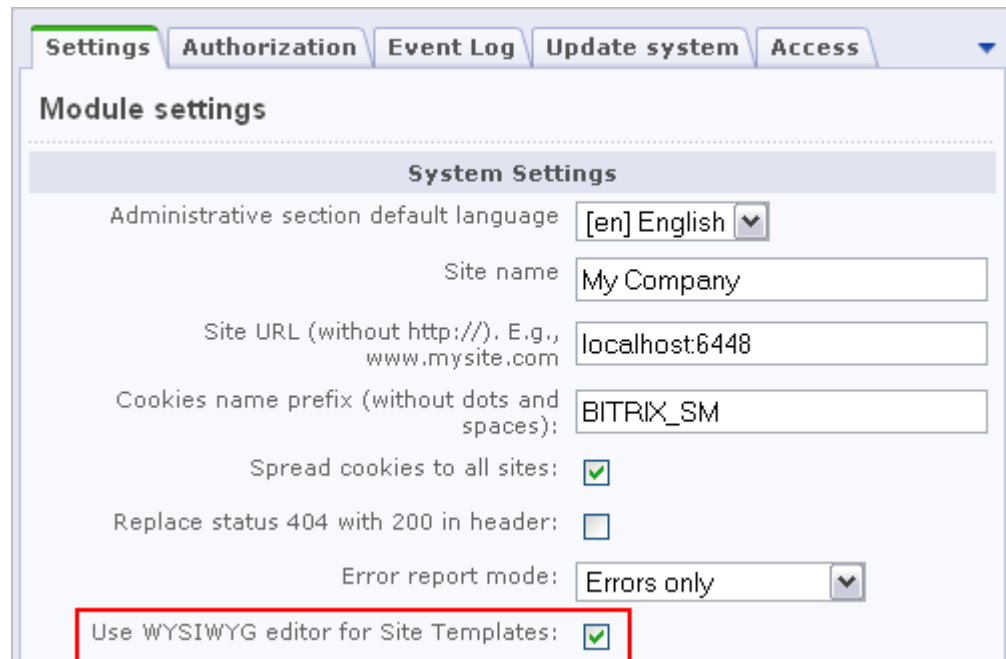




Fig. 2.3 The visual editor options

**Note!** A site design template can only be edited correctly if the attributes of HTML tags in the template do not contain PHP code and no PHP code islands are between the table rows and cells. If your design template code has such specificity, edit it as plain text.

The demo version templates are carefully prepared for HTML editing. It is not recommended to use this mode with other templates if you are not sure they conform the requirements: a template having irregular formatting can be lost when saving.

The visual editor shows the page **header** and **footer** as a monolithic block. The HTML code can be enriched with components and PHP functions to emit and render various information: metadata, page title, CSS, toolbars.

**Pay special attention** to the presence of a **#WORK\_AREA#** macro in your template: it demarcates the **header** and the **footer**. When a system processes the template, this macro is replaced with the actual page contents. You can insert this macro manually or by clicking  on the visual editor toolbar. When inserted, the macro shows as an image: . When creating a template, keep in mind that a template cannot be saved without the macro.

**Note!** The system places no restrictions on the appearance of templates or sites.

The CSS styles used in the template are recommended to be divided into two CSS separate files. Both files are stored in `/bitrix/templates/<template_id>/`. The first file, **styles.css**, contains styles used to render the page content (the work area). You will find the contents of this file on the **Site Styles** tab. The second file,

**template\_styles.css**, describes styles used to display the design template. This file is shown on the **Template Styles** tab.

All the templates are in the **/bitrix/templates/** directory. The template files are stored in a subdirectory whose name is the template ID. In the example, the template is located in the **/bitrix/templates/books/** directory.

**Note.** When you create a new template using the Control Panel interface, specify the template name, the description, the template HTML code, CSS styles, components and images. When the template is saved, the subdirectory **/bitrix/templates/<template\_ID>** is created automatically.

Each template has a uniform structure of files and directories. A simplest template can consist of the following files: **header.php**, **footer.php**, **styles.css**, **template\_styles.css**.

All the template images are located in the directory **/bitrix/templates/<template\_id>/images/**.

The template directory may also contain other files and components. You can view the template composition in Site Explorer by clicking the link beside the template ID.

Any design areas in the site template can be presented as individual files for quick access. Examples of such files can be the copyright area, contacts etc.

## Include Areas and Components

Some of the template elements are and can be implemented as active components. This simplifies the procedure of creating and managing a site.

Let us return to the **Books shop** template and learn the main components and include areas. Fig. 2.4 shows where the components and include areas are in this template.



Fig. 2.4 The template include areas

In the original **HTML** code, these areas are replaced with the appropriate component calls that emit the required data: metadata, the page title, CSS file inclusions, the Control Panel toolbar, the navigation chain, the site menus.

After you have created an **HTML** template prototype and added the required functions and components, you will have a ready-to-use active **PHP** template.

Information shown by include areas and components is available for quick editing in **Content** mode (the tab Content on the Control Panel toolbar, fig. 2.5).

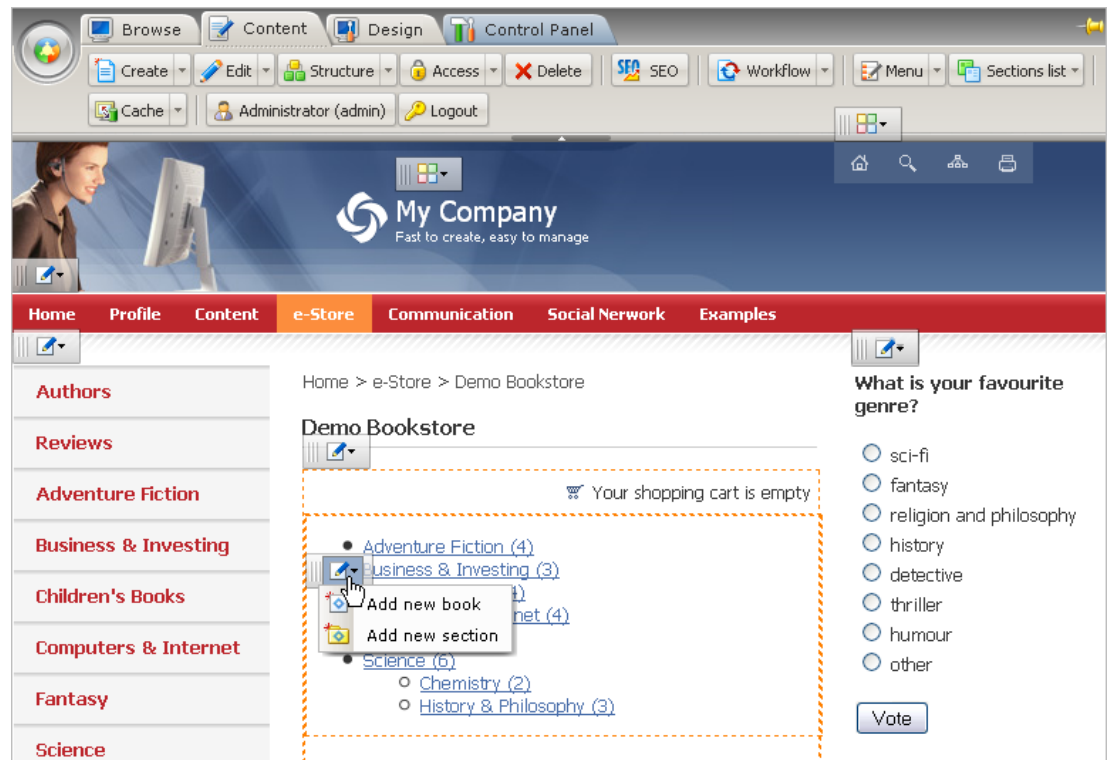
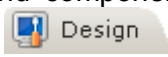


Fig. 2.5 Content edit mode

For example, you can use the information block controls to create a new element or a section (fig. 2.5).

The include areas are stored in separate files and can be conditional: for example, an include area can be shown only for a current section, a current page etc. The include area controls allow to edit the area content.

The parameters of include areas and components, site styles and templates are configured in site design mode (the  Design tab on the Control Panel toolbar, fig. 2.6).

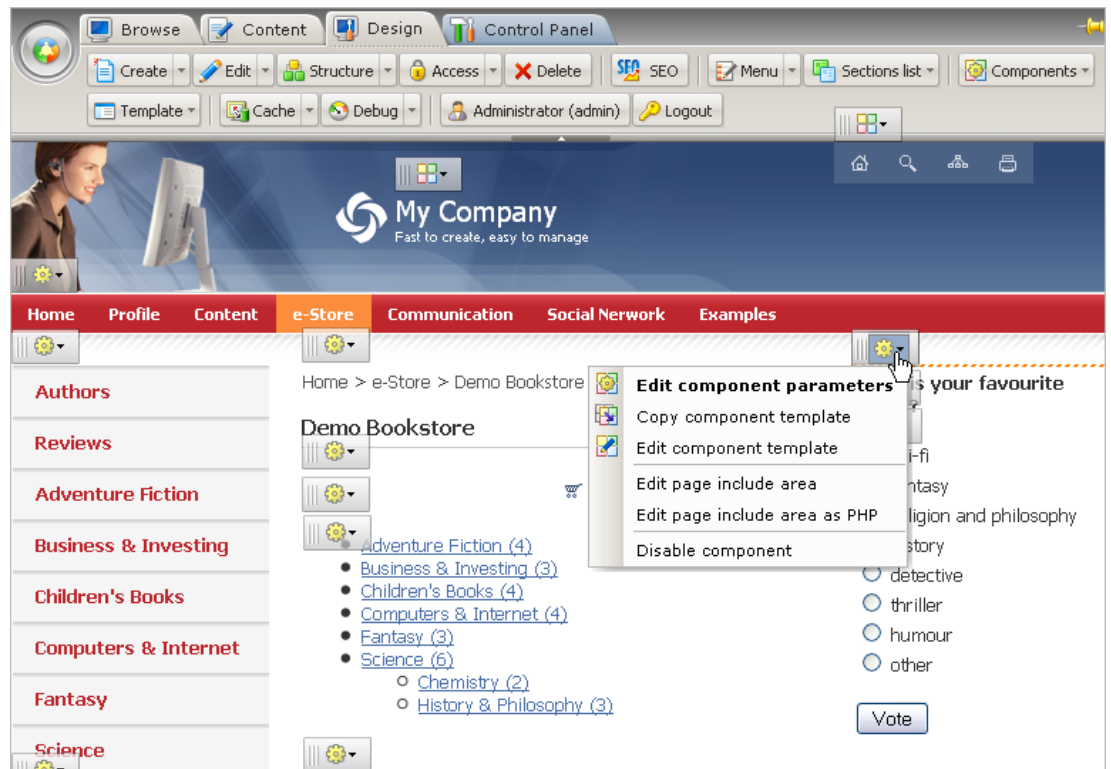


Fig. 2.6 Site design mode

The **Design** mode includes the site design and layout commands and the **Content** mode commands.

## Chapter 3. Integration Explained

---

### The File Structure

The Bitrix Site Manager architecture separates the view (the public section) from the software core. The active part of the system is located in the **/bitrix/** folder in the site root directory.

The subfolders in this folder contain the following files.

<i>/bitrix/templates/</i>	The site design templates and the user component templates. The files in this directory are the most important for proper integration of an existing design and the system.
<i>/bitrix/components/</i>	System components (in the <i>bitrix</i> namespace) and user components. System components can be modified by the update system. Never edit them manually.
<i>/bitrix/gadgets/</i>	System gadgets (in the <i>bitrix</i> namespace) and user gadgets. System gadgets can be modified by the update system. Never edit them manually.
<i>/bitrix/admin/</i>	The Control Panel interface. Contains management and property forms for all the system modules.
<i>/bitrix/cache/ /bitrix/managed_cache/ /bitrix/stack_cache/</i>	Cache files created by the system: dynamic information, access permissions, currency exchange rates etc.
<i>/bitrix/php_interface/</i>	Helper system files (database connection interface etc).
<i>/bitrix/modules/</i>	Class and function libraries for the system modules.
<i>/bitrix/images/</i>	Images for the system modules.
<i>/bitrix/tools/</i>	System and service files.
<i>/bitrix/updates/</i>	This folder is used by the update system.
<i>/bitrix/wizards/</i>	The system (in the <i>bitrix</i> namespace) and user wizards. The system wizards are modified by the update system; never edit them manually.
<i>/bitrix/</i>	This folder can contain additional service files.

## A Typical Page

A web site page is finalized by appending the *prologue* and *epilogue* code. In general, the structure of web site pages is the code similar to the following:

```
<?
require($_SERVER["DOCUMENT_ROOT"]."/bitrix/header.php");
$APPLICATION->SetTitle("Bitrix Site Manager 8.5");
?>
Page body goes here.
<?
require($_SERVER["DOCUMENT_ROOT"]."/bitrix/footer.php");
?>
```

## Page and Folder Properties

In Bitrix Site Manager, the page and section properties are used not only as entity dependent information storage. The properties is essentially the control mechanism allowing to manage the information display. A simplest example of page properties is the page metadata (the words you add to the META tag). A more sophisticated example: you can have the system to automatically change the site design depending on the current page. The variety of applications is virtually unlimited.

The page and section properties can be set using the web interface, or by calling the property related functions in the page script.

Some property names are reserved because they are used by the system functions. The reserved properties are:

- § **title** – this property is used to set the extra title for a page;
- § **adv\_desired\_target\_keywords** – specifies the desired advertising keywords for a page;
- § **not\_show\_nav\_chain** – shows or hides the navigation chain (i.e. navigation breadcrumbs) on a page or a site section.

## Creating and Editing the Site Templates

The following actions are required to be performed in order to create a new site design template.

- Open *Settings > System settings > Sites > Site templates*.
- Click **Add template** on the context toolbar. The following form will open (fig. 3.1):

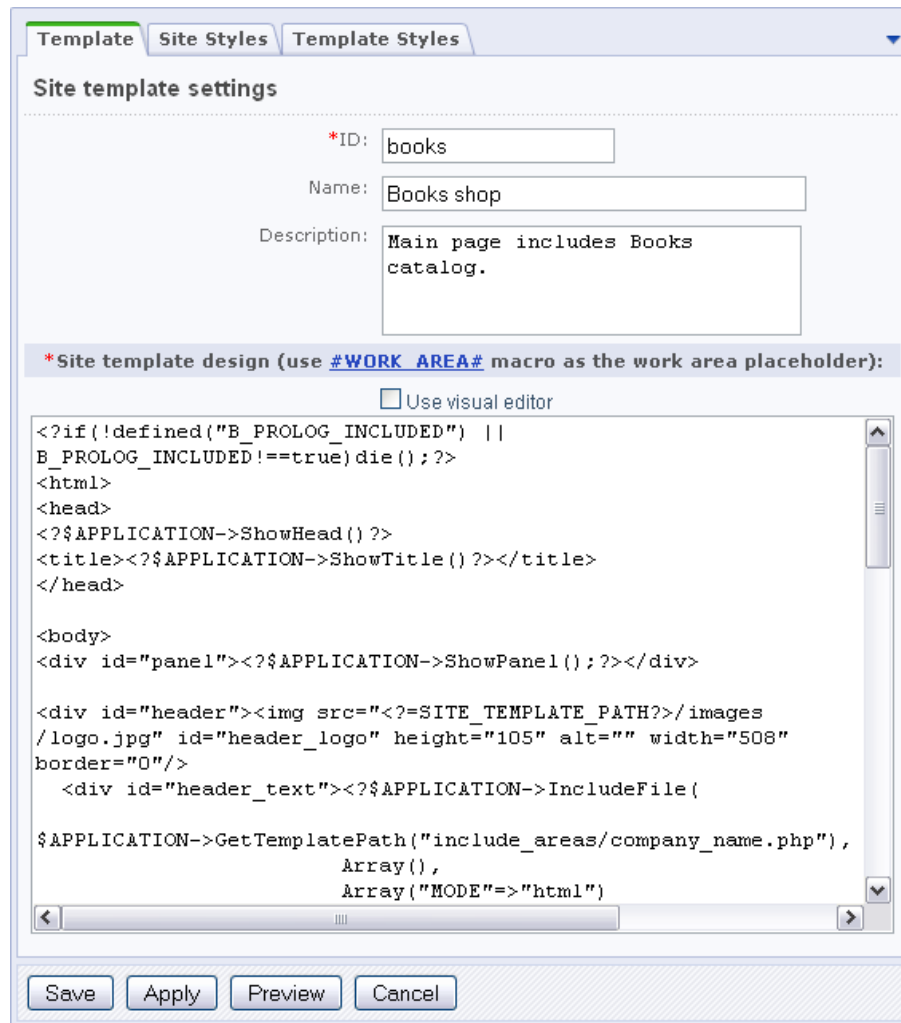


Fig. 3.1 The site template editor

In future, you can edit an existing template:

- § by clicking the **Template** button on the control panel toolbar in the public section (fig. 3.2):

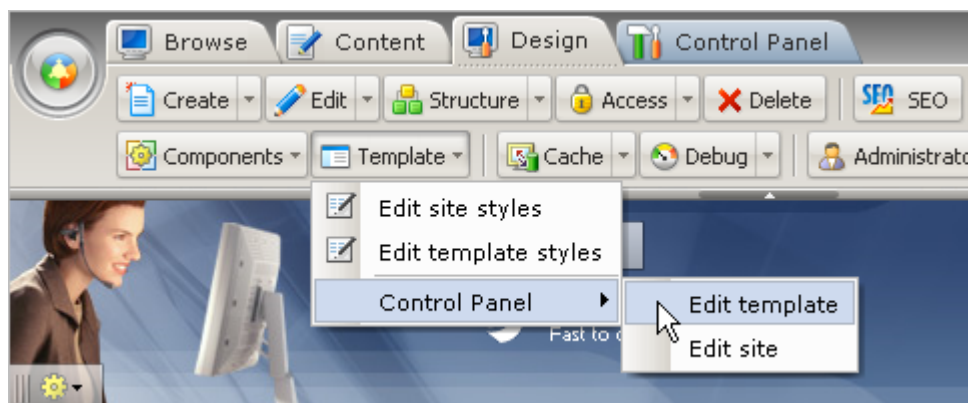


Fig. 3.2 Opening a template for editing in the public section

§ or by selecting **Edit** in the action menu (fig. 3.3):

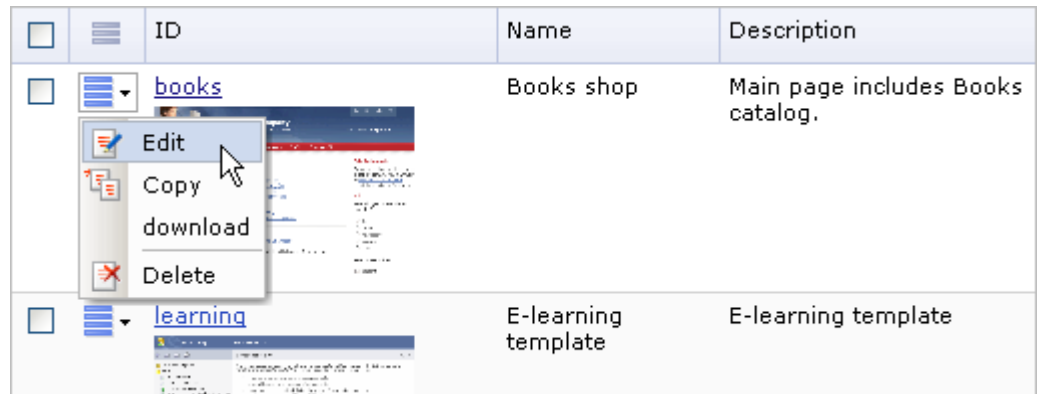



Fig. 3.3 Opening a template for editing in Control Panel

### Creating Service Areas

If you use the visual HTML editor to create or edit templates, you can still access the template service areas. To open these areas for editing, click **Edit Template Areas**  on the editor toolbar.

The area editor consists of the two tabs: **Top Area** and **Bottom Area** (fig. 3.4).

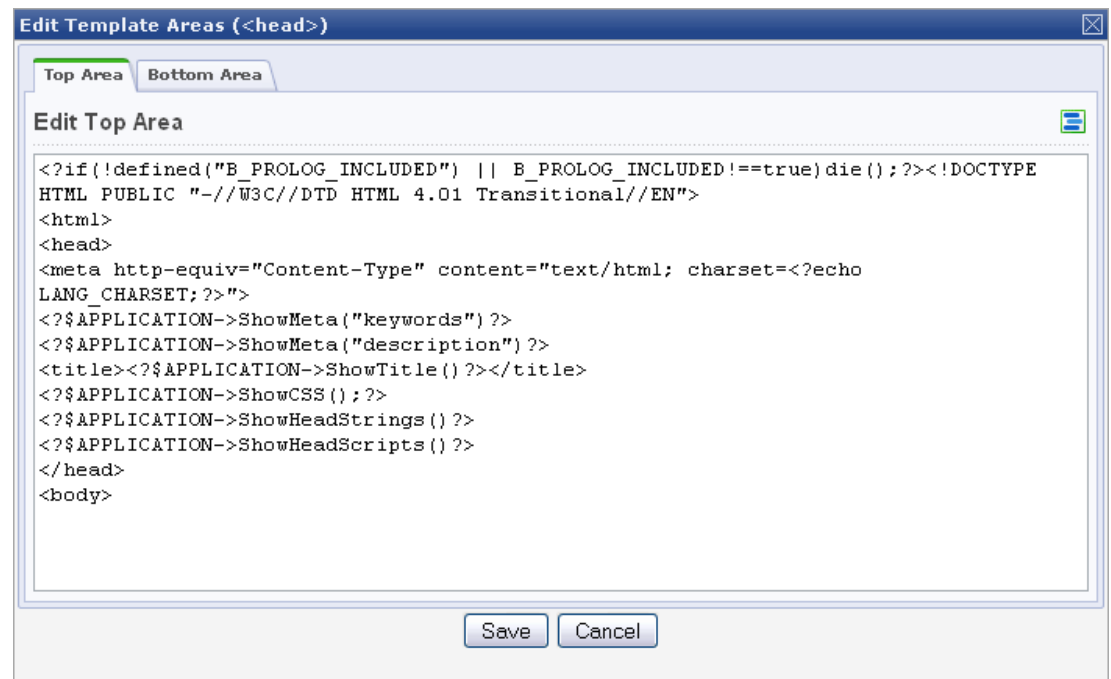



Fig. 3.4 The service area editor

The first tab shown the template portion that will appear before the **<body>** tag. You can reset it to the default contents by clicking  (fig. 3.4); this will insert the standard functions: page encoding definitions, title, page metadata, CSS file etc. Similarly, the **second** tab is used for the lower area of the template.

**Attention!** The functions *ShowMeta()*, *ShowTitle()*, *ShowCSS()* etc. can initialize the header elements anywhere in a script or a component. For example, a page title can be set even after the script output has been emitted. Thus, now you can set the page title in the page work area (i.e. in the **body** tag).

### Setting the Page Encoding

A proper charset configuration is required to make texts on your pages show correctly.

Each language used in the public section is configured individually in **Control panel** here: *Settings > System settings > Sites > List of sites*. You can set the encoding, time and date display format for each site individually (fig. 3.5).

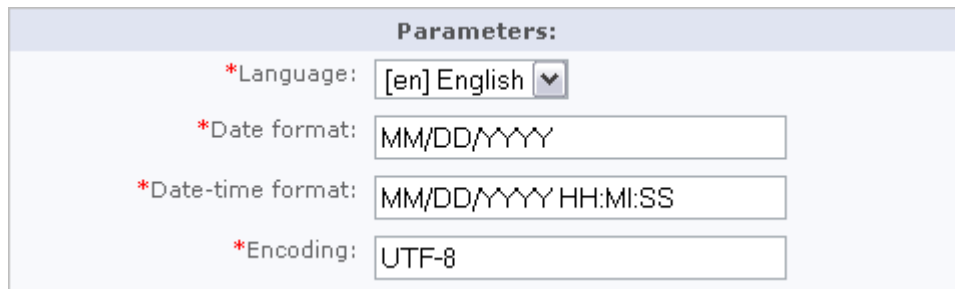


Fig. 3.5 The public section language parameters

**Note!** The system supports the **UTF-8** encoding for **MySQL** and **Oracle** editions starting from version 7.0.

The page encoding is initialized in the **<head>** tag of the prologue by resolving the **PHP** constant which obtains the encoding value from the current site language parameters:

```
<head>
...
<meta http-equiv="Content-Type" content="text/html; charset=<?echo
LANG_CHARSET?>">
...
</head>
```

### Metadata Control

A good example of metadata control is the mechanism to specify the page key words, and description. By default, the system is preconfigured to manage these types of metadata, but you can add your own metadata types in a similar way.

Users can manage the metadata in the page or section property forms.

The function *ShowMeta* renders the specified metadata in the page code:

```
<head>
...
<?$APPLICATION->ShowMeta("keywords")?>
<?$APPLICATION->ShowMeta("description")?>
...
</head>
```

These calls will result in the following **HTML** code added to the page code (for example):

```
<meta name="keywords" content="CMS, PHP, bitrix" />
<meta name="description" content="Bitrix Site Manager" />
```

Note that the page properties can be initialized dynamically. For example, for pages showing the catalog items, the page properties (**keywords** and **description**) can be set according to the required values of information block elements.

Thus, you can create the **keywords** and **description** properties and add their values to the page output dynamically.

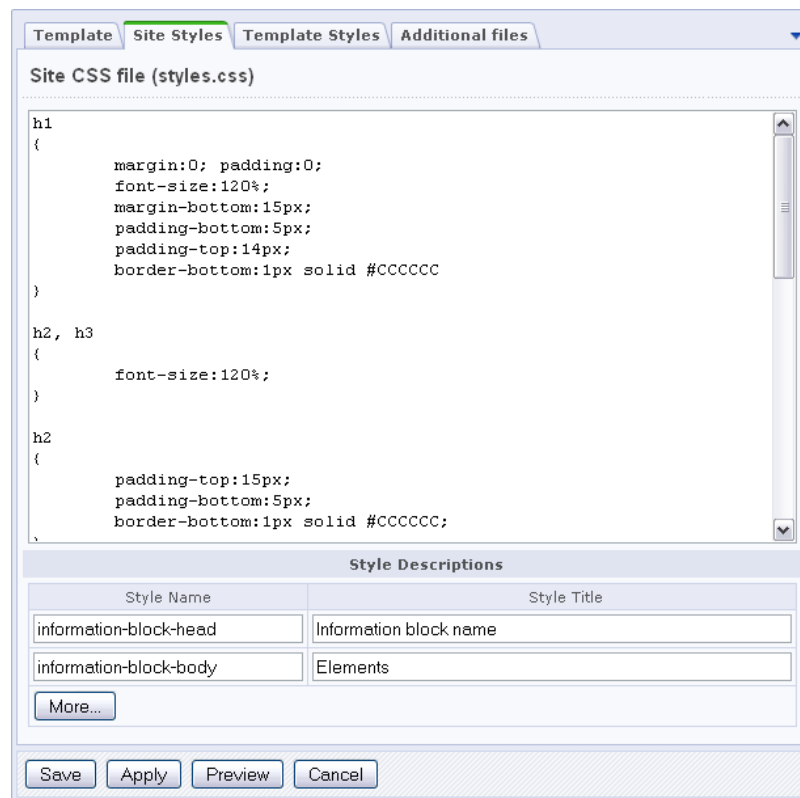
### Including a CSS File

Bitrix Site Manager uses several style sheets to format different types of elements. The following CSS files are used:

- § **styles.css** – contains styles for the **page content**.
- § **template\_styles.css** – contains styles for the **site design template**.

When copying styles from a CSS file of an original design template to a site template, you should discern the styles that are used to render the site content from those used to render the design elements.

It is important to give the styles meaningful names. The names have to be created for those styles in **styles.css** that are planned to be used for editing pages in the visual HTML editor because they will be listed in the styles combo box. You can create the style names in the design template editor page, in the **Site Styles** tab (fig. 3.6).



*Fig. 3.6 Editing the site styles*

You will find these styles in the visual editor toolbar combo box. The names defined here will be stored in the **<template\_ID>/styles.php** file.

A design template style sheet (the *template\_styles.css* file) can be created in the **Template Styles** tab (fig. 3.7).

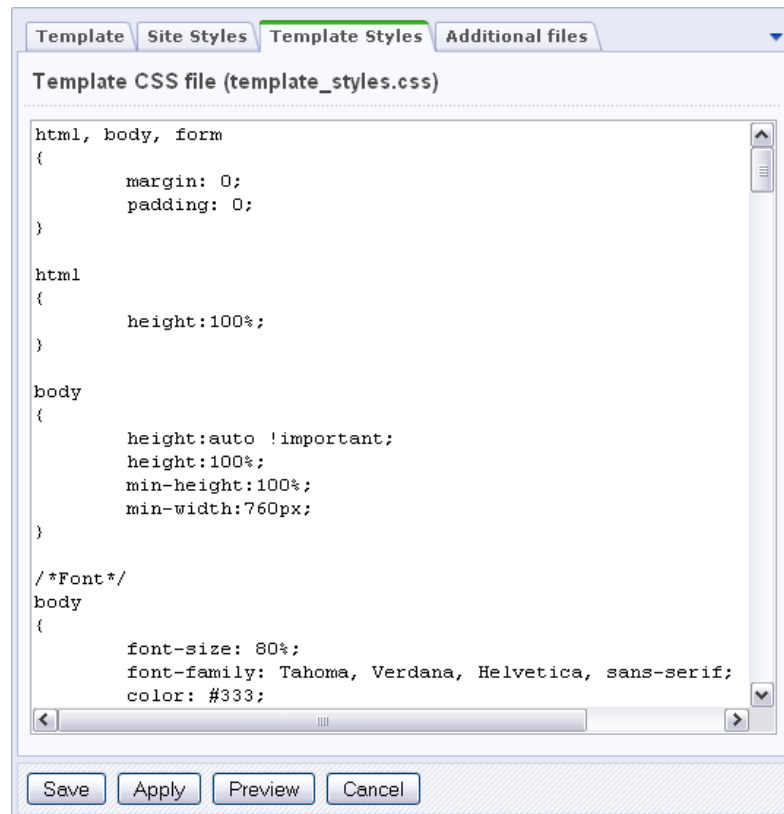


Fig. 3.7 Editing the template styles

To include the CSS files, only a single PHP call is required in the *head* tag scope:

```
<?
$APPLICATION->ShowCSS();
?>
```

This function prints the code that will import: the file **styles.css** and **template\_styles.css** of a current template; all additional styles defined for this page by the **SetAdditionalCSS()** function; the styles for components used on the page.

If the **ShowCSS()** function is called without parameters, the styles will be printed in the following format:

```
<LINK href="/bitrix/templates/books/styles.css" type="text/css"
rel="STYLESHEET"/>

<LINK href="/bitrix/templates/books/template_styles.css" type="text/css"
rel="STYLESHEET"/>
```

The styles set by **SetAdditionalCSS()** will be included using **require()**.

For pages, the visual editor imports **styles.css**. For templates, **template\_styles.css** and **styles.css** are imported.

Some of Components 2.0 can have their own style sheets that are included automatically.

## Showing the Control Panel Toolbar

After a user has authorized, the Control Panel toolbar may become available for them at the page top (fig. 3.8) if a user has appropriate access permissions.

The toolbar offers many functions to help a user:

- § configure the current section parameters;
- § edit a current page and its include areas;
- § add or modify the current section menu;
- § configure the component parameters;
- § quickly switch to Control Panel, etc.



Fig. 3.8 The Control Panel toolbar

In order to have the **Control Panel** toolbar visible in the template, add the following code right after the **<body>** tag:

```
<?
$APPLICATION->ShowPanel();
?>
```

You will find the detailed discussion on the Control Panel toolbar in the Bitrix Site Manager [help section](#).

## Adding HTML code

The large **Site template design** field of the **Template** tab is essentially the source code editor in which you modify the template HTML code. A well formed template must have the **#WORK\_AREA#** separator that splits the code into the **footer** and the **header**. The template cannot be saved without this separator.

Images, logos and other elements can be uploaded using the Bitrix Site Manager functions, or by any other method available to a user. You are recommended to use the system tools to upload images to a remote server in order to avoid file permission problems.

Image files should be stored in the **/bitrix/templates/<template\_ID>/images** folder. This ensures that all the images will be added to a back-up archive when exporting or copying the template so that you will not have to upload the images again.

Having uploaded the images, change paths to image files in the template HTML code.

## **Adding the Components**

When you add the components to a template, you have to adhere the following sequence of actions.

- Delete the portion of HTML code that used to implement functionality to be relayed to the component.
- Add the component call instead of that HTML code.
- Modify the component template and its CSS file to match the site design (if required). The detailed description of the component integration can be found in the chapter [Working with components](#).

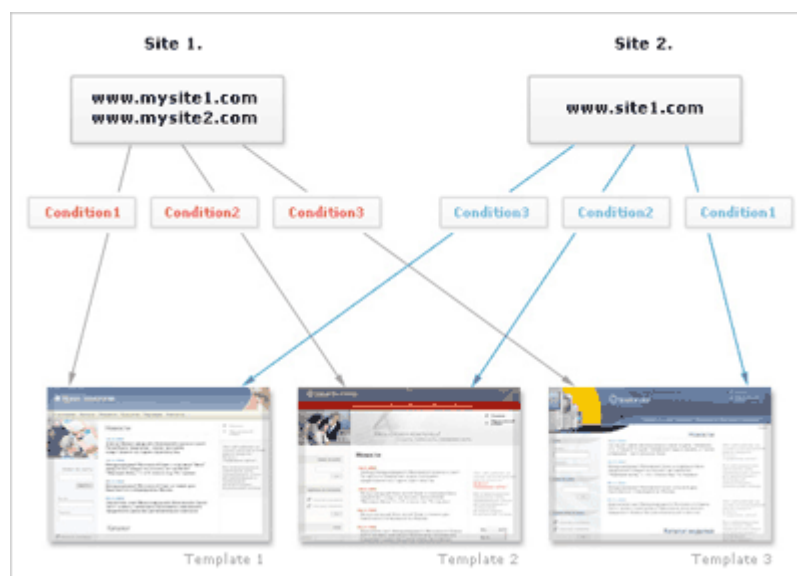
You can add as many components to a site template as required: the site navigators; advertisers; include area creators etc.

## **Applying Templates to Sites**

Bitrix Site Manager supports the creation and operation of more than one site. Each site can have an individual domain name, design, interface language and information content.

An unlimited number of templates can be assigned to each site. The use of templates unveils wide possibilities for the site design configuration according to various conditions. A template can be further reused for any other sites.

The software provides for flexible configuration of variety of designs for the sites and site sections. You can apply a special festive design for a certain period; create individual design templates for different site visitors depending on a certain URL parameter, etc. (fig. 3.9).



*Fig. 3.9 Sites and templates*

The templates are assigned to sites in the **Control panel**: *Settings > System settings > Sites > List of sites*.

For each design template, a condition can be specified on which the template will be assigned to the site pages (fig. 3.10). If no condition exists, the template is applied by default. Consider how the following conditions apply to the demo site.

*Template	Sort.	Condition type	Condition
Books shop	150	[no condition]	<no condition>
Print version	150	URL parameters	print = Y
E-learning template	150	For file or folder	/communication/learning/cou ...
(not set)	151	[no condition]	<no condition>
(not set)	152	[no condition]	<no condition>
(not set)	153	[no condition]	<no condition>


Fig. 3.10 Template application conditions

According to the conditions above, the design templates will be applied to the site pages in the order:

- § **Books shop** is the default template since it provides no condition.
- § The **E-learning template** will be used for pages in */communication/learning/course/* folder.
- § The **Print version** template takes effect if the page URL contains **print=Y**. For example, the URL <http://www.site2.com/?print=Y> shows a print version of the index page.

**Note!** The condition can contain any PHP code including API function calls. For more information, see the API help section.

When editing the template application conditions, always specify the sort index. The order in which the system attempts to apply the templates is defined by the sort index. If a situation occurs when more than one conditions are true, a template with a higher sort index will be applied. So, if you fail to specify the sort index, a wrong template may be used.

You can check whether the template is configured correctly by clicking  near the template drop-down list. You will see the site with the selected template applied in *preview* mode without actually applying the changes.

When creating a site template, various conditions can be used that affect the display of template elements in different site sections. For this, define a section property whose value will be checked in the site template:

```
<?if ($APPLICATION->GetProperty("SECT_PROP")=="Y"):??>
```

Such conditions can be introduced for any template element. For example, you can disable the include areas etc.

## Working with Components

Frequently used site areas can be formed as software components. Virtually any program script can be wrapped in a component.

Since version 6.0, the system uses new paradigm for components. The conceptual difference between the components 2.0 and 1.0 is that the components 2.0 separate the view from the business logic. More than one view can be created for a single logic unit. Views can depend on a current site template. A view (presentation template) can be created in any template language that can be used with PHP. For example: PHP, Smarty, XSL etc. There is no need to alter the component logic to modify the view. This has made the information display control much easier: a developer does not need to look deep into the component logic.

Physically, the components are located in a special folder (*/bitrix/components/*). This makes the site structure more robust and comprehensive. The folder has read access, which means that a component and its templates can easily use their additional resources. Components are the part of the *component namespace*. For example, all system components belong to the **bitrix** namespace and are stored in */bitrix/components/bitrix/*.

**Note!** Never modify the content of this folder manually. The update system will overwrite files in it; you will lose all the changes you made.

User components should be stored in a separate folder, for example: */bitrix/components/my\_folder/*; or in any subfolder of */bitrix/components/*.

Components 2.0 can be **simple** (single page) or **composite** (multiple page). **Simple** components create an area on a page surface (for example: news, table of currency rates etc.). **Composite** components create a *site section*. For example, the commercial catalog component creates the whole catalog section including the catalogs page, the groups page and the product pages. Essentially, from the visitor's point of view a composite component is a set of virtual pages represented by a single physical page.

The component names have the format "**identifier1.identifier2...**". For example: **catalog**, **catalog.element**, **catalog.section.list** etc. A good idea is to create hierarchical names, from common to particular, e.g. **catalog.section.elements** for a component that would display products of a certain group.

The component 2.0 folder contains the following subfolders and files:

- § subfolder **templates** in which the component view templates are stored. This folder is optional if the component has no view templates;
- § file **component.php** containing the component logic. This file must exist;
- § file **.parameters.php** containing the description of the component input parameters for the visual editor. This file must exist if the component uses input parameters;

- § folder **images** containing the images required by the component;
- § file **.description.php** containing the name and the description of the component. Additionally, it describes the disposition of a component icon in the visual editor component pane;
- § subfolder **help** containing component help files(optional);
- § subfolder **lang** containing language dependent messages (optional).

### Adding Components to a Page

A special interface has been integrated in the visual HTML editor to allow users quickly add components to their pages. When creating or editing a page, you select the required component and drag it to the page body.

After a component has been placed in the page body, the component parameters can be configured in the **Properties** bar. Later, you can select the component to view its parameters (fig. 3.11).

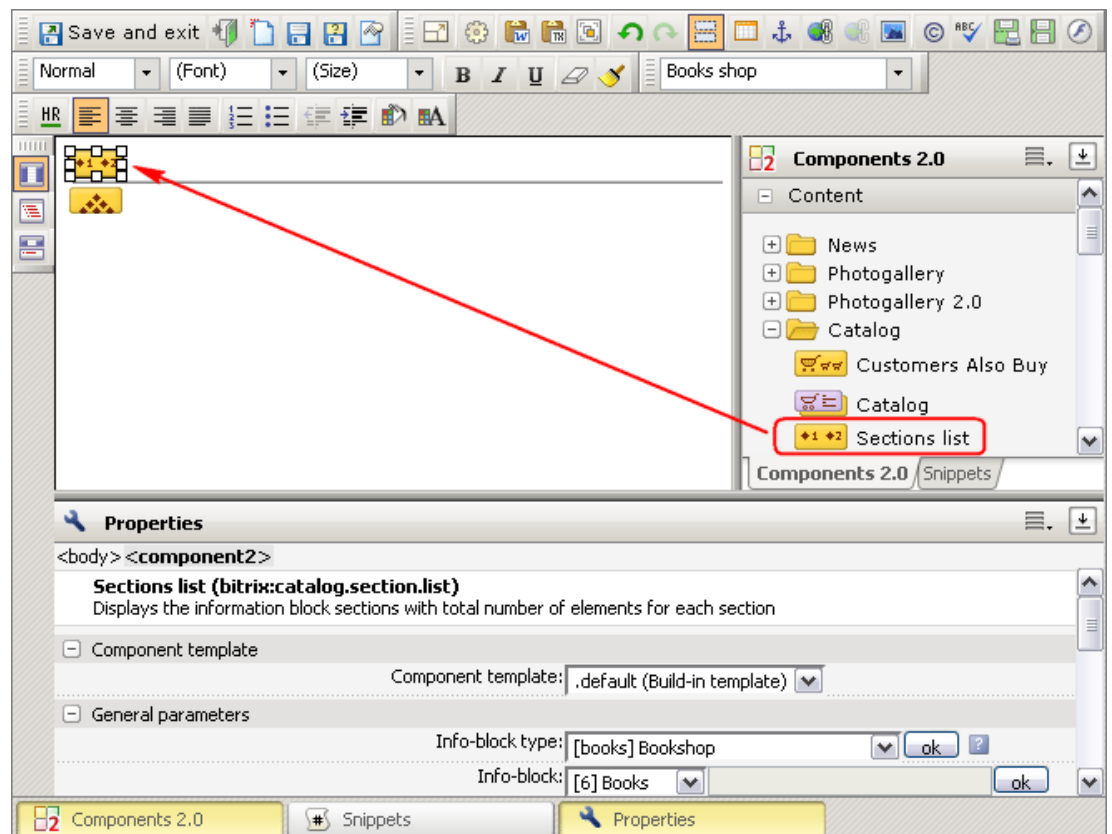


Fig. 3.11 Adding a component to a page

Internally, the components are inserted into a page by writing the **IncludeComponent()** function call. This function takes the following input parameters: the component name, the template name and the component parameters specified by a user.

For example, the following call inserts the component **Section List (bitrix:catalog.section.list)**:

```
<?APPLICATION->IncludeComponent(
    "bitrix:catalog.section.list",
    "",
    Array(
        "IBLOCK_TYPE" => "books",
        "IBLOCK_ID" => "6",
        "SECTION_ID" => $_REQUEST["SECTION_ID"],
        "SECTION_URL" => "/e-store/books/#SECTION_ID#/",
        "COUNT_ELEMENTS" => "Y",
        "TOP_DEPTH" => "2",
        "DISPLAY_PANEL" => "N",
        "ADD_SECTIONS_CHAIN" => "Y",
        "CACHE_TYPE" => "A",
        "CACHE_TIME" => "3600"
    )
);?>
```

## Managing the Component Content

The content rendered by a component can be managed in both the public section and Control Panel. To create and edit the site content directly in the public section, click the **Content** tab. Each component has a button that shows the content management menu (fig. 3.12).

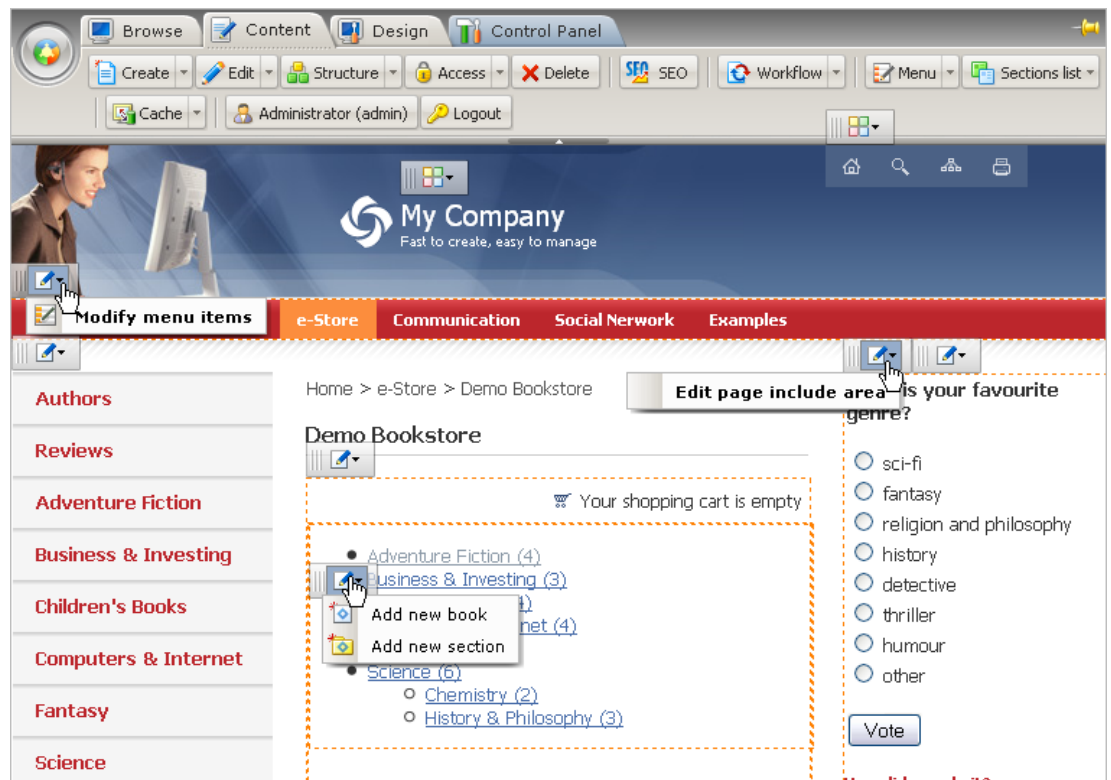


Fig. 3.12 The content management buttons

These menu items are also available in site design mode (the **Design** tab). They are isolated from the component management commands with a menu separator (fig. 3.13).

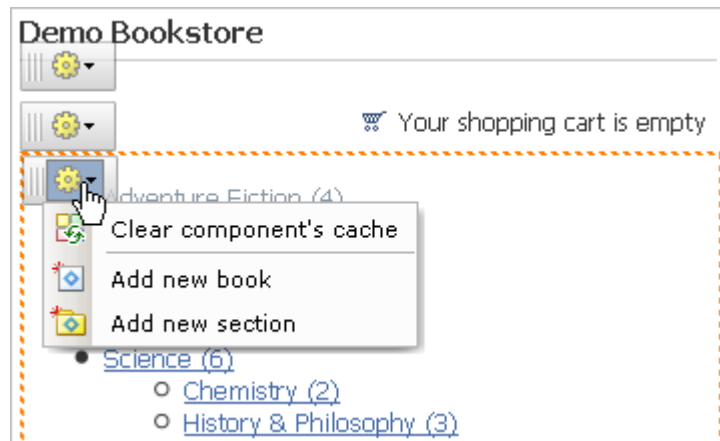


Fig. 3.13 The component menu

The set of content management commands may vary depending on a component and the information it displays (fig. 3.12).

The most typical commands are:

- § **Create element** - opens the information block element creation form.
- § **Create section** - opens the information block section creation form.
- § **Edit menu items** - opens the menu items editor form.
- § **Edit include area** - opens the include area content editor form. May vary depending on the component settings. For example, *edit as HTML* or *PHP*, edit a *page area* or a *section area*.

### Editing the Component Parameters

The system allows to configure the component parameters directly from the public section in site design mode (the **Design** tab). You can switch to the component parameters form:

- § by selecting **Edit component parameters** in the component command menu (fig. 3.14);

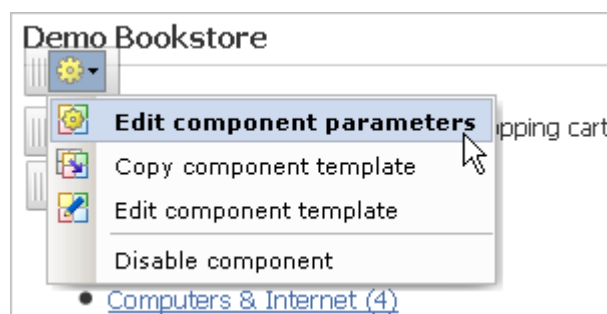
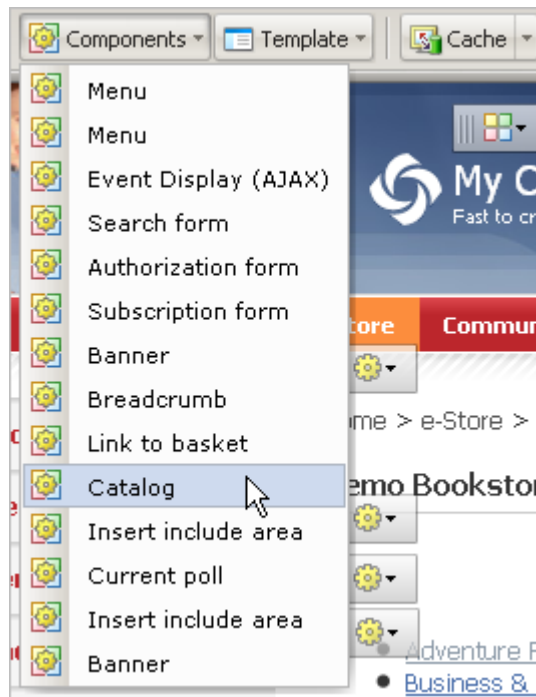


Fig. 3.14 Opening the component parameters

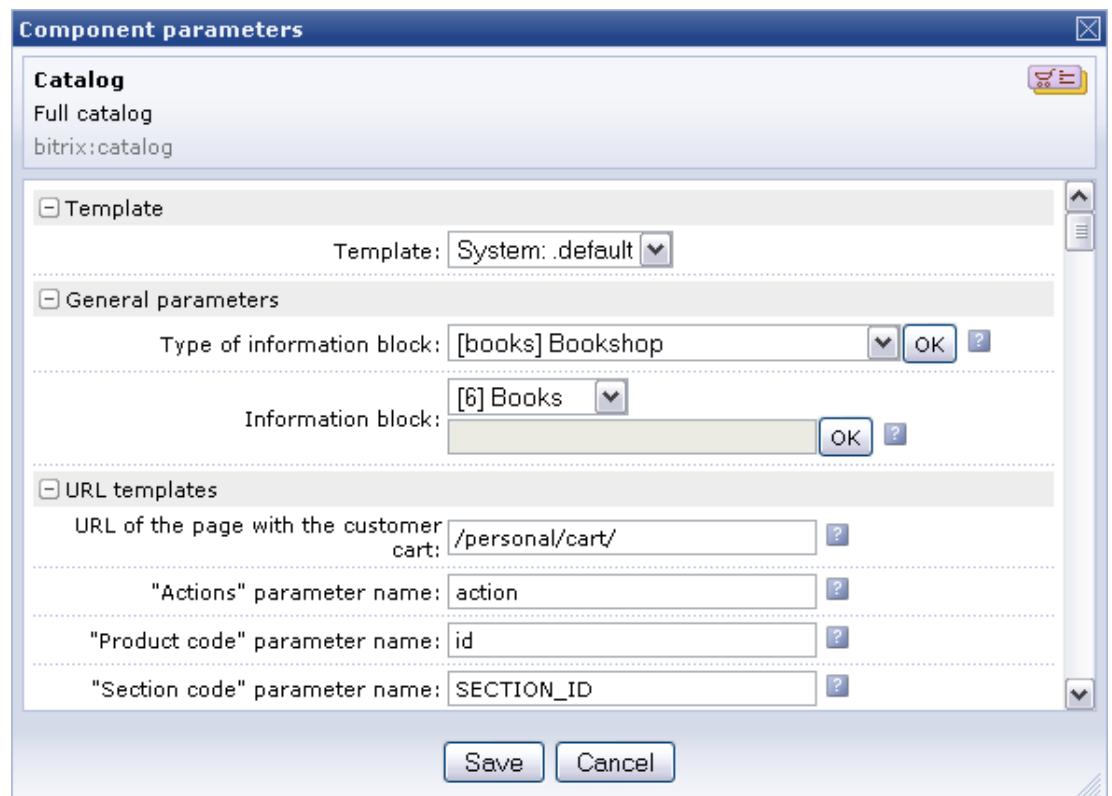
**Note!** This menu item is visible for all components except subcomponents of a composite component.

§ by selecting the menu item titled by the component name (fig. 3.15).



*Fig. 3.15 Selecting a component by the name*

The following component configuration form will open (fig. 3.16):



*Fig. 3.16 The component parameters form*

## Copying a Component Template

If you need to customize a component template to a certain site requirements, you should first copy the template to the site template folder, and only then, you can edit the copied template (see also: [Editing a component template](#)).

The system provides the convenient public interface to copy the component template. Just follow the instructions below.

- Switch to the site design mode (the **Design** tab).
- Select **Copy component template** in the component control menu (fig. 3.17).

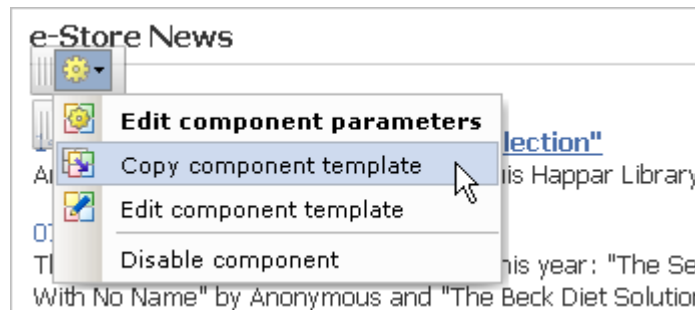


Fig. 3.17 Opening the template copy form

- Specify the template copy parameters in the form opened (fig. 3.18).

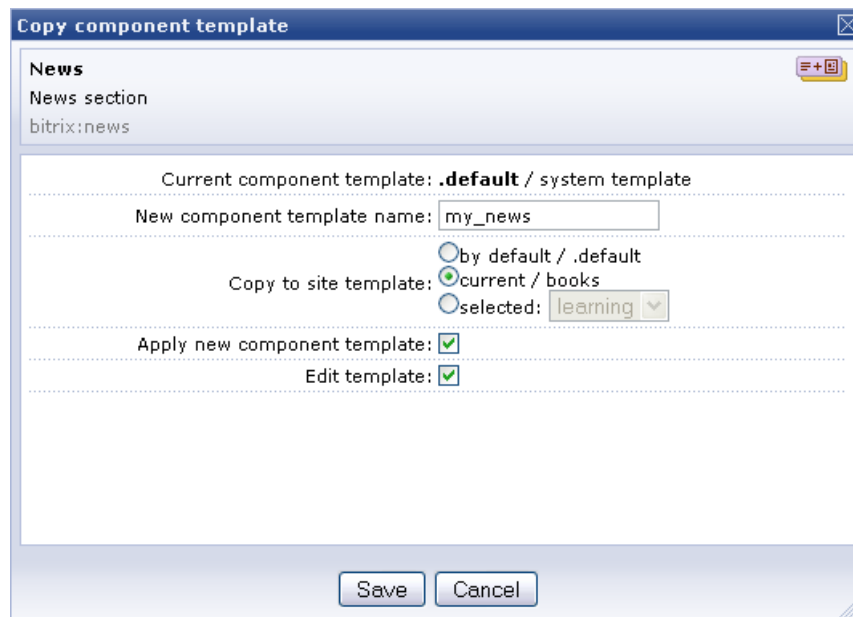


Fig. 3.18 The component template copy form

A component template copied to a site template becomes a **custom** template. Now you can modify as you wish, according to the design requirements.

## Editing a Component Template

Before you edit any component template, you have to copy it to the site template folder as described in the previous chapter.

**Note!** Modifying the system components and their templates that are in the `/bitrix/components/bitrix/` folder can have unpredictable effects. The content of this folder is updated by the update system and must not be modified by users.

You can quickly edit the template component from the public section in **Design** mode. Just click select **Edit component template** in the component control menu (fig. 3.19).

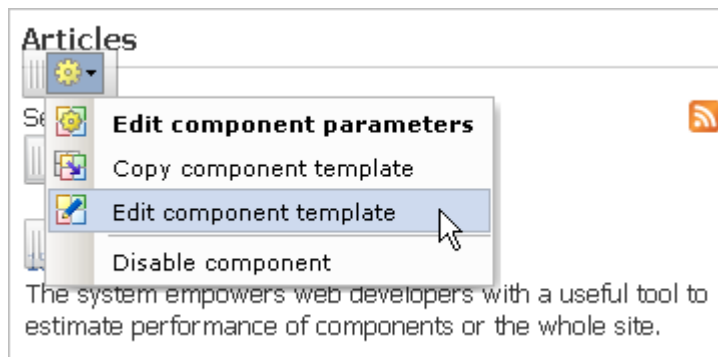


Fig. 3.19 Opening the component template editor

For **simple components**, the editor shows the contents of the `template.php` file (the simple component template, fig. 3.20).

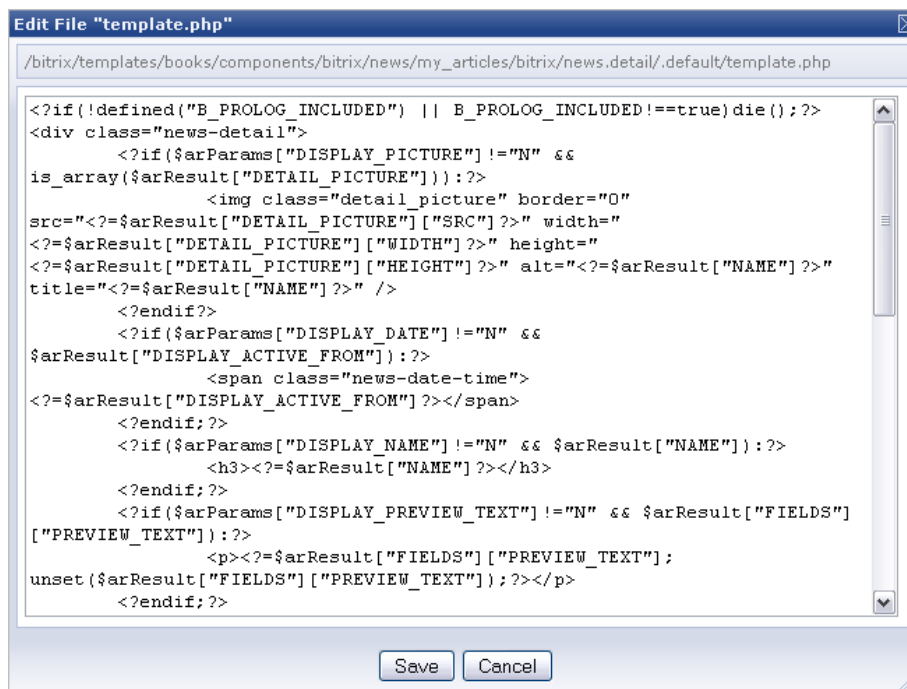
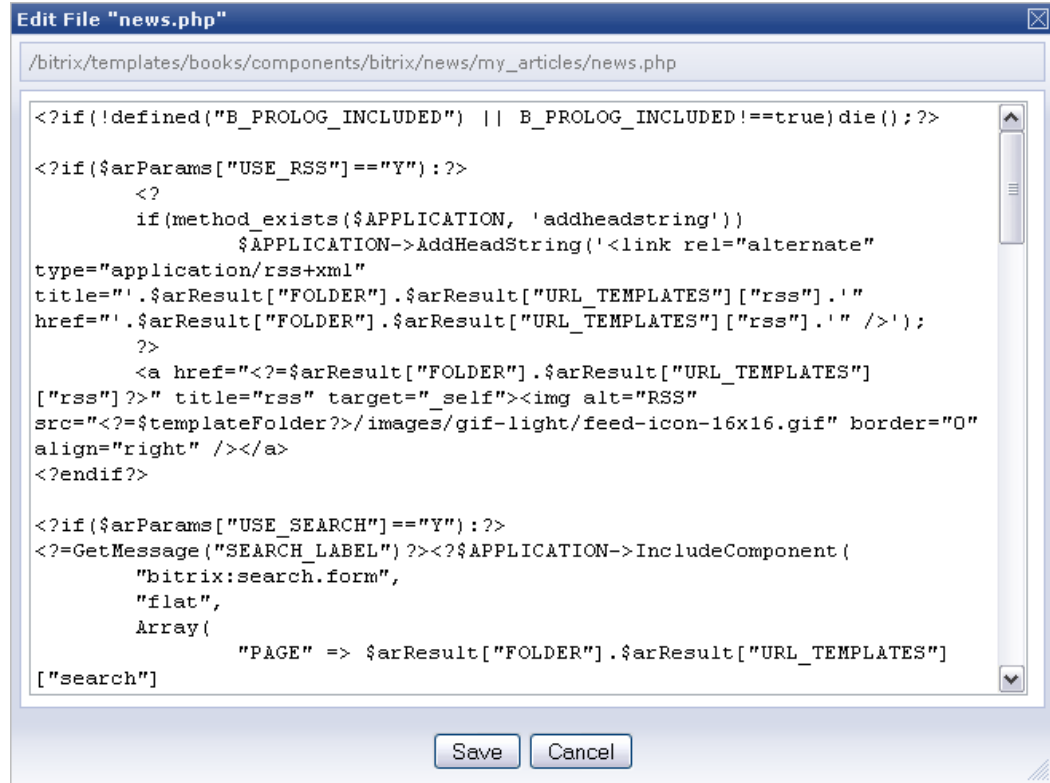


Fig. 3.20 An example of the simple component template

However, with **composite components**, you will encounter the ***component\_name.php*** file which is a template file defining the disposition of the subcomponents, fig. 3.21.



```

Edit File "news.php"
/bitrix/templates/books/components/bitrix/news/my_articles/news.php

<?if(!defined("B_PROLOG_INCLUDED") || B_PROLOG_INCLUDED!==true)die();?>

<?if($arParams["USE_RSS"]=="Y")?:?>
    <?
        if(method_exists($APPLICATION, 'addheadstring'))
            $APPLICATION->AddHeadString('<link rel="alternate"
type="application/rss+xml"
title="'. $arResult["FOLDER"]. $arResult["URL_TEMPLATES"]["rss"]. "'
href="'. $arResult["FOLDER"]. $arResult["URL_TEMPLATES"]["rss"]. "' />');
    ?>
    <a href="<?=$arResult["FOLDER"]. $arResult["URL_TEMPLATES"]
["rss"]?>" title="rss" target="_self"></a>
<?endif?>

<?if($arParams["USE_SEARCH"]=="Y")?:?>
<?=GetMessage("SEARCH_LABEL")?><?=$APPLICATION->IncludeComponent(
    "bitrix:search.form",
    "flat",
    Array(
        "PAGE" => $arResult["FOLDER"]. $arResult["URL_TEMPLATES"]
["search"]
    )
)
    ?>
</pre>

```

Fig. 3.21 An example of the composite component template

It is obvious that you cannot edit the composite component template since a composite component is multi-page (consisting of subcomponents); instead, you can edit the templates of individual subcomponents. To edit any of the subcomponent templates, select **Edit subcomponent template** menu item (fig. 3.22).

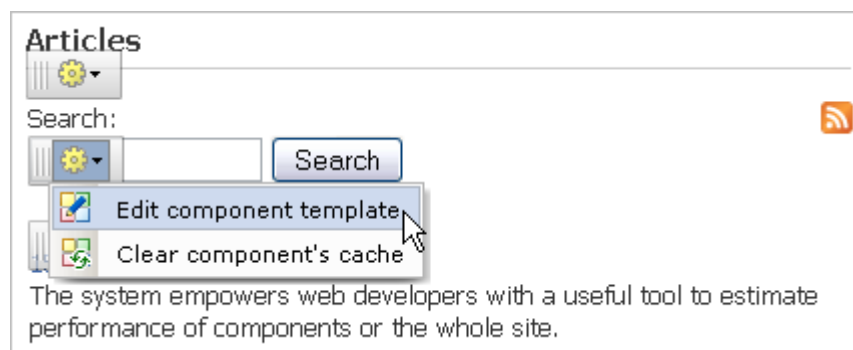


Fig. 3.22 Opening the subcomponent template for editing

When editing a template, the proper understanding of the way it functions is required. Consider the following sequence describing the chain of actions a template performs.



Suppose you want the breadcrumbs chain to show in a frame, each title in a separate cell. This can be done by creating a table with a pixel solid border as an outer HTML tag, in which the cells are created inside a PHP loop.

The best way to format the text is using CSS styles. However, if you still use HTML tags for formatting (e.g. **<b>** etc.), remember that your code must not produce overlapping tags.

### **Editing the Component Template CSS File**

The component appearance is defined by cascading style sheets in the **style.css** file. To edit the component template style sheet:

- Switch to **Design** mode.
- Click the component icon and select **Edit template CSS file** in the action menu (fig. 3.23).

**Note:** this command is available only if the component uses a custom template.



Fig. 3.23 Selecting the CSS file for editing

- Edit the file as required in the CSS file editor (fig. 3.24).

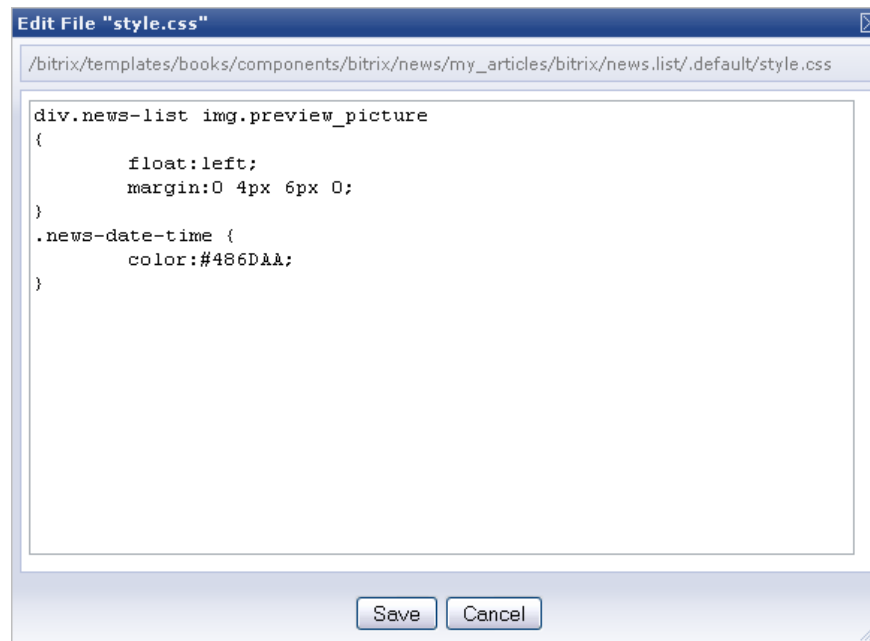


Fig. 3.24 The CSS file editor

The CSS file can contain any selectors according to the CSS specification.

## The Menu Component

A menu is an essential user interface part of any site. Let us consider the way the **Menu** component works in deeper detail.

Any Bitrix Site Manager powered site menu is based on the two entities:

- § a data array which can be edited in Control Panel (the distribution package includes sample menus);
- § a menu design template.

The data array defines the menu items, specifies their names and links. The array is stored in a file named like **.<menu\_type>.menu.php** in a respective site section folder. In **Site explorer**, this file has the title **Menu of type: "<menu\_type>"**.

### 1. Configuring a Template

**Menu**, as any other component 2.0, can have as many view templates as required. The menu template preparation begins with the selection of the site template areas in which you are planning to place the menus. Then, logically divide the projected menu in the "upper" and "lower" parts (which in fact means the template opening and closing code rather than the position), and the repeating elements (fig. 3.25). For a horizontal menu, these are table columns, while a vertical menu has items in rows.

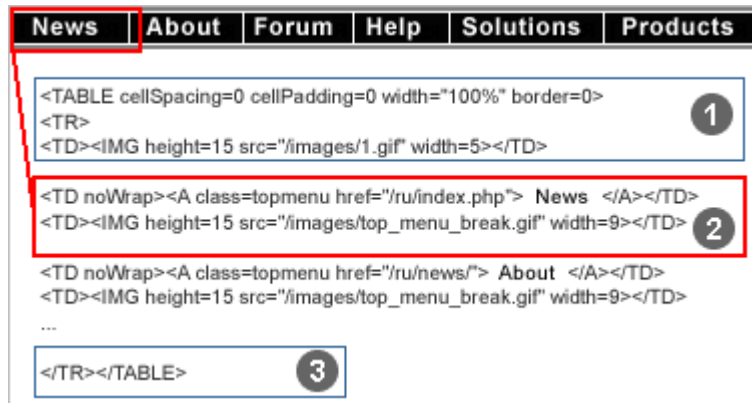


Fig. 3.25 An example of menu

To build a menu, the following actions are required:

- § decompose the menu to choose appropriate **HTML** elements;
- § create menu templates;
- § call the menu display function in a common template (in the **prologue** and the **epilogue**);
- § populate the menu according to the site structure.

For horizontal menus, the repeating elements (the menu items and separators) are table columns (fig. 3.25). For vertical menus, such elements are table rows.

Your menu template may require that you create additional selectors in the CSS file. For example, a text menu would need the menu item colors for the normal and hovered states (fig. 3.26). You may also want to somehow highlight the section titles (pop-up menu items) in the template. For example, these can use icons or bold font to indicate that they are not linked to a document, but open a new section (directory) etc.

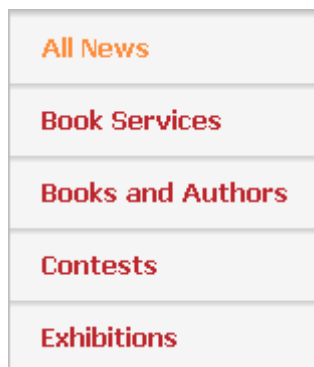


Fig. 3.26 Menu items

## 2. Menu Types

A site section can have multiple menu types: **top**, **left**, **bottom** etc. A simplest site might have only the **top menu**, which reflects the topmost navigational level. More

sophisticated sites usually include a **left** (or **right**) menu representing the secondary section level and shows links to pages and subsections of this section.

The menu types are configured in **Control panel: Settings > System settings > Module settings > Site Explorer**. Each site can have an individual set of menu types. For example, the demo version has the following two menu types: the **left menu** and the **top menu**:



Fig. 3.27 Menu type management

To keep information about a menu type, the system creates an individual file whose name uses the type title as a prefix. This prefix is also used in the name of a section menu description file. For example, **.left.menu.php** keeps items of the current section left menu, while **.top.menu.php** is for the top menu.

### 3. Menu Management

Each section menu can be controlled via Control Panel or the public section. In most cases, the public interface functions are more than enough for complete control over the menu.

However, in some cases you may want to edit menus in **Control panel**. The two menu edit modes exist: *simple* and *advanced*.

- q Open *Content > Site structure > ... > [section\_name]*.
- q Select **Edit menu** in the action menu of a required file (fig. 3.28).

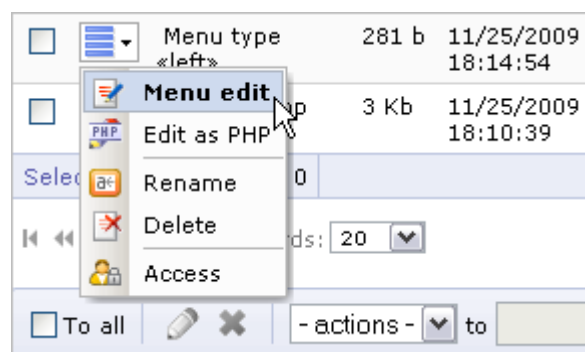
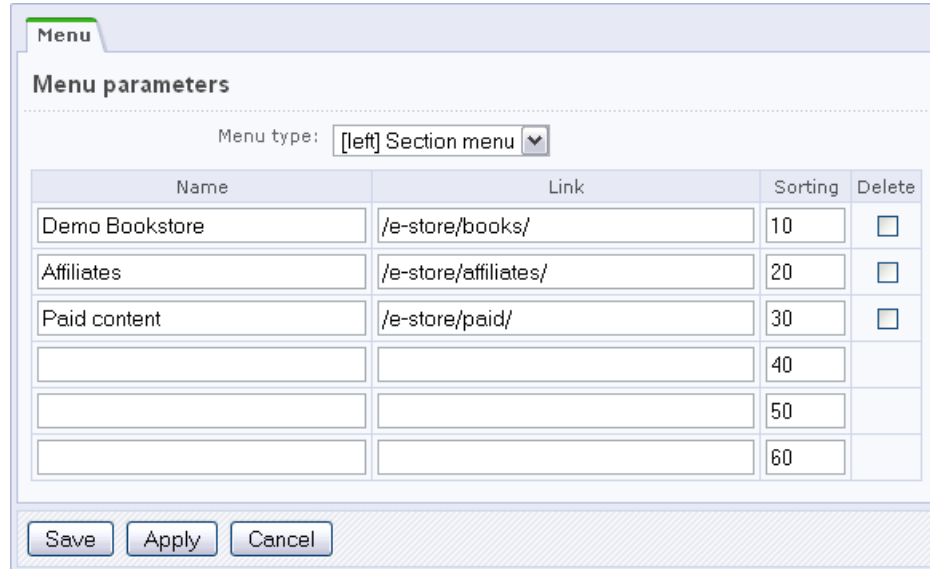


Fig. 3.28 Opening the menu for editing

- The simple mode form will bring up. Specify the menu item names, the menu item links and the sort indexes (fig. 3.29).



Name	Link	Sorting	Delete
Demo Bookstore	/e-store/books/	10	<input type="checkbox"/>
Affiliates	/e-store/affiliates/	20	<input type="checkbox"/>
Paid content	/e-store/paid/	30	<input type="checkbox"/>
		40	
		50	
		60	

Fig. 3.29 Simple menu editor

- To edit the menu in extended mode, click **Advanced mode** on the context toolbar. The menu editor form will transform as shown on fig. 3.30:

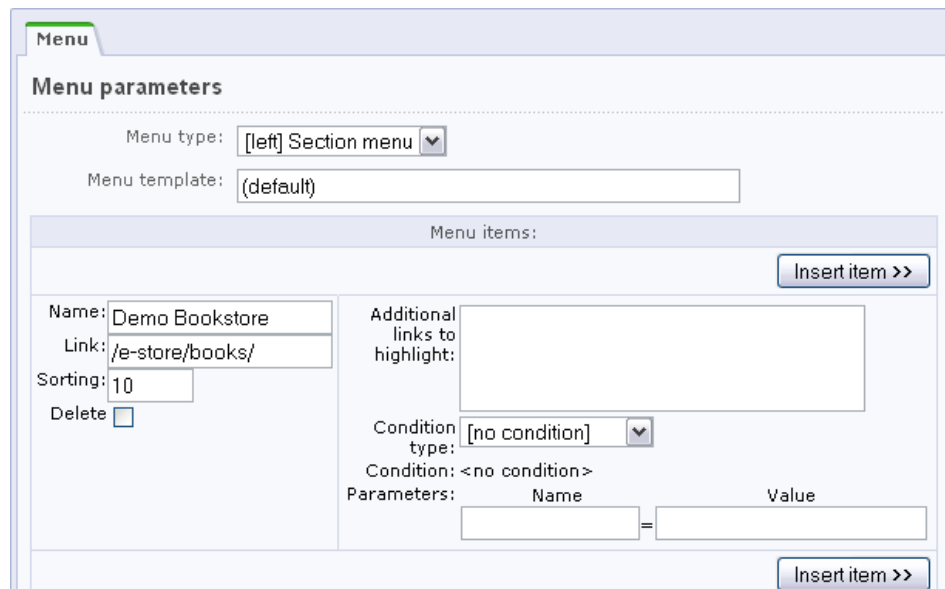


Fig. 3.30 Advanced menu editor

The following parameters can be configured in the advanced mode only.

- § Additional links that, being opened, will show the menu item as selected. Specify here paths and URL's, complete or incomplete. For example, if you

want the **Book Catalog** menu item to be highlighted for any page of the **Book catalogue** section, specify the path to the root folder of the book catalog: **/e-store/books/**. Alternatively, you can specify all the required pages.


- § Visibility conditions. For example, you can make the menu item visible to only users with certain access permissions.
- § Additional parameters: arbitrary data that the menu template can process and render. For example, to mark a menu item as a section title, specify the item parameter **SEPARATOR** and set the value to **Y**. When developing your menu template, you can verify if the parameter **SEPARATOR** equals **Y** and, if so, draw a separator under the menu item.

All the data that you add to the menu in Control Panel is saved in the appropriate site section folder in the **.left.menu.php** and **.top.menu.php** files containing the menu item description arrays. These files are saved in the root folder of a corresponding section. If no menu file exists in a section, the system searches the parent sections for the menu descriptions.

The system uses the following algorithm to show menus:

- § the menu template calls the menu rendering function;
- § the function checks if a menu descriptor file exists in a current section and, if so,
- § calls the menu generation template for this menu type;
- § emits the menu HTML code.

#### 4. Adding the Menu Component to a Page

The  **Menu** component (**bitrix:menu**) creates a menu. The component call has the following format:

```
// Att the top nested menu
<?APPLICATION->IncludeComponent(
    "bitrix:menu",
    "horizontal_multilevel",
    Array(
        "ROOT_MENU_TYPE" => "top",
        "MAX_LEVEL" => "3",
        "CHILD_MENU_TYPE" => "left",
        "USE_EXT" => "Y"
    )
);?>
```

The component is called in the site template exactly where you expect to see the menu. The component settings specify the template, menu type, nesting level etc.

Before you can edit the **Menu** component template, you first have to copy it to a site template. This procedure is discussed above (the [Copying a component template](#) and [Editing a component template](#) chapters).

Besides physical pages, you can set the menu to show entities of the logical level. For instance, the software setup includes a component to show the information

block sections in the left menu (**Menu items**, *bitrix:menu.sections*) and a usage example.

Since version 6.0, the menu template is a loop which iterates on the menu items creating them according to the component parameters.

## 5. Adding Information Block Sections to a Menu

The system allows to build menus by logical entities (for example, information block sections). The system setup includes an example of the left menu whose items refer to the physical level sections and the information block sections:

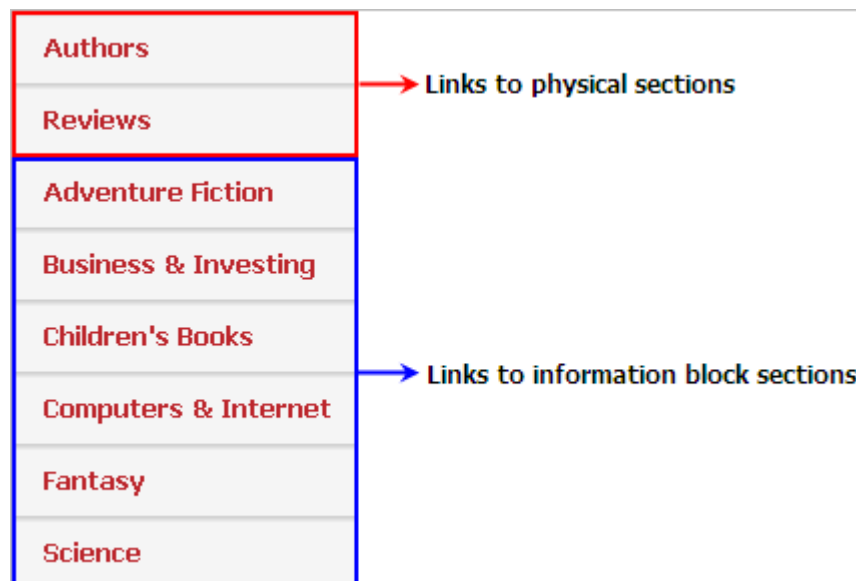



Fig. 3.31 The left menu

Only the first two items (**Authors** and **Reviews**) which have been created manually can be modified in the menu editor. The items referring to the information block sections will change automatically when the information block structure modifies.

The above described scenario requires that you check the **Use files .menu\_type.menu\_ext.php for menus** option in the **Menu** component settings. If the option is checked, the system verifies whether the *.menu\_type.menu\_ext.php* file exists in the current directory each time it builds the menu. If the file exists, it will be processed to build the menu items.

Let us consider an example of the described mechanism. Create the menu items **Authors** and **Reviews** manually. Then, do the following:

- Create an empty file *.left.menu\_ext.php* in the section whose menu would include information block items.
- Open the file for editing in the visual editor.
- Add the  **Menu items** component (*bitrix:menu.sections*) to the file.

- q Configure the component parameters.
- q Save the file.
- q Open the file for editing again, but this time in **Edit as PHP** mode.
- q Add the following code to verify the prologue inclusion:

```
<?
if(!defined("B_PROLOG_INCLUDED") || B_PROLOG_INCLUDED!==true)die();
global $APPLICATION;
$aMenuLinksExt = $APPLICATION->IncludeComponent(...
```

- q After the component call, type the menu inclusion code:

```
$aMenuLinks = array_merge($aMenuLinks, $aMenuLinksExt);
```

The file final code must be as follows:

```
<?
if(!defined("B_PROLOG_INCLUDED") || B_PROLOG_INCLUDED!==true)die();
global $APPLICATION;
$aMenuLinksExt = $APPLICATION->IncludeComponent(
    "bitrix:menu.sections",
    "",
    Array(
        "ID" => $_REQUEST["ID"],
        "IBLOCK_TYPE" => "books",
        "IBLOCK_ID" => "5",
        "SECTION_URL" => "/catalog/phone/section.php?",
        "DEPTH_LEVEL" => "1",
        "CACHE_TYPE" => "A",
        "CACHE_TIME" => "3600"
    )
);
$aMenuLinks = array_merge($aMenuLinks, $aMenuLinksExt);
?>
```

This example creates an array of information block sections ***\$aMenuLinksExt*** using the **Menu Items** component. Then, the following two arrays are merged: ***\$aMenuLinks*** (containing the standard left menu items) and ***\$aMenuLinksExt***. Further, when the **bitrix:menu** component creates a menu, the left menu is built by calling ***.left.menu\_ext.php***.

### The Include Area component

The include area templates are standard **HTML** pages containing the predefined formatting elements: tables, images, styles etc. Copy your new templates to the ***/page\_templates/*** folders:

- § ***/bitrix/templates/default/page\_templates/*** - for all templates, or
- § ***/bitrix/templates/<template identifier>/page\_templates/*** - for individual templates.

You can create include areas:

- § for a certain page;
- § for the whole section;
- § for the whole site to show, for example, copyright information, company name etc.

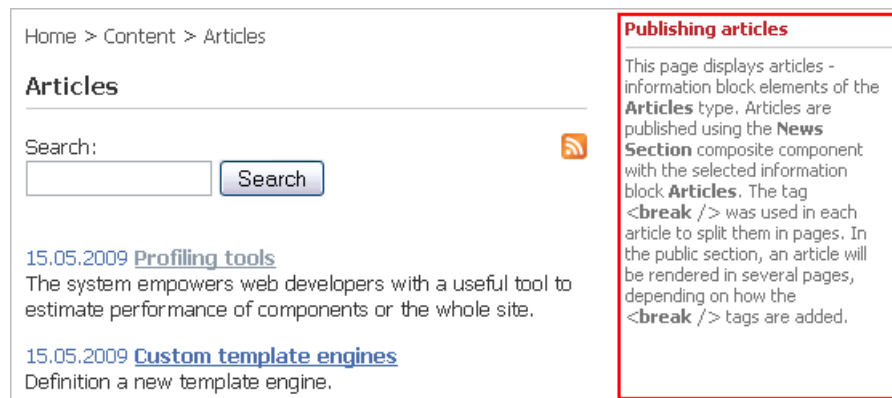



Fig. 3.32 An include area example

The include are individual files having `_inc` suffix by default. This suffix is added to the *page* name. For a site section, the include area file is `sect_inc.php` by default. Pages and sections can have as many include areas as required. If you create more than one include area for a single page or section, you should specify the file suffix manually in the component settings (fig. 3.33).

### Adding an Include Area

Perform the following actions to add an include area.

- Open the site template or the page in the visual editor.
- Add  **Include area** component to the page and set the component parameters (fig. 3.33).

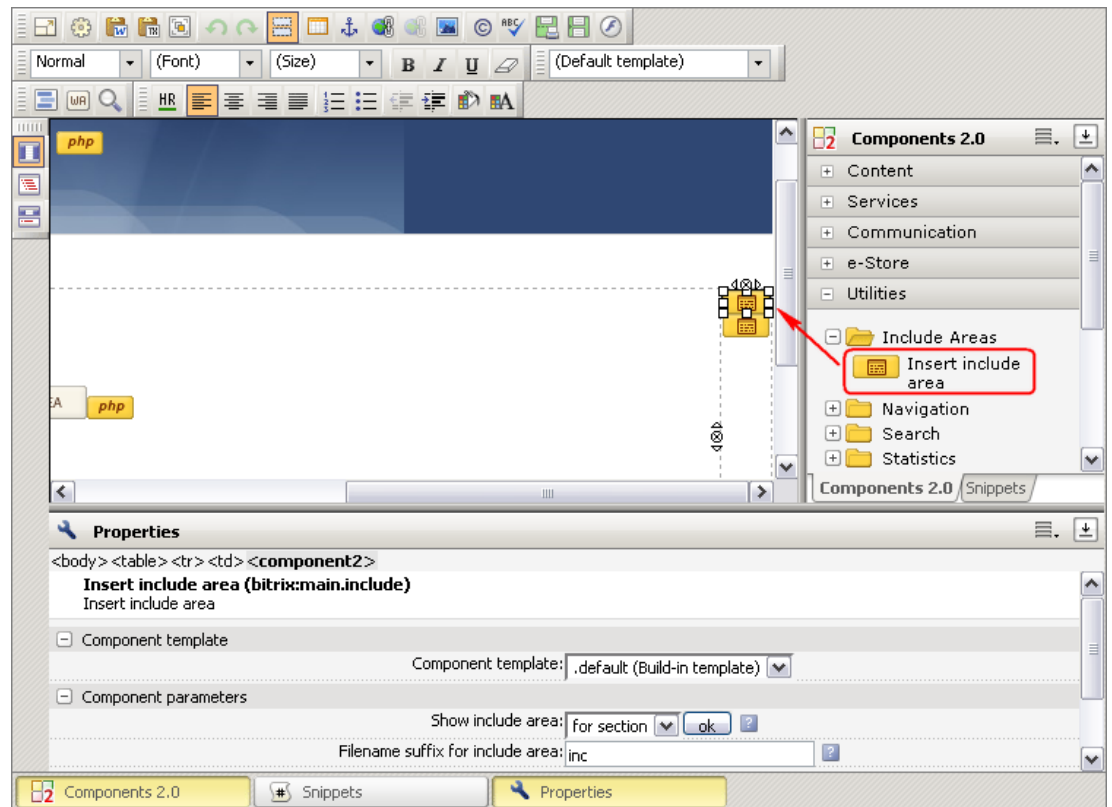


Fig. 3.33 Adding the Include Area component

The include areas of a currently open document can be created or edited directly from the site public section in **Content** or **Design** mode. The site elements that can have include areas will show the quick access toolbars (fig. 3.34).



Fig. 3.34 Working in the public section, adding an include area

Clicking the **Add include area to the current page** command will open the include area in the visual editor.

You can choose any view template for your include areas. The templates can be found in **.content.php** which along with the other page templates are located in **/bitrix/templates/.default/page\_templates/**.

All templates listed in **.content.php** will be available when creating a new page in the visual **HTML** editor in the new page menu.

## The Component Settings

The **Include area** component has the following parameters (fig. 3.35):

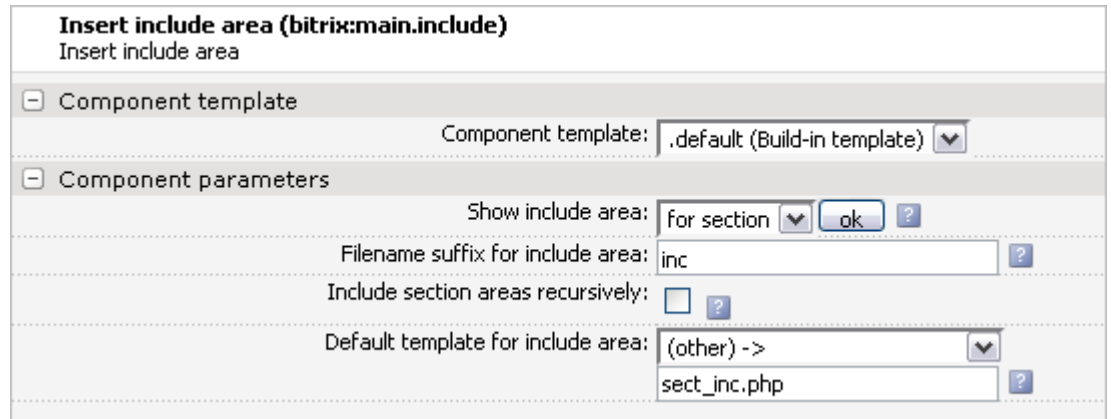


Fig. 3.35 The include area components

The **Show include area** parameter specifies an object for which an include area is destined: a page, a section or the whole site

**Note:** other settings depend on the selection in the first parameter.

**File name suffix for include area** specifies the text that will be appended to the name of the file that will contain the include area.

The parameter is visible for **page** or **section** include areas. The default value is **inc**, in which case the file name is **<page\_name>\_inc.php** for a page, and **sect\_inc.php** for a section.

The **Include section areas recursively** option is available for section include areas only. A section shows its include area if it exists. If this option is checked, it specifies that, if a section has no include area, the component will search all the parent sections and show the first found include area.

**Path to include area file** – available for include areas that use the contents of an external file to render itself. Such include areas are site-wide.

The **Default template for include area** field specifies the page template that will be used by this include area. The parameter is always available.

## Using Language Dependent Message Files

If you intend your template to support different languages, you will obviously need to localize any textual information you add to the template: titles, image ALT texts, button captions etc.

Structure your template by making the **HTML** code common for all languages and replacing all the text elements with the text extraction PHP calls. The language messages are stored in a separate files, for example: ***/bitrix/templates/<template\_identifier>/lang/ru/*** for the Russian language and ***...../lang/en/*** for the English language, respectively. The message files for other languages are created similarly.

The message file has the same name as the PHP file for which it is intended. For example, the language messages for the copyright include area (the ***/bitrix/templates/<template\_identifier>/include\_areas/copyright.php*** file) are stored in:

- § ***/bitrix/templates/<template\_identifier>/lang/ru/copyright.php*** - for the Russian interface;
- § ***/bitrix/templates/<template\_identifier>/lang/en/copyright.php*** - for the English interface.

The following code is an example of the language message file for the English language:

```
<?
$MESS ['SEARCH_TITLE'] = "Search";
?>
```

At the beginning of the template file, add the message file inclusion instruction:

```
// /bitrix/templates/<template_identifier>/lang/ru/copyright.php
<?
IncludeTemplateLangFile(__FILE__);
?>
```

Now you can call the text message extraction function anywhere in the template file:

```
<font class="copy"><?echo GetMessage("COPY");?></font>
```



- § ***/bitrix/php\_interface/dbconn.php*** (included at the very beginning);
- § or ***/bitrix/php\_interface/after\_connect.php*** (this one is included after the database has been established);
- § or ***/bitrix/php\_interface/<site\_code>/init.php*** (after the site has been resolved).

It is not necessary to specify the encoding explicitly. It will be set automatically to **windows-1251** for the Russian language or **iso-8859-1** for other languages.

# Appendix 1. Recommendations on the HTML Template Preparation

---

While designing your site, do not forget to visually mark out the line of demarcation between the prologue (**header.php**) and the epilogue (**footer.php**).

Select the design elements that can be further added to the style sheet: fonts, colors etc.

When developing a multilevel menu, it would be wise to pick out repeating elements to simplify the menu template creation and further control over these menus.

To simplify the support of different language mirrors, tend to use text instead of graphics.

When cutting the design images and preparing the HTML template, you have to provide for placeholders of the page surface areas: menus, banners, forms.

It is recommended to prepare the template using tables or layers.

If you create an image rich design, notice the monochromatic areas. Replace them with empty table cells of the corresponding color when preparing the template HTML layout.

## Appendix 2. Adding Buttons to the Control Panel Toolbar

---

You can add buttons to the control panel toolbar.

Let us consider an example: add the **Edit Lesson** button, which is to be shown when viewing an online course. Create a file

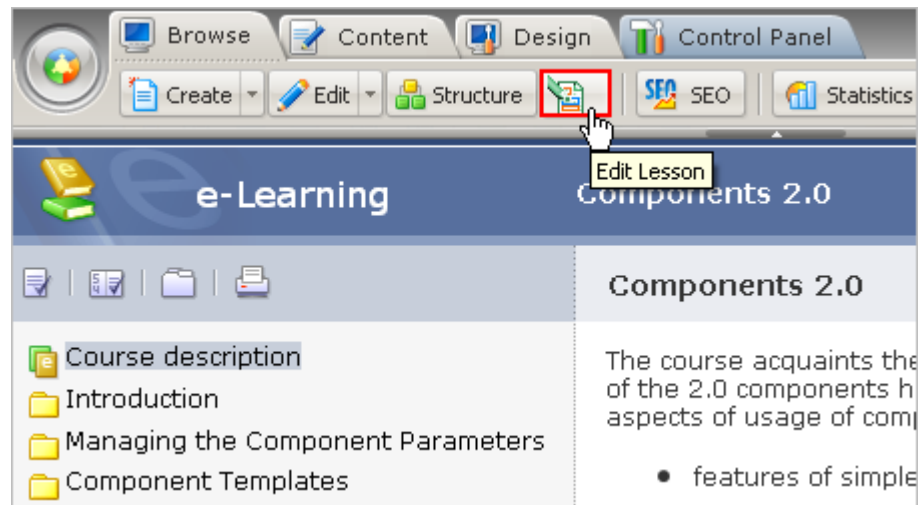
*/bitrix/php\_interface/include/add\_top\_panel.php* and add the following code to it:

```
<?
// check the page URL and the user permissions
if (($APPLICATION->GetCurPage() ==
    "/communication/learning/course/index.php") &&
    $USER->IsAdmin())
{
    $main_sort = 100; // the button group sort weight
    $alt = "Edit Lesson"; // the tooltip text
    $link = "Y"; // the button is the link
    // the link to the page
    $href =
        "/bitrix/admin/learn_lesson_edit.php?lang=".LANGUAGE_ID."&ID=" .
        $_GET["LESSON_ID"]."&COURSE_ID=" . $_GET["COURSE_ID"].
        "&CHAPTER_ID=" . $_GET["CHAPTER_ID"];
    // the button image
    $src = "/bitrix/images/fileman/panel/edit_lesson.gif";

    // add the button
    $APPLICATION->AddPanelButton(array("LINK"=>$link, "HREF"=>$href,
        "SRC"=>$src, "ALT"=>$alt,
        "MAIN_SORT"=>$main_sort,
        "SORT"=>200));
}
?>
```

The Control Panel pages call **\$APPLICATION->ShowPanel()** in the page prologue to check if this file exists and shows the buttons.

Eventually, this code adds the button which will only be visible to administrators and on the course pages (fig. 4.2).



*Fig. 4.2 A custom button on the toolbar*

When clicked, this button switches to Control Panel and opens the lesson being viewed for editing.

For the **Information Blocks** module, the buttons can also be added using **CBlock::ShowPanel()**. The number of buttons for the information block elements page and the element view page can be different. The information block components have a special parameter which specifies to add buttons to the control panel toolbar.

## Appendix 3. Customizing the Design of Helper Elements

---

### Customizing the Error Message Format

Sometimes you may need to bring the visual representation of the system error messages into line with the site design.

The appearance of the database connection error messages is defined in */bitrix/php\_interface/dbconn\_error.php*. Edit this file to get the look you need.

To change the appearance of the database request error messages, edit the file */bitrix/php\_interface/dbquery\_error.php*.

### Customizing The Site Maintenance Message

To change the message your visitors see when the site is undergoing maintenance works, copy the file */bitrix/modules/main/include/site\_closed.php* to */bitrix/php\_interface/<language>/* or */bitrix/php\_interface/include/*. Edit the file as required.

### Configuring the Breadcrumbs Layout

Pagewise display of continuous information can be easily coded by calling **NavPrint()**. This function prints links for breadcrumb style navigation. The following parameters can be used to configure the breadcrumbs:

**NavPrint(\$title, \$show\_allways=false, \$StyleText="text", \$template\_path)**

Here:

- § **\$title** – titles of elements to be shown;
- § **\$show\_allways** – if **false**, the function will not show navigation links if all entries fit one page. If **true**, the links are always shown.
- § **\$StyleText** – the CSS class to render the navigation links.
- § **\$template\_path** – the path to the navigation link template.