

# Building an eZ publish site

Created 23/12/2003

The eZ publish documentation is a community project and is available under the GNU Free Documentation License. If you edit the documentation your contribution will be released under the terms of this license.

# Table of contents

Building an eZ publish site . . . . .	1
Prerequisites . . . . .	1
Installation . . . . .	1
Case . . . . .	3
Creating a new site . . . . .	4
Testing the administration interface . . . . .	6
Setting up the main layout . . . . .	8
Creating a bare-bone main template . . . . .	9
Creating and using a custom style sheet . . . . .	11
Customizing the main layout . . . . .	13
Creating sections . . . . .	16
The welcome page . . . . .	17
Adding the contents of the welcome page . . . . .	17
Setting the default page . . . . .	18
Creating and using a custom template . . . . .	19
The news page . . . . .	21
Adding news articles . . . . .	23
Assigning the News folder to the News section . . . . .	25
Overview of the latest news . . . . .	25
Full display of an article . . . . .	29
News archive . . . . .	32
The members page . . . . .	35
Creating a custom content class . . . . .	36
Adding members . . . . .	38
List of members . . . . .	44
Member info page . . . . .	46
The guestbook . . . . .	48
Creating the content class . . . . .	48
Adding content . . . . .	49
Creating the template . . . . .	50
Adding an action button . . . . .	52
Making the button work . . . . .	54
The input template . . . . .	56
Testing the guestbook . . . . .	59
Implementing an approval mechanism . . . . .	61
Modifying the entry page . . . . .	61
Creating a workflow . . . . .	61
Connecting the workflow to a trigger function . . . . .	62
Approving entries . . . . .	62
The "runcronjob" script . . . . .	63

The links page . . . . .	63
Adding content . . . . .	64
Creating the template . . . . .	65
Displaying the sub-folders . . . . .	67
Displaying the contents of a folder . . . . .	68
Creating a tree-style appearance . . . . .	69
Same content at different locations . . . . .	71

# Building an eZ publish site

This chapter is basically a step-by-step tutorial that explains how to build an eZ publish site from scratch. The tutorial is written for eZ publish 3.2. It is targeted at people who have little or no previous experience with eZ publish, but have some generic knowledge about web development. The tutorial demonstrates how eZ publish can be used to build a dynamic community-website. It is based on the "learning-by-example" technique that many people prefer when it comes to understanding and learning new technologies.

Throughout this chapter, a lot of new topics will emerge. Because of the flexible and complex nature of eZ publish, it is impossible to understand and learn everything at once. In other words, the reader should pay some, but definitely not too much attention to details. Advanced and complex functionality is explained in later chapters. For now, the reader should simply follow and accept the examples and solutions that are presented in the tutorial.

## Prerequisites

To follow this tutorial, the reader should:

- have access to and know how to use a computer system
- be able to install eZ publish
- have read and understood the ["eZ publish basics"](#) chapter
- be able to navigate and manage files on a filesystem
- be able to edit text files using a text editor
- have basic knowledge of HTML
- have some knowledge of CSS

## Installation

Before beginning, you need to have eZ publish installed on a computer system. It should be installed using the ["Normal installation"](#) method with the following settings:

Primary language: English  
Additional languages: none  
Site template(s): plain  
Site access configuration: URL

In other words, you need to download, unpack and configure eZ publish using the setup wizard. Choose the English language as the primary language. Don't choose any additional languages, we won't need any at this point. Make sure you install only the "Plain" site and that you choose the default site access configuration. Don't tamper with the default title, URL, etc. You will learn how to change these manually later on. Security issues will be discussed in a different

chapter, we won't bother with those at this point. Make a note of the URLs that can be used to reach the administration and user interface of the plain site that was configured. These URLs are presented at the final step of the installation. For a detailed explanation of the installation steps, please refer to "[The setup wizard](#)" section within the "[Installation](#)" chapter.

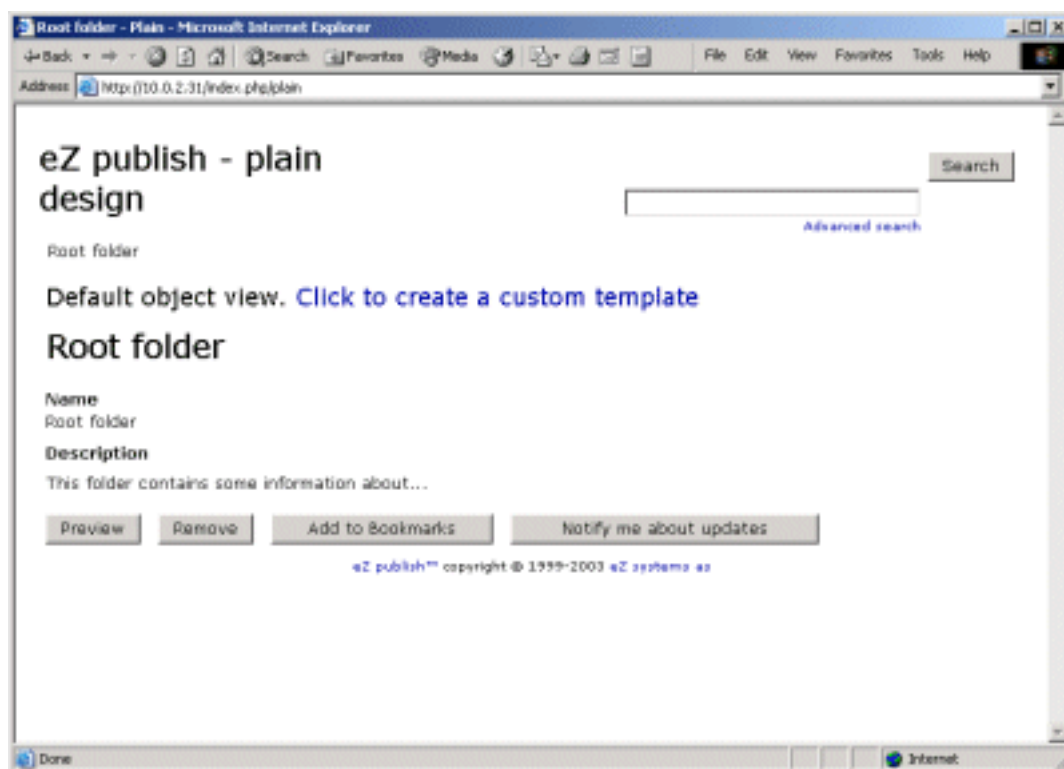
Assuming that a plain site was installed correctly, you should be able to access the site's user and administration interface. The URLs used to access the interfaces should look something like this:

*<http://www.example.com/path/index.php/plain>*

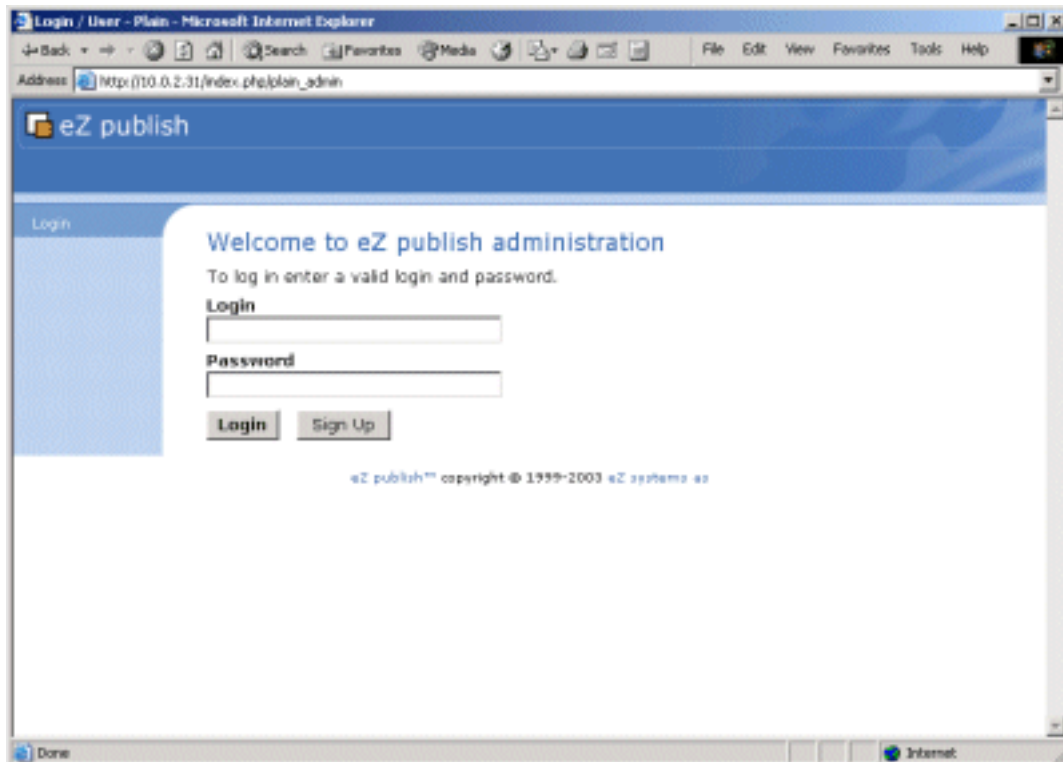
and

*[http://www.example.com/path/index.php/plain\\_admin](http://www.example.com/path/index.php/plain_admin)*

The first URL will take you to the default public/user interface of the "Plain" site that was installed. It should look something like this:



The second URL will take you to the login section of the site's administration interface, which should look something like this:



## Case

Lets assume that you're approached by members of a local chess club called "The Scandinavian Check Mates", TSCM for short. This is basically a small community that consists of approximately 40 chess fanatics. The club has had a homepage for several years now, which was made by a former TSCM member. Unfortunately, this member has moved to the other side of the planet and has no intention of maintaining the TSCM webpages. Because of the increasing popularity of the internet and the club itself, the members have decided that something should be done with the site, which currently looks a bit "outdated".

The club members have no experience with web development. However, they all know how to use a computer. Lately, they've been surfing around the internet, and have come up with a temporary/preliminary list of requirements for a new club-site:

It should look slick & fancy, but not too bloated.

There should be a page with news bulletins.

A list of members must also be included.

It would be nice to have a guestbook of some sort.

It should be easy to maintain/update and extend the site.

What we have here is a typical situation where an organization needs a new website. The club members have been inspired by looking at various sites on the internet and have a fairly good idea of what they want. However, keep in mind that this is only a preliminary list of requirements. An experienced developer should already have a gut feeling, telling that the club certainly will come up with new requests along the way. In other words, we should be flexible throughout the entire development process. Fortunately, flexibility and dynamic behavior are some of the key features of eZ publish. Within a couple of hours, we shall deliver a powerful, dynamic, modern

and expandable web solution that could be easily changed to suit any kind of small/mid-sized community.

## Creating a new site

We'll create a new site by reusing some of the elements and default files that were generated when the "Plain" site was installed. This is achieved simply by renaming and changing the contents of the "Plain" site's configuration files.

### 1. Renaming the siteaccess directories

As already mentioned, the plain site has two interfaces, a user interface and an administration interface. Inside eZ publish, these are called "plain" and "plain\_admin".

The "settings/siteaccess/" directory contains subdirectories for sites/interfaces that eZ publish may be configured to run. Navigate into the directory and rename "plain" to "tscm" and "plain\_admin" to "tscm\_admin".

Make sure that the user running the web server has write access to the "settings/siteaccess/tscm" directory and the files that reside inside it.

### 2. Modifying the siteaccess files

eZ publish comes with a lot of configuration files, all residing in the "settings" directory. The ".ini" files are the original/default eZ publish configuration files. These should never be changed. Instead, you should always use the override system, which is basically a set of files containing settings that will override the default configuration. In addition to the global override configuration files, each site can have its own set of override files. eZ publish reads the configuration files in the following order:

Default configuration files  
(/settings/\*)

Site specific override files  
(/settings/siteaccess/[sitename]/\*)

Global override files  
(/settings/override/\*)

Configuration settings in (1) will be overridden by settings in (2), which again will be overridden by settings in (3). Some of the override files end with an ".append" and some end with the ".append.php" extension. The latter is because of security issues related to sites running in non virtual host mode.

Edit the "settings/siteaccess/tscm/site.ini.append" file.

Modify the "SiteName=Plain" setting by replacing "Plain" with "The Scandinavian Check Mates".

Modify the "SiteDesign=plain" setting to "SiteDesign=tscm". This tells the system to use the "tscm" design when the user/public interface of the TSCM site is accessed.

Modify the "VarDir" setting to equal "tscm". This tells the system to use the "var/tscm" directory instead of "var/plain".

Add the following lines at the end of the file:

```
[TemplateSettings]
TemplateCache=disabled
Debug=disabled
```

```
[ContentSettings]
ViewCaching=disabled
```

```
[DebugSettings]
DebugOutput=enabled
```

When building a site, it is always a good idea to turn off the cache engines and enable debug output. When the "DebugOutput" is set to "Enabled", eZ publish will print a lot of debug related information right below the actual page that it has rendered. Use this information to solve problems. Debug output may be turned off at any time by setting "DebugOutput=" to "disabled". The screenshots within this tutorial will not contain any debug output.

Edit the "settings/siteaccess/tscm\_admin/site.ini.append" file.

Modify the "SiteName=Plain" setting by replacing "Plain" with "The Scandinavian Check Mates".

The "SiteDesign" setting in this file tells the system to use the default admin interface, which is exactly what we want.

Modify the "VarDir" setting to equal "tscm". This tells the system to use the "var/tscm" directory instead of "var/plain".

### **3. Modifying "settings/override/site.ini.append.php"**

Edit the "settings/override/site.ini.append.php" file and replace the occurrences of "plain" and "plain\_admin" with "tscm" and "tscm\_admin". You'll probably have to do this six times.

### **4. Removing the plain var directory**

Under "var/", each site has a directory that is used for storage of site specific content files (usually images and binary files), logs and cache files. If the "var/plain" directory exists, delete it.

We've already told eZ publish to use the "var/tscm" directory when the TSCM site is accessed. The directory will be automatically created by eZ publish.

### **5. Creating a design directory**

In the root directory of eZ publish, there is a subdirectory called "design". This directory contains all the design related files that may be used by various sites. Navigate into the design directory and create a subdirectory called "tscm". We will not (re)use any of the files that reside in the "plain" directory; feel free to remove it.

Under the "design/tscm" directory, create the following subdirectories:



fonts	(for fonts, used when rendering bitmaps of text)
images	(non-content specific images; banners, logos, etc.)
override	(for custom overrides)
stylesheets	(for CSS files)
templates	(for site-specific templates)

In addition, create a subdirectory called "templates" inside the "override" directory. Make sure that the user running the web server has write access to the this directory.

## 6. Testing

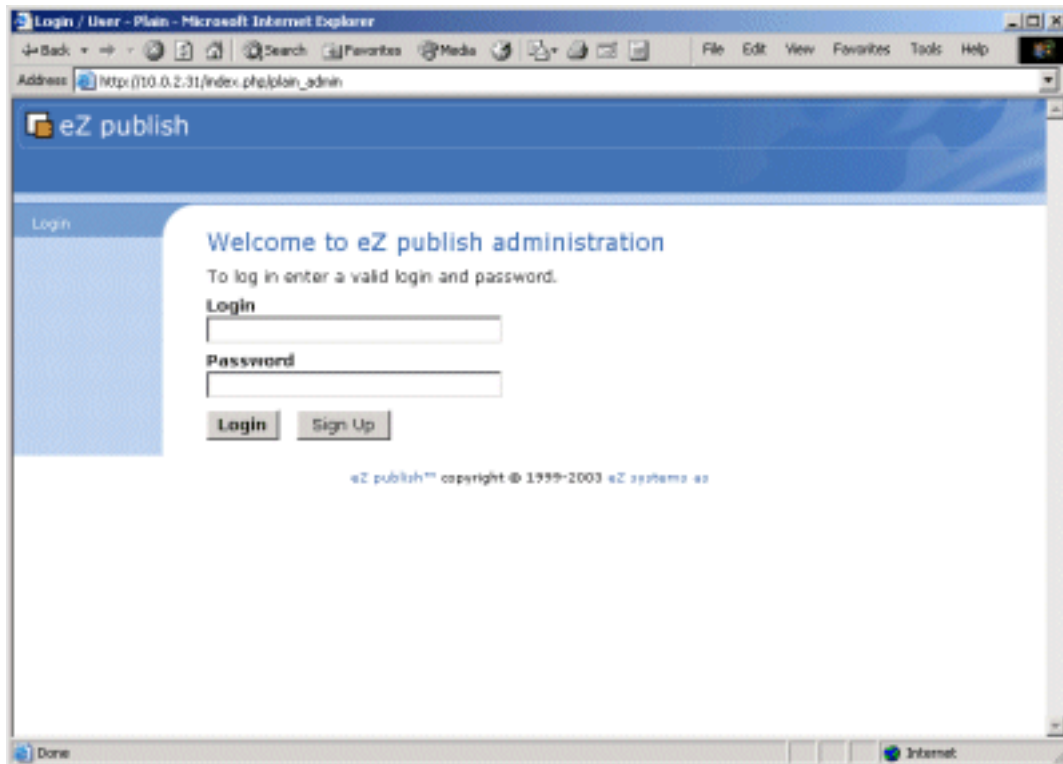
You should now be able to reach the site by replacing the "plain" and "plain\_admin" part of the URL with "tscm" and "tscm\_admin".

Don't worry about the way we're accessing the site at this point (the [...]/index.php/tscm URL is ugly and temporary), we'll discuss how to change the access method later on.

When browsing "tscm", the site pops up without problems. But wait! How is this possible? - you might ask; thinking that the design directory we just created is completely empty - and we've told the system to look for design in that directory. The explanation is that if eZ publish is unable to find something, it always falls back to the standard/default settings, designs, etc. What you're looking at, is the output from one of the standard templates.

# Testing the administration interface

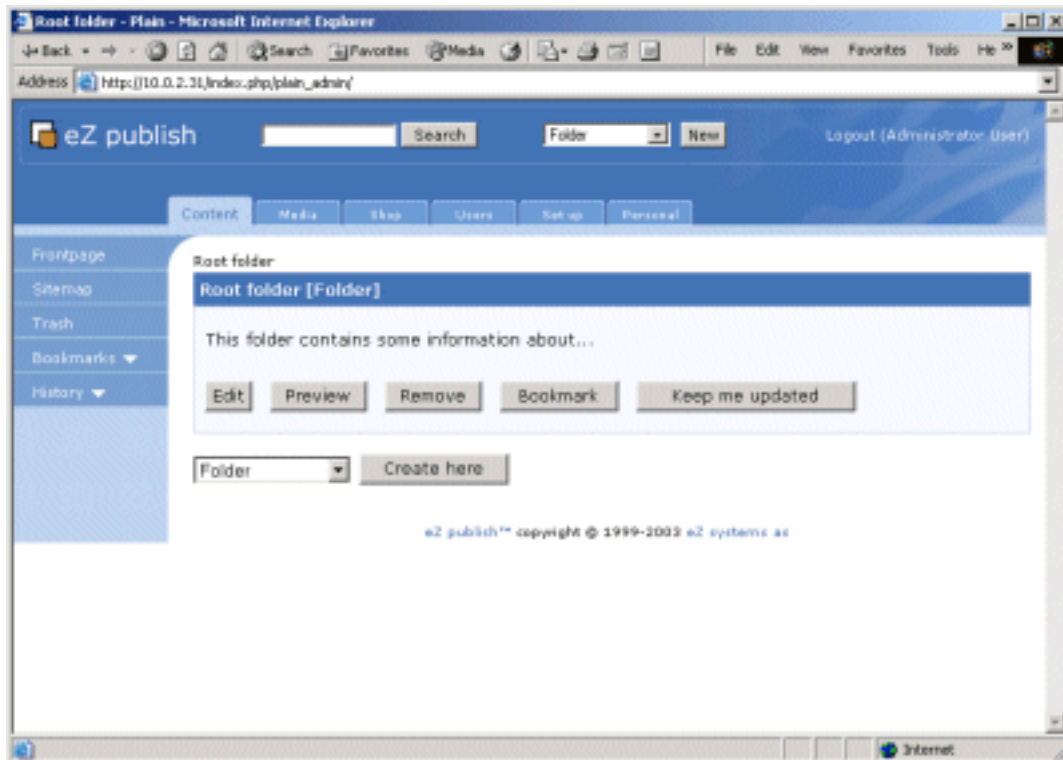
Type in the URL that points to the administration interface of the TSCM site. You should see a page that looks like this:



Log into the administration interface with the default eZ publish administrator username/password combination:

username: admin  
password: publish

After a successful login, your browser should display a page that looks something like this:



This is the standard administration page. What you're looking at is the root folder of the TSCM site. Since we haven't created any content yet, the root folder is completely empty.

The administration interface will not be discussed in detail. However, throughout this tutorial you'll learn how to use some of its most important features. The point of this exercise was to bring up the interface and log in. To minimize hassle, simply stay logged in (you'll need to use the administration interface frequently). For a comprehensive walkthrough of the interface itself, please refer to "[The Administration Interface](#)" chapter (unfortunately, this chapter hasn't been published yet, it will be in the near future).

### Clearing the caches

It is a good idea to clear the caches at this point. Just to make sure and to avoid unwanted surprises. The following text explains how to do this:

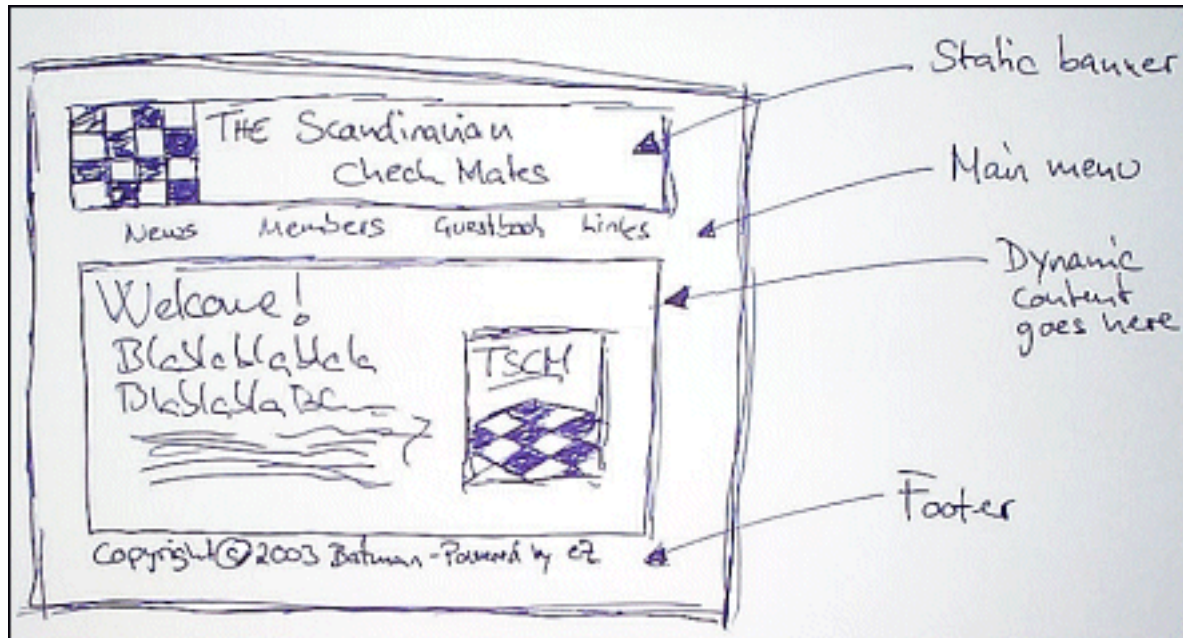
Make sure you're logged into the administration interface.  
Click the "Setup" tab.  
Click on "Cache" (from within the menu on the left hand side).  
Click "All Caches".

### IMPORTANT!

You need to do this every time you create and make use of a new template (or else, you'll not be able to see the changes that you've made). The next release of eZ publish (3.3) will have a configuration setting for disabling the template override cache. When using 3.2, you'll have to clear the cache manually.

# Setting up the main layout

Lets keep things at a simple level. We'll put up a nice banner and a static menu on the top of the page, reserve some space for dynamic content in the middle, and include a small footer at the bottom. The following raw sketch illustrates how we wish to structure the layout of the site:



In other words, the TSCM site will consist of 5 parts:

- Welcome page
- News section
- Member list
- Guestbook
- Links

## Creating a bare-bone main template

eZ publish uses templates as the fundamental unit of site design. A template is basically an extended HTML file that describes how some particular type of content should be visualized. The system will automatically look for a file called "pagelayout.tpl", which is the main template file. If it isn't found, eZ publish will fallback to the default/standard "pagelayout.tpl" file (which resides under design/standard/templates/...).

Lets put some content into our own "pagelayout.tpl" file. At the minimum, it should contain the following lines:

```
{*?template charset=latin1?*}  
  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <link rel="stylesheet" type="text/css"
href="{ "stylesheets/core.css" |ezdesign} />
    <link rel="stylesheet" type="text/css"
href="{ "stylesheets/debug.css" |ezdesign} />
    { * Include the standard "page_head.tpl" file. * }
    {include uri="design:page_head.tpl" }
  </head>
  <body>
    Hello world!
  </body>
</html>

```

Clearly, this is simply a standard XHTML file with some additional eZ publish code. All eZ publish specific template code is inside the "{" and "}" brackets.

Copy & paste the lines from above into a text editor and save the file as "design/tscm/templates/pagelayout.tpl". Try to surf the site ([...]/index.php/tscm). The browser should present you with a page containing the "Hello world!" string.

(IMPORTANT: Remember to clear the cache from within the administration page, or else, you will probably not see any changes!).

Take a look at the document source (usually right click on the page and choose "View Document Source") to view the output that eZ publish has generated so far. If everything works properly, you should be able to see a bunch of meta and link-rel tags.

The following text contains a comprehensive explanation of the eZ publish specific code that we've put into our custom main template file. Feel free to skip to the next section (the following text is all about detail).

### Template charset revelation

```
{*?template charset=latin1?*}
```

Reveals the charset of the template file to the template engine. If not specified, the engine will fallback to the default template charset, which is specified in the "site.ini" configuration file. It is possible to use special characters, for example UTF8 characters in a template. However, to be able to use the templates with an arbitrary charset, they should be written in plain ASCII.

### Special inclusion of style sheets

```
{ "stylesheets/core.css" |ezdesign }
{ "stylesheets/debug.css" |ezdesign }
```

What happens here is that the text strings within the quotes are being piped into an operator called "ezdesign". This operator prepends the current site-design directory to the inputted text. The result is a valid file location. The first line will return "/design/tscm/stylesheets/core.css", the second line returns "/design/tscm/stylesheets/debug.css". If the file doesn't exist, the operator will simply

fallback to the standard design (/design/standard/stylesheets/...). Instead of specifying a hardcoded location, we use the "ezdesign" operator. This will allow us to change the design of the site in an easy way without having to modify the template file itself. The "core.css" file is used by some of the standard templates; the "debug.css" file is used to format the debug output.

### Including the standard "page\_head.tpl" file

```
{* Include the "page_head.tpl" file. *}  
{include uri="design:page_head.tpl"}
```

The first line is simply a template comment (comments are placed inside "{\*" and "\*}"). The second line tells the template engine to include the "page\_head.tpl" file. The contents of the included file will be inserted at the exact position where the include function is called. Since we haven't got our custom "page\_head.tpl", the system will use the standard "page\_head.tpl" file (which is located here: "design/standard/templates/page\_head.tpl"). This is an important template file that takes care of setting some eZ publish specific variables, sets the HTML page title, generates meta information and so on. It should always be included.

## Creating and using a custom style sheet

So far we have only set up some basic things without generating any remarkable visual output. Lets add some custom CSS code and tell the main template to use it. The following parts of the tutorial will heavily depend on the CSS code presented in this section, so make sure you do this part right.

The CSS code will tell the browser to use a custom background image, called "background.png". This image file is available from the following link: [background.png](#). Grab and save the image as "design/tscm/images/background.png".

### Creating a custom CSS file

Copy & paste the following lines (from the box below this text) and save the file as "design/tscm/stylesheets/tscm.css".

```
body  
{  
    background-color: #ffffff;  
    background-image: URL( "/design/tscm/images/background.png" );  
    background-repeat: repeat;  
}  
  
p  
{  
    font-size: 80%;  
}  
  
td
```

```

{
  font-size: 100%;
}

hr
{
  height: 0px;
}

a
{
  color: #444444;
}

a:visited
{
  color: #666666;
}

a:hover
{
  color: #aaaaaa;
}

.main
{
  width: 732px;
  margin-left: auto;
  margin-right: auto;
}

.border
{
  color: #ffffff;
  background-color: #333333;
}

.menu
{
  width: 80%;
  background-color: #474747;
  margin-left: auto;
  margin-right: auto;
  text-align: center;
  font-size: 80%;
  padding: 2px;
}

.menu a
{
  color: #cccccc;
}

.menu a:visited
{
  color: #bbbbbb;
}

.menu a:hover
{
  color: #eeeeee;
}

```

```

.content
{
    padding: 32px;
    text-align: left;
}

.footer
{
    color: #bbbbbb;
    background-color: #333333;
    padding: 4px;
    text-align: center;
    font-size: 70%;
}

.footer a
{
    color: #eeeeee;
}

.footer a:visited
{
    color: #eeeeee;
}

.footer a:hover
{
    color: #ef8e00;
}

.pagetitle
{
    font-size: 170%;
    padding-bottom: 8px;
}

```

### Using the custom CSS file

Edit the main template file ("design/tscm/templates/pagelayout.tpl"). In the head section, add the following code after the already existing stylesheet-related lines:

```

<link rel="stylesheet" type="text/css"
href="{ "stylesheets/tscm.css" | ezdesign} />

```

This line will tell eZ publish to make use of the custom CSS file that we just created.

### Testing

Reload/refresh the TSCM page in the browser. You should be able to see that the background has changed from plain white to grayish vertical stripes.

## Customizing the main layout



It is time to create the main layout for the site. We'll set up a simple layout using a table. The table will be split up into the following sections:

Chess piece	Banner	Chess piece
Chess piece	Main menu	Chess piece
Border	Dynamic content	Border
Footer		

The following list contains links to the banner and the layout-images that will be used to create the main appearance of the site.

Banner image: [banner.png](#)

Upper left section of chess piece: [piece\\_upper\\_left.png](#)

Upper right section of chess piece: [piece\\_upper\\_right.png](#)

Lower left section of chess piece: [piece\\_lower\\_left.png](#)

Lower right section of chess piece: [piece\\_lower\\_right.png](#)

Grab and save these images inside the "design/tscm/images/" directory.

### Setting up the main layout

Edit the main template file ("design/tscm/templates/pagelayout.tpl"). Copy & paste the lines from the box below into the "body" section within the template file itself (simply replace the "Hello world!" string with the following code).

```
<!-- Main table -->
<table cellspacing="0" cellpadding="0" class="main">

  <!-- Chess piece + banner + chess piece -->
  <tr>
    <td></td>
    <td><a href="{"/}|ezurl}></td>
  </tr>

  <!-- Chess piece + main menu + chess piece -->
  <tr>

    <!-- Left chess piece -->
    <td></td>

    <!-- Main menu -->
    <td class="border">

      <!-- Main menu table -->
      <table cellspacing="0" cellpadding="0" class="menu">
        <tr>
```

```

        <td><a href="{ "/" | ezurl }>News</a></td>
        <td><a href="{ "/" | ezurl }>Members</a></td>
        <td><a href="{ "/" | ezurl }>Guestbook</a></td>
        <td><a href="{ "/" | ezurl }>Links</a></td>
    </tr>

    <!-- End of main menu table -->
</table>

</td>

<!-- Right chess piece -->
<td></td>

</tr>

<!-- Left border + dynamic content + right border -->
<tr>
    <td class="border"></td>
    <td class="content">{$module_result.content}</td>
    <td class="border"></td>
</tr>

<!-- Footer -->
<tr>
    <td colspan="3" class="footer">
        {include uri="design:footer.tpl"}
    </td>
</tr>

<!-- End of main table -->
</table>

```

### The footer

Notice that the main template includes a file called "footer.tpl". In other words, the actual content of the footer will be put into this file instead of the main template file. This is simply done in order to demonstrate and practice the split-up and inclusion of template files. Create an empty text file and copy the following line into it (everything on one line):

```

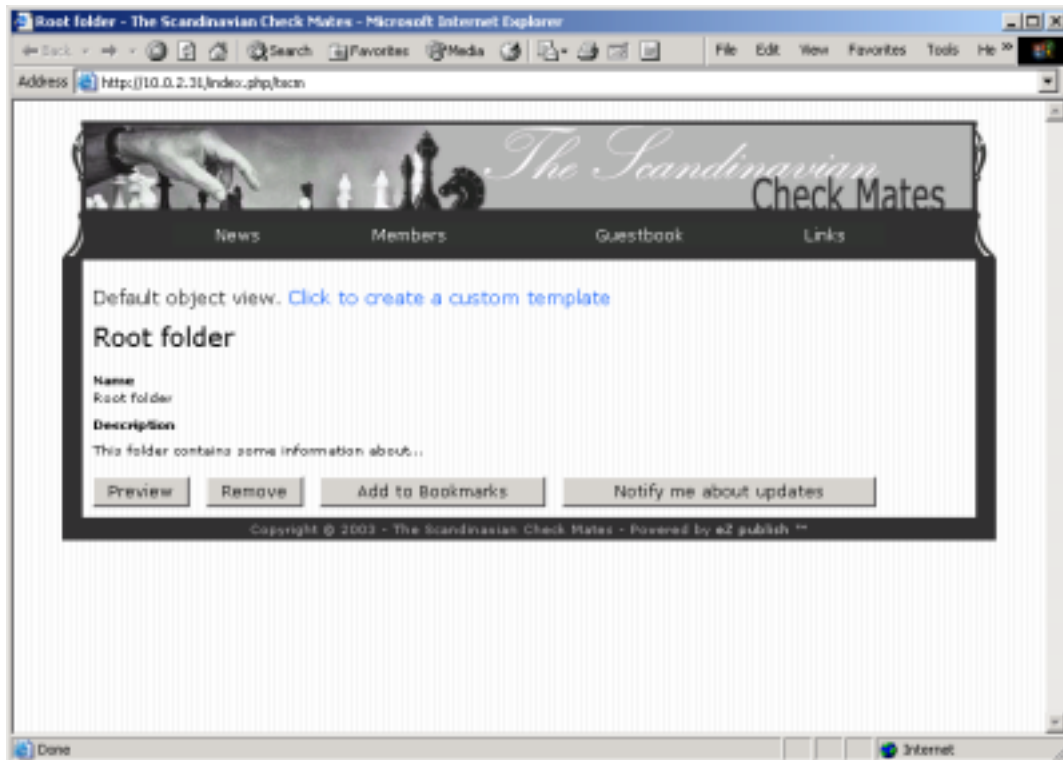
Copyright &copy; 2003 - The Scandinavian Check Mates -
Powered by <a href="http://ez.no">eZ publish</a> &trade;

```

Save the file as "design/tscm/templates/footer.tpl".

### Testing

Reload/refresh the TSCM page again. (IMPORTANT: Remember to clear the cache from within the administration page, or else, you will probably not see any changes!). You should see something that resembles the following screenshot:



## Creating sections

In order to be able to separate and manage various parts of the site, we have to use something called *sections*. Sections are used to segment the site in a logical way. They are very handy when it comes to the creation of custom templates (override of standard templates in different ways) and access control. For more information about sections, please read the [Sections](#) section within the "[eZ publish basics](#)" chapter.

The TSCM site can be divided into four dynamic sections:

- News
- Members
- Guestbook
- Links

This separation will make our work a lot simpler when it comes to creating custom templates that will override the standard templates. The following list of steps explains how to create the sections:

- Make sure you're logged into the administration interface.
- Select/click the "Setup" tab.
- Click on "Sections" (from the menu on the left hand side).
  - The interface will display a list of the current sections.

- Click "New".
- In the name field, type in "News section".
- Set the "Navigation Part" dropdown box to "Content".

Click "Store".

Repeat steps 5 through 7 for the remaining sections using the following section names:

"Member section"  
"Guestbook section"  
"Link section"

That's it.

## The welcome page

This section explains how to

make use of the CMS to store a welcome page  
configure eZ publish to show the welcome page by default  
set up a custom layout/design for the welcome page

## Adding the contents of the welcome page

eZ publish comes with a handful of ready-to-use content classes. Instead of hardcoding the contents of the welcome page into a template file, we will use one of these classes for storing it.

The article class is more or less suitable for this purpose.

Make sure you're logged into the administration interface.

From the dropdown box in the middle of the screen, select "Article".

Click the "Create here" button.

The edit article page will appear.

Put some text into the *title*, *intro* and the *body* fields.

Feel free to use the example text presented here:

**Title:**

*Welcome to the TSCM website!*

**Intro:**

*The Scandinavian Check Mates (TSCM) is a popular chess club located in Skien, the sub-capitol of Norway. The check mates have been supporting chess-activities in Skien continuously since the founding of TSCM in 1921.*

**Body:**

*We are a non-profit organization, incorporated with the state of Norway in 1963, that is dedicated to the advancement of the game. As the recognized state affiliate of our national organization, the Norwegian Chess Federation, we are responsible for sponsoring and conducting all local championship tournaments for both youths and adults. Our extensive scholastic program includes holding the state team championship event for primary, elementary, middle school, and high school teams. At present, we have nine championship tournaments available for adults, with all but the Senior Championship and Women's Championship open to everyone. We frequently lend assistance and give publicity to local organizers who run chess tournaments and promote chess*

*events.*

Upload a thumbnail image by clicking the "Browse" button.  
Feel free to use the following image file: [chess\\_kings.png](#).

Click the "Send for publishing" button. We're done.  
The system will bring you back to the root folder.

## Setting the default page

The default configuration (for the TSCM site) instructs eZ publish to display the contents of the root node. However, this is not what we want. The default display should be the contents of the welcome article that we just created. In order to do this, we must:

Find the identification number of the node that points to the content object which contains the actual welcome article.

Make a site-specific configuration change that will instruct eZ publish to display the desired node instead of the root node.

### **Finding the ID of the node we want to display**

In the administration interface, hover the mouse pointer over the article that you just created (titled "Welcome to the TSCM website!"). Look in the status bar of the browser, it should read something like:

"http://[...]/tscm\_admin/content/view/full/44". The end of the URL specifies the actual ID of the node. Your system may display a different number than 44. Make a note of the node's identification number.

### **Changing the default node**

Bring up the "settings/siteaccess/tscm/site.ini.append" file in an editor. Modify the "IndexPage" setting so that it points to the desired node ID. If the desired node ID is 44, then the line should read "IndexPage=/content/view/full/44".

### **Testing**

If you browse the site now, you'll see something like this:



eZ publish will display the desired node using the standard template for the "Article" content class. The following section describes how to create and use a custom template.

### Bug-note

If the image is missing: try to edit the article (by clicking on it from within the administration interface) and upload the thumbnail image again. You may have to do this two or three times. The image will eventually appear after a couple of uploads. This happens only the first time you upload an image. This is an annoying bug in eZ publish 3.2 (will be fixed in the 3.3 release).

## Creating and using a custom template

Since we're unsatisfied with the way our welcome page is displayed, we'll create a custom template for it. This is easily solved by instructing eZ publish to use a desired template each time a specific node is accessed. The following guide should take you through the necessary steps. Bring up the administration interface.

Click on the "Setup" tab.  
Click on "Templates" (in the menu on the left hand side).  
A list of template files will appear.

Locate a file called "/node/view/full" and click on it.  
On the next page, simply click "Create New".  
Type in "full\_view\_welcome" into the template name field.  
Make sure the "Section" and "Class" dropdown boxes are set to "Any".  
Type in the identification number of the welcome-page node.  
Make sure that "Empty" is chosen in the "Base template on" section.  
Click create.

eZ publish will then generate a custom template that will be used instead of the standard template every time the specified node is accessed. The newly generated template file will be put in the "design/tscm/override/templates/" directory. Edit the newly generated file. You can either use the administration interface for this (by clicking on the edit icon for the desired template file) or your favourite text editor. Put the following lines into the file:

```
<div class="pagetitle">
  {attribute_view_gui attribute=$node.object.data_map.title}
</div>
<div class="imageright">
  {attribute_view_gui attribute=$node.object.data_map.thumbnail}
</div>
{attribute_view_gui attribute=$node.object.data_map.intro}
{attribute_view_gui attribute=$node.object.data_map.body}
```

This template code will simply extract the contents of the welcome object and display it in a nice way. Clear the caches and try to browse the TSCM site again. By now, it should look something like this:



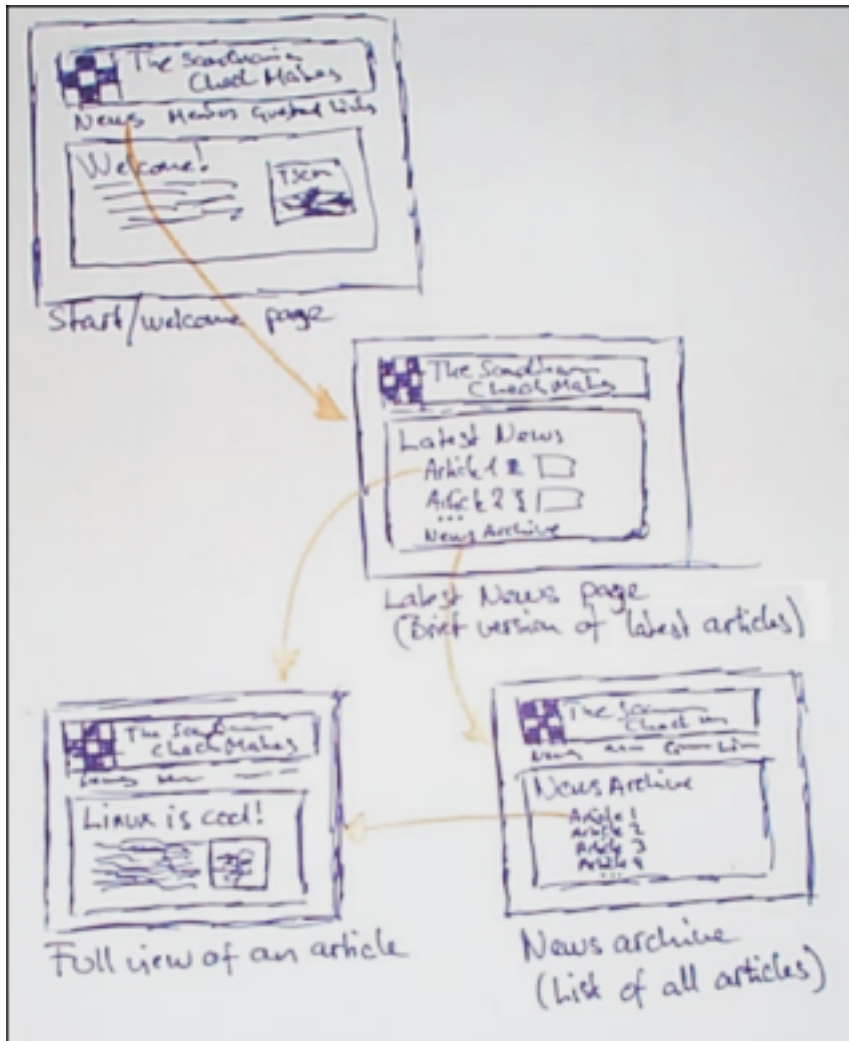
## The news page

The news page will be the first "real" dynamic page that we create. Actually it will consist of not one, but three dynamic pages:

- Overview of the seven latest news articles
- Full display of a specific article
- News archive (list of all news articles)

The following sketch illustrates this:





As you can see from the illustration, the news overview will be accessible from within the main menu of the site. The full display of an article will be shown when the user clicks the title, the thumbnail image or the "Read more" part of a specific article (also by clicking on an article title from within the archive).

The archive will be accessible from the bottom of the news overview. It will simply present a list of all the articles that have been published. The articles in the overview and the archive list will be sorted by their publishing date (recent articles will appear at the top of the list).

### Adding more CSS

In order to format the news-related pages in a nice way, we need to add a bit more CSS. Simply copy and append the following CSS code to the contents of the "tscm.css" file.

```
.latest_news
{
    width: 75%;
    margin-left: auto;
    margin-right: auto;
}
```

```

.headline_link a
{
    color: #000000;
    font-family: Times New Roman, Times;
    font-size: 140%;
}

.headline_link a:visited
{
    color: #000000;
    font-family: Times New Roman, Times;
    font-size: 140%;
}

.headline_link a:hover
{
    color: #888888;
    font-family: Times New Roman, Times;
    font-size: 140%;
    text-decoration: none;
}

.headline
{
    color: #000000;
    font-size: 200%;
    font-family: Times New Roman, Times;
    padding-bottom: 24px;
}

.news_archive
{
    width: 85%;
    font-size: 80%;
    margin-left: auto;
    margin-right: auto;
}

.center
{
    font-size: 80%;
    text-align: center;
}

```

## Adding news articles

One way to start is by creating and structuring the actual news-content. The administration interface of eZ publish makes this an easy match. For the news section, we'll use two of the built-in content classes, *Folder* and *Article*. We'll create a folder dedicated for news. Later on, all news articles will be placed inside this folder. The following list of steps explains how to create the folder.

Bring up the administration interface.

Make sure you're in the root folder.

(If not: click the "Content" tab).

From the dropdown box approximately in the middle of the screen, select "Folder".  
Click the "Create here" button.  
Fill out the *name* and the *description* fields.

**Name:**

*News*

**Description:**

*This folder contains all the news articles.*

Click "Send for publishing".  
The system will bring you back to the root folder.

What we've done so far is that we've created a folder. A folder is simply a named container into which we put things that belong there. Obviously, we'll put all our news articles into the newly created folder. Having a folder without content is kinda silly, so lets create a couple of news articles. The following list of steps explains how to create an article within the newly created News folder.

Bring up the administration interface.  
Make sure you're in the root folder.  
(If not: click the "Content" tab).

Click on the name of the folder that we just created.  
The system will take you to it (and show the empty contents of the folder).

From the dropdown box, select "Article"  
Click the "Create here" button.  
The system will bring up the "Edit Article" page.

Fill in the *Name*, *Intro* and *Body* fields. For example, you could add the following content:

**Title:**

*Batman beats Joker at local tournament*

**Intro:**

*Batman was playing against the evil Joker. The game ended in a non-violent way. However, rumor has it that NCF is considering sanctions against both players in this game for crimes against chess.*

**Body:**

*After beating his main competition in round 2, The Joker seemed to have things wrapped up. Indeed, Batman quickly won two pawns in his round 3 game against The Joker (1412). Then Batman seemed to enter a twilight zone of the mind in which he forgot how to play chess. After a series of incredible blunders Batman escaped from a lost middlegame into a totally lost endgame, down a piece and several pawns in a simple position where the pawns could not be stopped. Then the Joker, 'dizzy from success' as the Russians say, allowed Batman to trap his bishop and force a repetition of the position; but the fat lady hadn't sung yet. Batman, dissatisfied with swindling a draw in a game which most players would long since have resigned, wanted more. So he declined to force the repetition, and promptly walked into a one-move knight fork of his own king and rook. Rumor has it that NCF is considering sanctions against both players in this game for crimes against chess.*

Add a thumbnail image.  
Feel free to use this image: [article\\_image\\_01.png](#)

Click the "Send for publishing" button.  
The system will bring you back to the "News" folder.  
You should be able to see the title of article that you've just created.

Having only one news article is boring, so lets create another one.  
Simply repeat the first steps of the previous step-by-step list (create a news article in the news folder).

Feel free to use the following content:

**Title:**

*New TSCM site*

**Intro:**

*The board of TSCM has decided that it is time to change the original web page, which hasn't been changed since 1998. The new site will be powered by a system called eZ publish and will hopefully be a bit more flexible than the static page that we've had before.*

**Body:**

*Since John is gone, and because of the increasing popularity of the internet and our club, TSCM has decided to hire a freelance developer called Jane Doe to build a new TSCM site. The site will be running on the top of a sophisticated system called "eZ publish" (created by a handful of crazy people at <http://www.ez.no>) and will hopefully become the part of our new image. The Scandinavian Check Mates would like to thank John Doe (the previous webmaster) for his efforts. Rumors say that John moved to Brazil a couple of months ago. He will be offline for some years while researching the capitalist's primitive slaughter of the rain forests.*

Add a thumbnail image.

Feel free to use this image: [article\\_image\\_02.png](#)

Click the "Send for publishing" button.

The system will bring you back to the News folder.

You should be able to see the title of the articles that were added.

## Assigning the News folder to the News section

The "News" folder is currently assigned to the default eZ publish section, which is section one. However, we wish to gain a bit more control over the content that is inside this folder. This means that we'll have to assign the folder (and its contents) to a custom section. We've already created a couple of custom sections. The following text explains how to assign the "News" folder (and its contents) to the "News section".

Bring up the administration interface.

Click on the "Setup" tab.

Click on "Sections" (from within the menu on the left hand side).

Locate the line that says "News section".

Click on the small "binder" icon (assign) which is within the "News section" line.

The "Choose section assignment" page will appear.

Set the radio button to the "News" folder.

Click the "Select" button.

eZ publish will then assign the "News" folder and all the articles in it (also future articles that are placed there) to the "News section".

# Overview of the latest news

The main menu (just below the site-banner) contains a link called "News". When this link is accessed, eZ publish should display the seven latest news articles. We'll achieve this functionality by creating a custom version of the "Folder" content class' full-view template. The full-view template will list the children of the node that is being accessed. Each child will be shown using a custom line-view template.

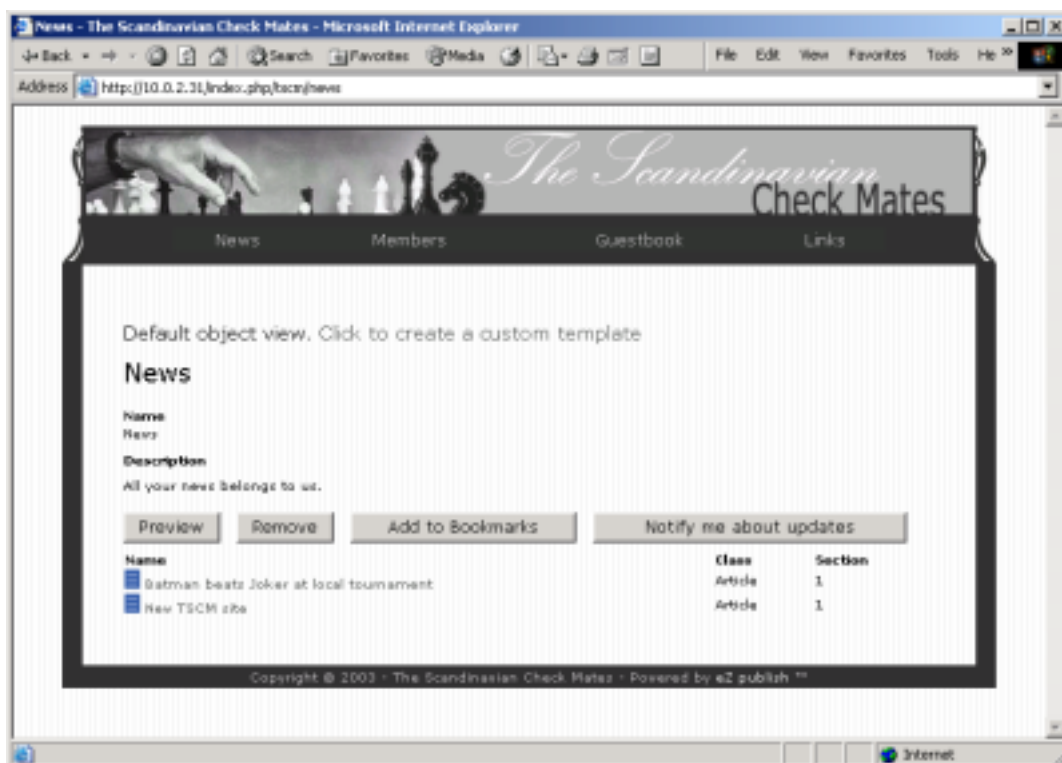
## Fixing the menu

First, we have to make sure that the link in the menu works. The following piece of code takes care of this:

```
<a href="{ "/news" | ezurl }>News</a>
```

Insert this code into the "pagelayout.tpl" file (replace the previous News-link). Note that we're not telling eZ publish the ID of the node that contains the "News" folder. We're using the URL alias of the "News" folder, which is "news". The URL alias is piped into the "ezurl" operator. The operator takes care of setting the correct path based on the siteaccess settings.

Browse the TSCM site. Click on the "News" link. eZ publish will then display the contents of the "News" folder using the standard full-view folder template. A standard line-view template will be used to display each child of the folder. The browser should display something that resembles the following screenshot:



## Creating custom templates

The following text explains how to generate a custom full-view-template for the

"Folder" content class within the "News" section.

Bring up the administration interface.

Click on the "Setup" tab.

Click on "Templates" (in the menu on the left hand side).

A list of template files will appear.

Locate "/node/view/full.tpl" and click on it.

On the next page, simply click "Create New".

Type in "full\_view\_news\_folder" into the "Name" field.

Set the "Class" dropdown box to "Folder".

Set the "Selection" dropdown box to "News section".

Do not enter anything into the node field (leave it blank).

In the "Base template on" section, select "Empty file".

Click "Create".

This will generate an empty file

("design/tscm/override/templates/full\_view\_news\_folder.tpl") and instruct eZ publish to use it every time a folder in the "News section" is viewed. Put the following lines into the file:

```
<h1>Latest news</h1>

{* Grab some of the content of the node that is being viewed. *}
{let children=fetch( content, list, hash( parent_node_id, $node.node_id,
                                         sort_by, $node.sort_array,
                                         limit, 7,
                                         class_filter_type, include,
                                         class_filter_array, array(
'articles' ) ) ) }}

{* LOOP: For each child of the node... *}
{section name=Child loop=$children}

    {* Display the content of the child using a line-view template. *}
    {node_view_gui view=line content_node=$Child:item}

    <hr>

{* End of loop. *}
{/section}

{* End of namespace. *}
{/let}
```

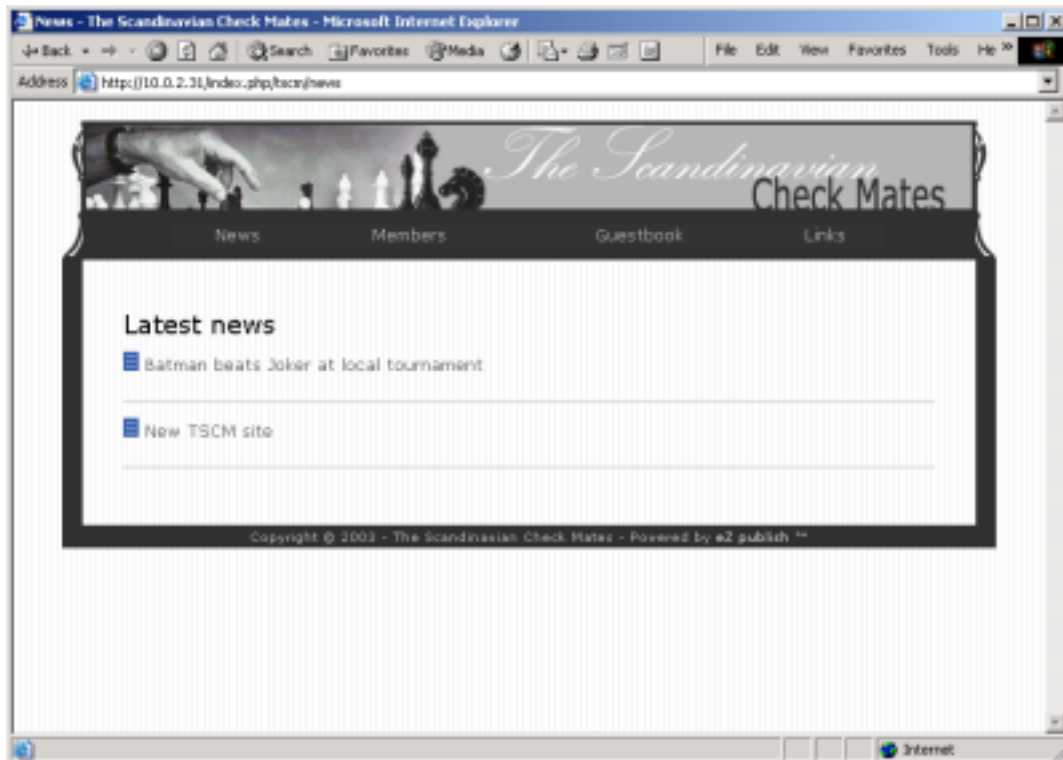
This piece of template code does the following:

Displays the name of the node that is being viewed.

Grabs the seven most recent articles from the node that is being viewed.

Displays the grabbed articles using a line-view template.

Try to access the "News" page again (either by refreshing the page or by clicking on the "News" link on the main menu). You should be presented with something like this:



The newly generated full-view template will automatically make use of the standard line-view template. The following text explains how to create a custom template for the line-view of the "Article" class within the "News section".

Bring up the administration interface.

Click on the "Setup" tab.

Click on "Templates" (in the menu on the left hand side).

A list of template files will appear.

Locate "/node/view/line.tpl" and click on it.

On the next page, simply click "Create New".

Type in "line\_view\_news\_article" into the "Name" field.

Set the "Class" dropdown box to "Article".

Set the "Selection" dropdown box to "News section".

Do not enter anything into the node field (leave it blank).

In the "Base template on" section, select "Empty file".

Click "Create".

This will generate the "design/tscm/override/templates/line\_view\_news\_article.tpl" template file. eZ publish will use this template every time it is instructed to display a list-view of article objects within the "News section". Put the following lines into the file:

```
<table class="latest_news">
  <tr>
    <td colspan="2">
      <div class="headline_link">
        <a href={$node.url_alias|ezurl}>
          {attribute_view_gui
attribute={$node.object.data_map.title}
          </a>
        </div>
```

```

        </td>
    </tr>
    <tr>
        <td>
            {attribute_view_gui attribute=$node.object.data_map.intro}
        </td>
        <td valign="top">
            <a href={$node.url_alias|ezurl}>
                {attribute_view_gui
attribute=$node.object.data_map.thumbnail image_class="small"}
            </a>
        </td>
    </tr>
</table>

```

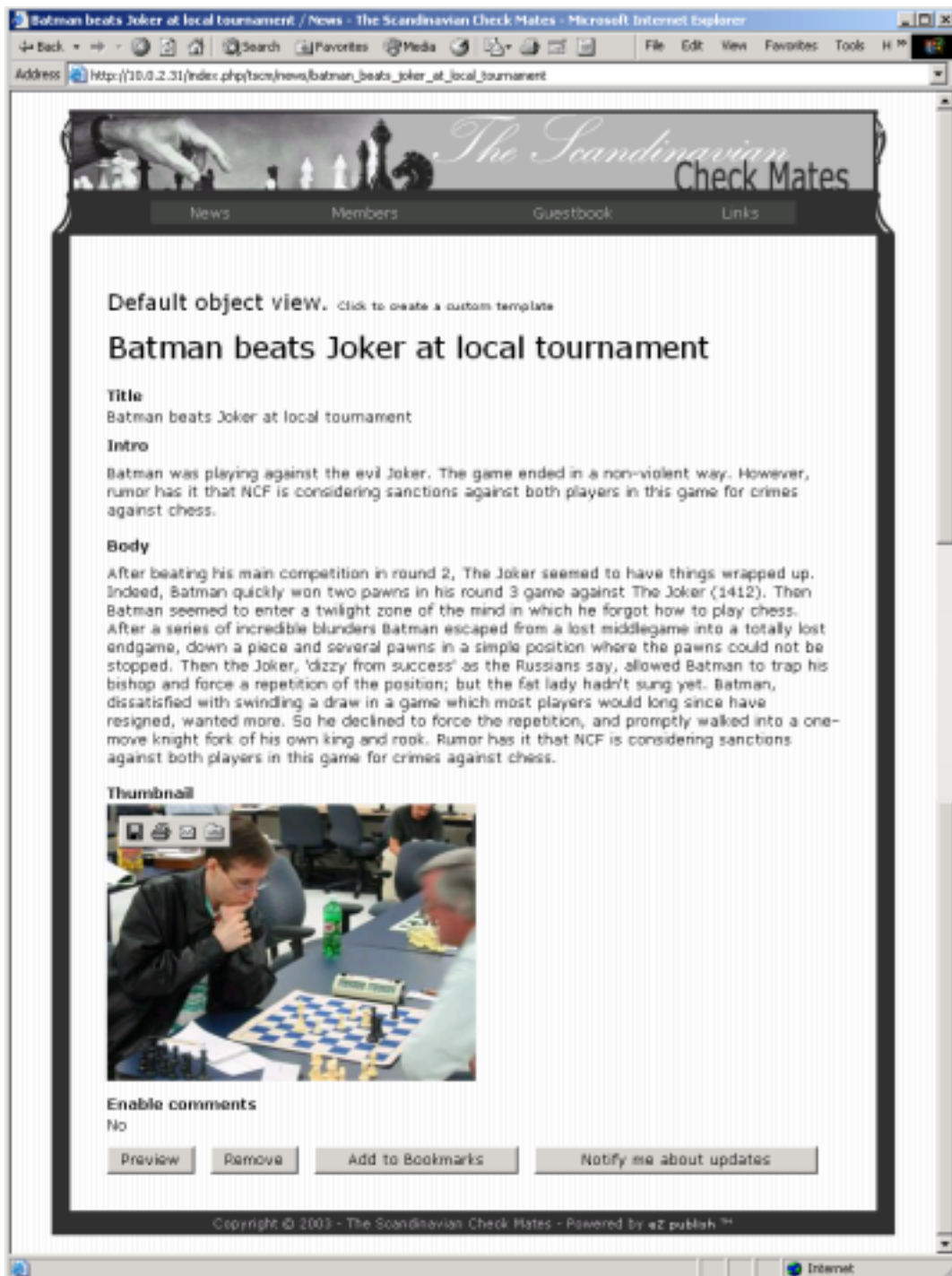
This code snippet takes care of getting and displaying some the contents of a news article. For each article, the headline, the intro and the article's thumbnail image is displayed. Browse the "News" page again. You should be looking at a page that resembles the following screenshot:



## Full display of an article



The news overview that we created in the previous section displays a list of the seven most recent articles that are stored inside the news folder. However, the page only shows a small fragment of the articles. The entire contents of an article can be accessed by clicking its headline or image. These are links to the full view of the article. When accessed, you should be able to see something like this:



What you're looking at is the standard template that eZ publish uses to display the full contents of an article object. The standard template is rather boring. We should really use a custom template instead. The following text explains how to generate a custom full-view-template for the "Article" content class within the "News section". Bring up the administration interface.

Click on the "Setup" tab.  
Click on "Templates" (in the menu on the left hand side).  
A list of template files will appear.

Locate "/node/view/full.tpl" and click on it.  
Click "Create New" to create a new template.  
Type in "full\_view\_news\_article" into the "Name" field.  
Set the "Class" dropdown box to "Article".  
Set the "Selection" dropdown box to "News section".  
Do not enter anything into the node field (leave it blank).  
In the "Base template on" section, select "Empty file".  
Click "Create".

This will generate an empty file  
("design/tscm/override/templates/full\_view\_news\_article.tpl") and instruct eZ publish  
to use it every time an article that resides in the "News section" is viewed. Put the  
following lines into the newly generated file:

```
{* Display the headline, use huge characters. *}  
<div class="headline">  
  {attribute_view_gui attribute=$node.object.data_map.title}  
</div>  
  
{* Display the thumbnail image, right-justified. *}  
<div class="imageright">  
  {attribute_view_gui attribute=$node.object.data_map.thumbnail}  
</div>  
  
{* Display the intro-text using bold characters. *}  
  
  {attribute_view_gui attribute=$node.object.data_map.intro}  
  
{* Display the actual body/content of the article. *}  
{attribute_view_gui attribute=$node.object.data_map.body}
```

This code will take care of displaying an article in a nice way. Try to access one of the articles  
from within the "Latest News" page (by clicking on either a headline or an image). You  
should be presented with something like this:



## News archive

In the news archive, we simply wish to list all the news articles that have been published. The list should contain the headline/title of the articles and the date they were published.

We could produce the archive by overriding the folder class' standard full-view template within the news section. However, this template has already been overridden when we created a custom template for the news page. A possible solution is to create a new view. We'll call it "archive-view". The following text explains how to create a new view and how to make a custom template for it. In addition, the manual creation of an URL alias is also explained; we'll create a friendly URL for the news archive.

### Creating a new view

The following text explains how to create a new view.

Create the following directories:

"design/tscm/templates/node"

"design/tscm/templates/node/view"

Create an empty file and save it as "design/tscm/templates/node/view/archive.tpl".

## Creating a custom template

The following text explains how to create a custom template for the newly created view.

Bring up the administration interface.

Click on the "Setup" tab.

Click on "Templates" (in the menu on the left hand side).

A list of template files will appear.

Click on an arbitrary template file.

A new page with a dropdown box will appear.

Make sure the dropdown box is set to "tscm".

Click set.

Go back to the template list (using the menu on the left hand site).

Jump to the last page by using the navigator at the bottom of the page.

Locate "/node/view/archive.tpl" and click on it.

On the next page, click "Create New".

Type in "archive\_view\_news\_folder" into the "Name" field.

Set the "Class" dropdown box to "Folder".

Set the "Selection" dropdown box to "News section".

Do not enter anything into the node field (leave it blank).

In the "Base template on" section, select "Empty file".

Click "Create".

This will generate the "design/tscm/override/templates/archive\_view\_news\_folder.tpl" template file. eZ publish will use this template every time it is instructed to display an archive-view of a folder within the News section. Put the following lines into the newly created template file:

```
<div class="pagetitle">
  News archive
</div>

{* Grab all the news articles. *}
{let children=fetch( content,
                    list,
                    hash( parent_node_id, $node.node_id,
                        sort_by, $node.sort_array,
                        class_filter_type, include,
                        class_filter_array, array( 'article' )
                    )
                )
}

<table class="news_archive">
  <tr>
    <td>
      Article:
    </td>
    <td>
      Published:
    </td>
  </tr>

  {* Loop through all articles that we just fetched. *}
  {section name=Child loop=$children}
  <tr>
    <td>
      {* Display a link to the article. *}
      <a href={$:item.url_alias|ezurl}>{$:item.name}</a>
      <br />
    </td>
  </tr>
</table>
```

```

        </td>
        <td>
            { * Display the date the article was published. * }
            { $:item.object.published|l10n(shortdate) }
        </td>
    </tr>
    { * End of loop. * }
    { /section }
</table>

{/let}

```

This code will take care of listing all the news articles that are published in a nice fashion.

### Creating an URL alias for the News archive

At this point, the news archive can only be reached using a system URL that tells eZ publish which view to use and which node to show ("[\[...\]/content/view/archive/\[node-number\]](#)" - where [node-number] is the identification number of the node that "contains" the News folder object). In order to be able to reach the news archive using a friendly URL, we must manually create a URL alias for it. The following text explains how to do this.

Bring up the administration interface.  
 Click on the "Content" tab.  
 The system will display contents of the root folder.

While hovering your mouse pointer over the "News" folder, look in the status bar of the browser. It should display something like:  
 "[http://\[...\]/tscm\\_admin/content/view/full/\[node-number\]](#)"  
 Make a note of the node number.

Click on the "Setup" tab.  
 Click on "URL translator" (in the menu on the left hand side).  
 In the "System URL" field, type in: "[content/view/archive/\[number\]](#)"  
 In the "Virtual URL" field, type in: "[news/archive](#)"  
 Click "Add".

What we have done is that we've instructed eZ publish to recognize the friendly URL, and when accessed, display the news archive. At this point, you should be able to access the news archive using both the system URL and the newly created virtual URL:  
[http://\[...\]/index.php/tscm/content/view/archive/\[node-ID-of-News-folder\]](#)  
[http://\[...\]/index.php/tscm/news/archive](#)

Try to browse these URLs. In either case, the browser should display something that looks like this:



The following text explains how to create a link that can be used to access the news archive.

### Creating the actual link to the archive

We'll make the archive accessible from the bottom of the latest news page. Insert the following code at the end of the "design/tscm/override/templates/full\_view\_news\_folder.tpl" file:

```
<br>
<div class="center">
<a href="{ "/news/archive" |ezurl}>
Click here to access the news archive...
</a>
</div>
```

## The members page

The [case](#) states that the TSCM people would like to have some sort of member-list. We could create this statically, but what fun would that be? Also, simple list of members is rather boring. We could easily store information about the members in the CMS. Lets keep things at a simple level and divide the member section into two pages:

Member-list page

Member information page

The member-list page should contain a list of all the club-members. Each row in the list showing the name of a member. The names could be displayed as hyperlinks. When one of these links is accessed, the system should take us to another page (the member information page). This page could be used to display all the information that we have about a member.

## Creating a custom content class

As pointed out in the previous section, we'll use the CMS to store information about the members. Lets say that we would like to store the following information (for each member):

Name  
Nickname  
Date of birth  
Gender (Male/Female)  
Telephone number  
E-mail address  
Facial photo

But wait! How can we store all this custom information? If you take a look at the list of the built-in content classes, you'll discover that none of them is even close to being suitable for storing information about TSCM members. What we have here is a typical content management problem. Fortunately, unlike other content management systems, eZ publish allows the users to create their own content classes. Instead of having to fit our data into a predefined, rigid structure, we have the power to create our own custom structure. The following list of steps explains how to do this.

Make sure that you're logged into the administration interface.  
Click on the "Setup" tab.  
Click on "Classes" - from within the menu on the left hand side.  
A list of class groups will appear.

Click on "Content" (from within the list).  
A list of available classes will appear.

Click on the "New class" button.  
The class edit page will appear.

eZ publish will then start the creation process of a new, custom content class. We'll need to specify a name for this class, lets just call it "Member". An identifier will also have to be specified. Each content class may be identified using its ID number or identifier string. The identifier is a friendly string. It is easier to remember and more flexible than an ID number.

Type "Member" into the name field.  
Type "member" into the identifier field.

We'll now start adding attributes to our new "Member" content class. The following text explains how to add the desired attributes.

### **The "Real name" attribute.**

This attribute will be used to store the real name of a member. We'll use the built in "Text line" datatype for this attribute. Since we can't have a member without a name, we'll set the required flag. This means that every time somebody tries to add or modify a member, the system will not update that member as long as the name field is empty.

Select "Text line" from the "Datatypes" dropdown box.  
Click the "New" button (adjacent to the "Datatypes" dropdown box).  
A new attribute will appear.

Type "Real name" into the name field.  
Type "realname" into the identifier field.  
Make sure that the "Required" and "Searchable" checkboxes are checked.  
The rest of the checkboxes should be unchecked.

You've now added the first attribute to the "Member" class. Since the "Searchable" checkbox is checked, this attribute will be searchable by the built-in search engine. In other words, it will be possible to search for members (by name). This functionality will be revealed and explained at a later stage.

#### **The "Nickname" attribute.**

Some chess fanatics tend to have lame nicknames such as "King", "Knight-rider", "Rookie" etc. The nickname attribute will be used to store a possible nickname.

Select "Text line" from the "Datatypes" dropdown box.  
Click the "New" button (adjacent to the "Datatypes" dropdown box).  
A new attribute will appear.

Type "Nickname" into the name field.  
Type "nickname" into the identifier field.  
Type "Nickname missing." into the default field.  
Make sure that the "Searchable" checkbox is checked. The rest of the checkboxes should be unchecked.

#### **The "Date of birth" attribute.**

This attribute will be used to store the birthdate of a member. We'll use the built-in "Date field" datatype.

Select "Date field" from the "Datatype" dropdown box.  
Click the "New" button (adjacent to the "Datatypes" dropdown box).  
A new attribute will appear.

Type "Date of birth" into the name field.  
Type "birthdate" into the identifier field.  
Make sure that the "Required" checkbox is checked. The rest of the checkboxes should be unchecked.

#### **The "Gender" attribute.**

This attribute will be used to specify whether a member is a male or a female specimen. The "Selection" datatype will be used. This datatype is perfect for quick'n dirty single and multiple choice scenarios.

Select "Selection" from the "Datatype" dropdown box.  
Click the "New" button (adjacent to the "Datatypes" dropdown box).  
A new attribute will appear.

Type "Gender" into the name field.  
Type "gender" into the identifier field.  
Click on the "New option" button.  
You'll now have two options.

Type "Male" into the first option field.  
Type "Female" into the second option field.



Make sure that the "Required" checkbox is checked. The rest of the checkboxes should be unchecked.

#### **The "Telephone number" attribute.**

This attribute will be used to store a member's telephone number. Since some people don't have a phone, we'll not mark this attribute as required. We'll use the "Text line" attribute to store phone numbers.

Select "Text line" from the "Datatype" dropdown box.  
Click the "New" button (adjacent to the "Datatypes" dropdown box).  
A new attribute will appear.

Type "Telephone number" into the name field.  
Type "phone" into the identifier field.  
Type "Phone number missing." into the default field.  
Leave the default checkbox settings.

#### **The "E-mail address" attribute.**

This attribute will be used to store a member's electronic mail address. Since some people don't have an E-mail address, we'll not mark this attribute as required. We'll use the "E-mail address" attribute to store the members e-mail addresses.

Select "Email" from the "Datatype" dropdown box.  
Click the "New" button (adjacent to the "Datatypes" dropdown box).  
A new attribute will appear.

Type "E-mail" into the name field.  
Leave the default checkbox settings.

#### **The "Picture" attribute.**

This attribute will be used to store a digitized photograph of a member's face. We'll only allow image files that are smaller than one megabyte.

Select "Image" from the "Datatype" dropdown box.  
Click the "New" button (adjacent to the "Datatypes" dropdown box).  
A new attribute will appear.

Type "Picture" into the name field.  
Type "picture" into the identifier field.  
Set the maximum file size to 1 MB.

That's it. The "Member" class is now made up of exactly those seven attributes that we wished for at the beginning. Click the "Store" button; eZ publish will then store the class definition in the database. The next section explains how to add member objects.

## Adding members

The previous section explained how to build a custom content class for storing information about members. We'll now use the administration interface to create a couple of instances of this class. In other words, we'll create a bunch of member objects and put some data into them.

Lets create a couple of members. They should all be put into a dedicated "Members"

folder. By now, you should be able to do the following operation(s) without following a detailed step-by-step guide.

Create a folder called "Member" (in the root of the content folder).

Assign this folder to the "Members section".

Navigate into the members folder.

Create a couple of member objects.

Feel free to use the example data provided below.

**Example data:**



Alexander Ferrari  
Happy King  
1970.02.02  
Male  
555-3212  
alex@example.com  
[alex\\_ferrari.png](#)

Henry Von Metal  
Draculaboy  
1967.03.02  
Male  
555-3456  
dracula@example.net  
[henry\\_metal.png](#)



Pia Pianissimo  
LadyQueen  
1975.07.06  
Female  
555-5555  
lollipop@example.com  
[pia\\_pianissimo.png](#)

Thomas Hellbender  
Dark Bishop  
1977.05.12  
Male  
666-0911  
hellraiser@example.org  
[thomas\\_hellbender.png](#)

# List of members

The member-list page will contain a list of all the club members. Each row in the list will show the name of a member. The following text explains how to create the member list page.

First of all, add the following lines to the CSS file:

```
.member_list
{
  width: 85%;
  font-size: 80%;
}
```

Make sure that the "Members" link in the main menu points to the "Member" folder.

Create a new override for the "/node/view/full.tpl" template.

Feel free to call it "full\_view\_member\_folder.tpl".

Set the override keys to the "Folder" class and the "Member" section.

Replace the contents (if any) of the generated template file with the code provided below.

```
<div class="pagetitle">
  Current members
</div>

{* Grab all the child nodes. *}
{let children=fetch( content, list, hash( parent_node_id, $node.node_id,
                                         sort_by,
                                         $node.sort_array ))}

  <table class="member_list" cellpadding="0" cellspacing="0"
border="0">

  {* Loop through all nodes that we just fetched. *}
  {section name=Child loop=$children}

  member. *}
  <tr>
    <td>
      <a href={$:item.url_alias|ezurl}>{:item.name}</a>
    </td>
  </tr>
```

```
        { * End of loop. *}
        {/section}

</table>

{/let}
```

This code will list the name of all the members. Try to surf the member page. You'll see that the correct template pops up. However, despite that we've added members to the member folder, no names will appear. This phenomenon is normal. By default, content that is stored using custom classes is not accessible to anonymous users. We'll need to allow anonymous web surfers to read content that is stored in "Member" objects. The following text explains how to do this.

Make sure that you're logged into the administration interface.

Click on the "Users" tab.

A list of users/groups will appear.

Click on "Roles" (from within the menu on the left hand side).

A list of roles will appear.

Click on the "Anonymous" user.

The properties of the Anonymous user will appear.

Click on the "Edit" button.

Click the "New" button.

Select "Content" from the "Give access to module" dropdown box.

Click the "Allow limited" button.

Select "Read" from the "Function" dropdown box.

Click the "Allow limited" button.

Set the class to "Member", the section and the owner to "Any".

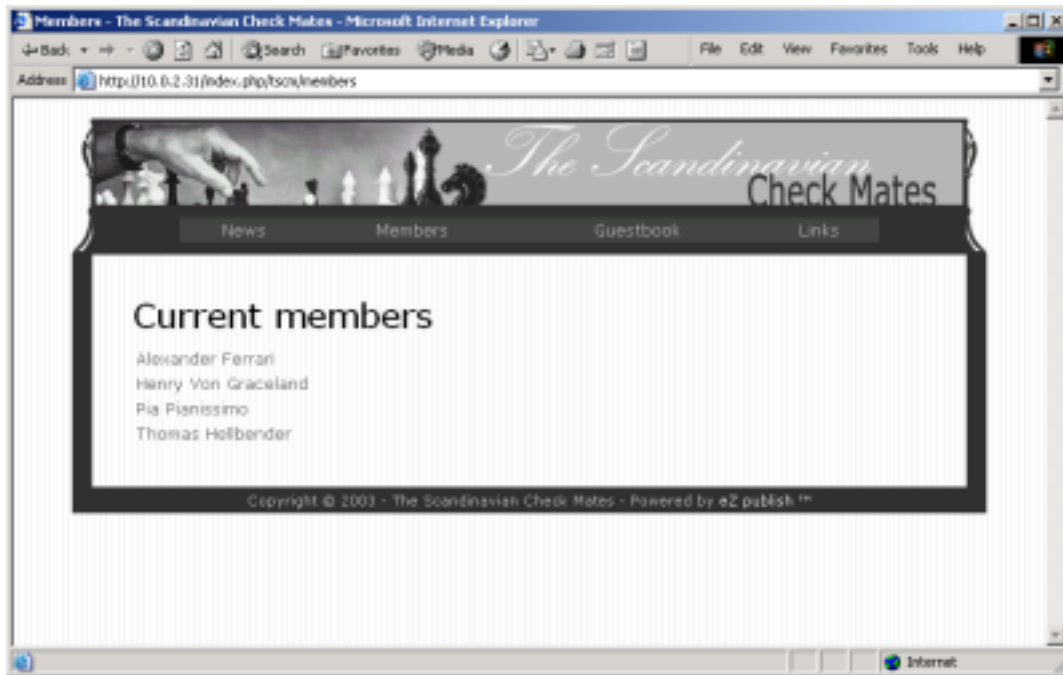
Click the "OK" button.

Click the "Store" button.

The properties of the anonymous user will be displayed again. The last line should read something like "Content Read Class(Member)" - which means that the anonymous user has read-access to member objects in the content module.

Try to browse the "Members" page again. If everything was done correctly, you'll be able to see a list of members; the page should look something like this:





## Member info page

This page will be used to display all the information that we have about a member. Currently, when a member is accessed (using one of the links from within the member list page), eZ publish displays all the attributes of that member using the default object view. The following list of steps explain what you should do to change this.

Append the following lines of code to the TSCM CSS file.

```
.member_info
{
    border-style: none;
    font-size: 80%;
}
```

Create a new override for the "/node/view/full.tpl" template.

Feel free to call it "full\_view\_member\_class.tpl".

Set the override keys to the "Member" class and the "Member" section.

Replace the contents (if any) of the generated template file with the code provided below.

```
<div class="pagetitle">
    Member information
</div>

<div class="imageleft">
```

```

        { * Display a picture of the member. * }
        {attribute_view_gui attribute=$node.object.data_map.picture}
</div>

{ * Display the attribute names and their values. * }
<table class="member_info" cellspacing="4" cellpadding="3">
  <tr>

<td>{ $node.object.data_map.realname.contentclass_attribute.name }:</td>
  <td>{attribute_view_gui
attribute=$node.object.data_map.realname}</td>
  </tr>
  <tr>

<td>{ $node.object.data_map.nickname.contentclass_attribute.name }:</td>
  <td>{attribute_view_gui
attribute=$node.object.data_map.nickname}</td>
  </tr>
  <tr>

<td>{ $node.object.data_map.birthdate.contentclass_attribute.name }:</td>
  <td>{attribute_view_gui
attribute=$node.object.data_map.birthdate}</td>
  </tr>
  <tr>

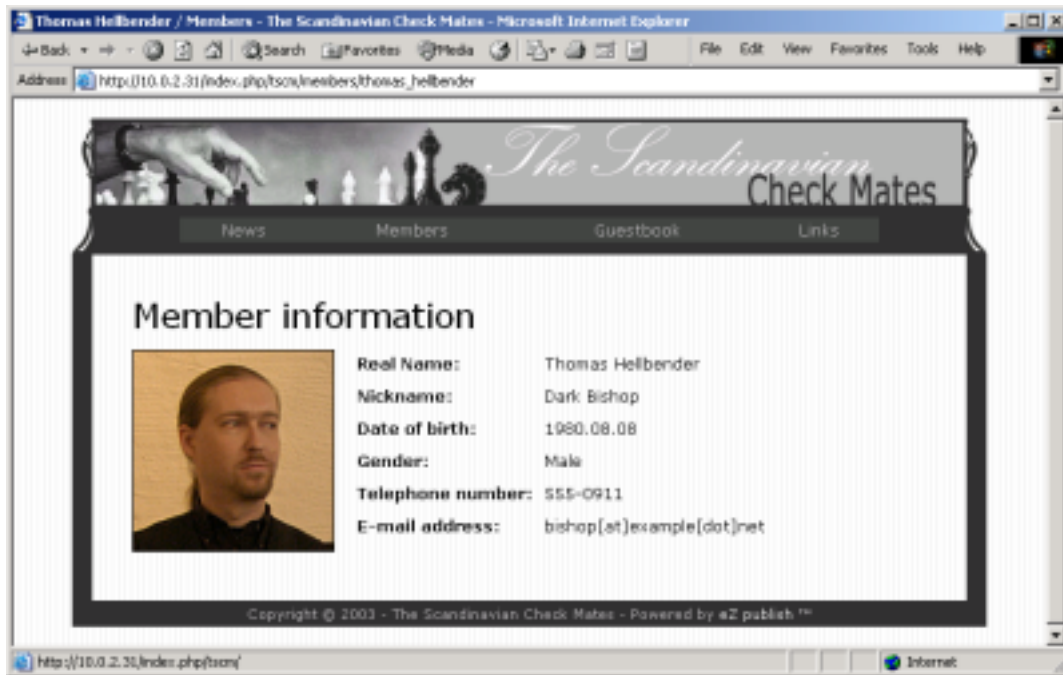
<td>{ $node.object.data_map.gender.contentclass_attribute.name }:</td>
  <td>{attribute_view_gui
attribute=$node.object.data_map.gender}</td>
  </tr>
  <tr>

<td>{ $node.object.data_map.phone.contentclass_attribute.name }:</td>
  <td>{attribute_view_gui
attribute=$node.object.data_map.phone}</td>
  </tr>
  <tr>

<td>{ $node.object.data_map.email.contentclass_attribute.name }:</td>
  <td>{attribute_view_gui
attribute=$node.object.data_map.email}</td>
  </tr>
</table>

```

This code takes care of displaying member information in a nice way. The picture of a member is displayed at the left hand side. Information about the member (name, phone number, etc.) is displayed on the right hand side (prepended by the name of the attribute). Notice that the names of the attributes is not hardcoded in the template file. Whenever you decide to rename an attribute (for example replace "Real name" with "Name" - you will not have to bother with updating the template file. The following screenshot shows what the output should look like at this point.



## The guestbook

The [case](#) states that "The Scandinavian Check Mates" would like to have a guestbook of some sort. The guestbook is somewhat different from what we've been doing so far. It will provide functionality that allows anonymous surfers to add content to the CMS of eZ publish through the TSCM website. In addition, we'll make use of the workflow system to approve guestbook entries.

The guestbook section of the site will basically consist of two parts:

List of entries

Entry submit page

The list of entries will be a read-only page showing all the entries that have been added to the guestbook. The submit page will be accessible from the "List of entries" page. It will display a form (with a couple of text fields), a "Submit" and a "Cancel" button.

## Creating the content class

Since eZ publish doesn't have a built-in content class that is 100% suitable for our guestbook needs, we'll simply build our own. Let's say that for each guestbook entry, we would like to store two things:

The name of the person who signs the guestbook

The entry/text itself

Use the administration interface to build a content class called "Guestbook entry". If you're not used to creating custom classes, read the ["The members page"](#) section again.

Make sure that the class is built up of a "text line" and a "text field". The text line will be used to store the name/nick of the person who submits an entry. The text field will be used to store the entry itself. Make sure that the text line is the first attribute and call it "Name". Let the identifier be "name". The second attribute (the text field) should be called "Entry" and its identifier should be "entry". The guestbook doesn't have to be searchable, so feel free to turn off the searchable options. Make sure that both attributes are required. The length of the first attribute should be limited to 32 characters. The "Preferred number of rows" of the second attribute should be set to 15.

Finally, make sure that the anonymous role has read access to the newly generated content class. If you're unsure about this, read the ["List of members"](#) section again.

## Adding content

In order to be able to test the guestbook properly, we'll need to add some content to it. Use the administration interface to create a folder called "Guestbook" and put a couple of guestbook entries in it. Make sure that the nodes in the folder are sorted by the date/time they were published (in the reverse order). To do this, select "Published" from the the "Sort by" dropdown box from within the edit page of the "Guestbook" folder. In addition, select the "Ordering" radiobutton that is on the right hand side (adjacent to "Main"). This will instruct eZ publish to show the most recent entries on the top of the list. If you lack inspiration, feel free to use the following example data for the guestbook folder:

### **Folder description:**

*This is the official guestbook of The Scandinavian Check Mates. Feel free to add an entry (press the "Sign the guestbook" button that is below this text). Please note that the TSCM club reserves the right to moderate and/or remove offensive and tasteless entries.*

### **Guestbook entries:**

Name: *Madonna DeLafore*

Entry: *Hello! My name is Madonna and I would really like to join your club. However, I must say that the new club-site looks a bit dull. I've heard rumors about webshop functionality, tree-menus and polls, will this be implemented in the near future? I hope so. Take care!*

Name: *Bill Collins*

Entry: *Your lame jimbo jumbo chess club sucks! We have a much better chess club in Porsgrunn with a lot of geeks who are much much smarter than you! This site is a joke. I know that our Porsgrunn club doesn't have a site but thats just because we're cool. Cool people don't need a silly website to gain respect.*

Name: *Tina Turnoff*

Entry: *Hi! Currently, I am a member of the Porsgrunn chess club. However, after surfing your site,*

*I must say you guys seem so exciting. Could I please, please join your club? Pretty please, with sugar on top! I'm aware of that I have to move to Skien and I'm very much prepared to do so! PS: Bill is my ex-boyfriend and he is an idiot.*

## Creating the template

Use the administration interface to create a template override for the guestbook folder. Hints: Remember to assign the guestbook folder to the guestbook section. Override the "/node/view/full.tpl" template for the folder class within section "Guestbook". Name the new template "full\_view\_guestbook\_folder". Base the template on an empty file. In addition, you should also make sure that the "guestbook" link in the main menu actually points to the guestbook. When you're finished with all this, attempt to write your own template code that takes care of displaying the following elements:

- The guestbook folder's description
- A list of all the guestbook entries

Put your code into the newly generated/empty "design/tscm/override/templates/full.tpl" file. If you're unsure about the template code (or if you're simply just lazy), feel free to peek at (or copy) the template code that is provided below.

```
<div class="pagetitle">
  {$node.name}
</div>

<table class="guestbook">
  <tr><td>
    {$node.object.data_map.description.content.output.output_text}
  </td><td>
    
  </td></tr>
  <tr><td>
    <!-- The button will be here! -->
  </td><td>
  </td></tr>
</table>

{* Grab all the guestbook entries. *}
{let name=test counter=0 children=fetch( content,
                                         list,
                                         hash( parent_node_id,
                                                $node.node_id,
                                                sort_by,
                                                $node.sort_array
                                              )
                                         )
}

<table class="news_archive">

{* Loop through all the entries. *}
{section loop=$:children}
```

```

{* Increment the counter by one. *}
{set counter=${counter|inc}

<tr><td>
  <hr />
  Submitted by {${item.object.data_map.name.content|wash}
  on {${item.object.published|l10n(shortdate)}:
  <br />
  <br />
  {${item.object.data_map.entry.content|wash}
</td></tr>

{* End of loop. *}
{/section}

  <tr><td><hr /></td></tr>
</table>

<div class="center">
  Number of entries in the guestbook: {${counter}
</div>

{* Release the counter and the children variable. *}
{/let}

```

Notice the use of the "wash" operator. It takes care of translating bogus strings into friendly ones. The wash operator will make sure that HTML tags found inside guestbook entries don't mess up the output. In other words, people who submit text with HTML tags will not be able to destroy the original page layout. Tags will be displayed as normal text (they will not be interpreted by the browser).

The template code above makes use of an image called "guestbook.png". Feel free to use the following icon/image: [guestbook.png](#). This image is non-content specific (it belongs to layout/design), so therefore, we'll place it inside the TSCM design directory. Download and place the image into the "design/tscm/images/" directory.

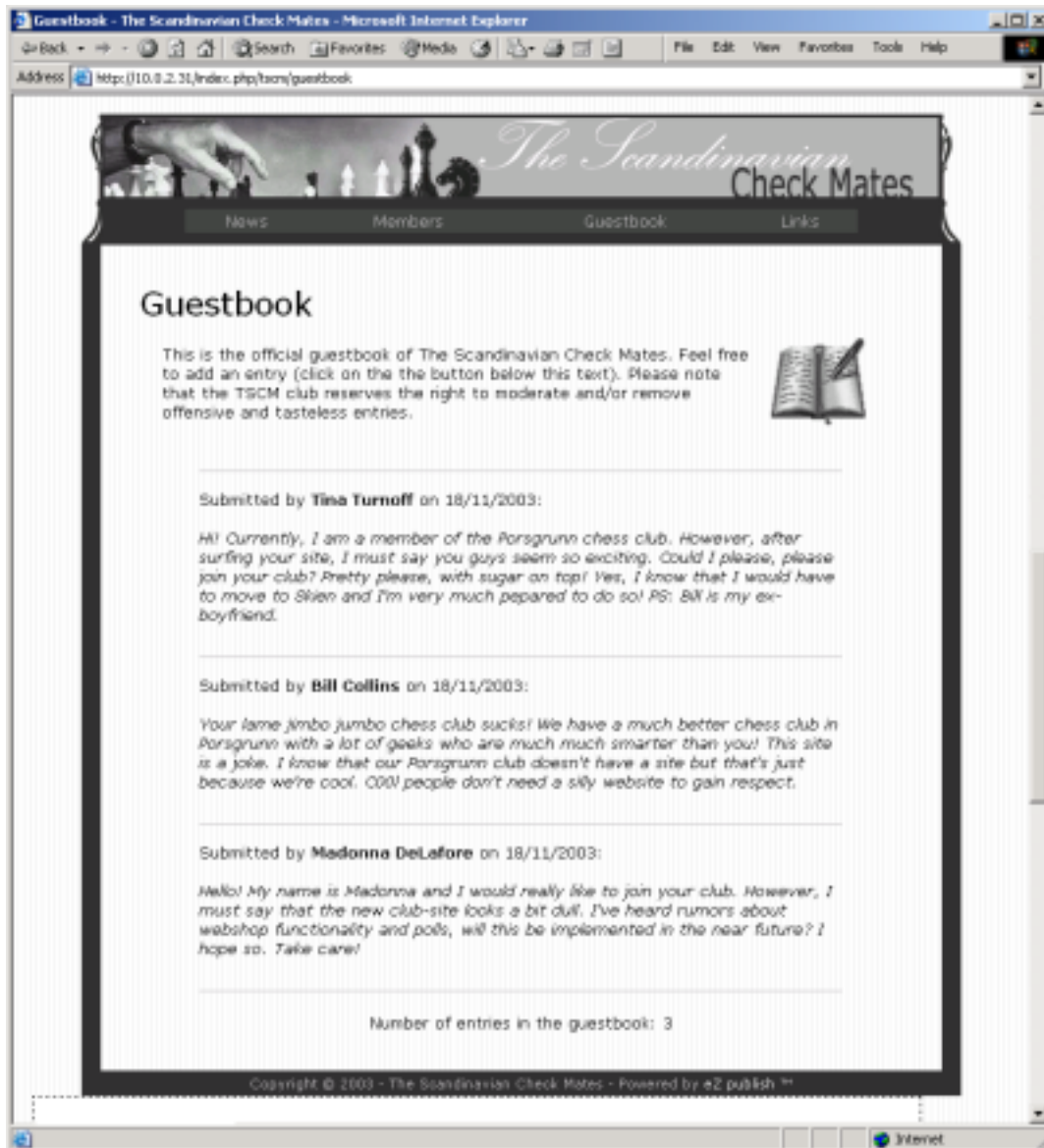
Some of the HTML code that was presented above also makes use of some additional CSS. Make sure you put the following lines into the TSCM CSS file (things might look a bit ugly without).

```

.guestbook
{
  margin-left: auto;
  margin-right: auto;
  width: 95%;
}

```

If you've used the example data and the example code, you should be looking at something that resembles the following screenshot (when accessing the guestbook page from the main menu).



## Adding an action button

Edit the template file that is used to display the guestbook page ("design/tscm/override/templates/full\_view\_guestbook\_folder.tpl"). Copy and paste the code that is provided below at a location where you wish to display the "Sign the guestbook" button. This button will take the user to the input form. If you used the suggested code from the previous section then find the HTML comment that says "The button will be here!" and simply replace it with the code that is provided below.

```
<form method="post"
      action={"content/action/" | ezurl}>
```

```

<input class="button"
      type="submit"
      name="NewButton"
      value="Sign the guestbook" />

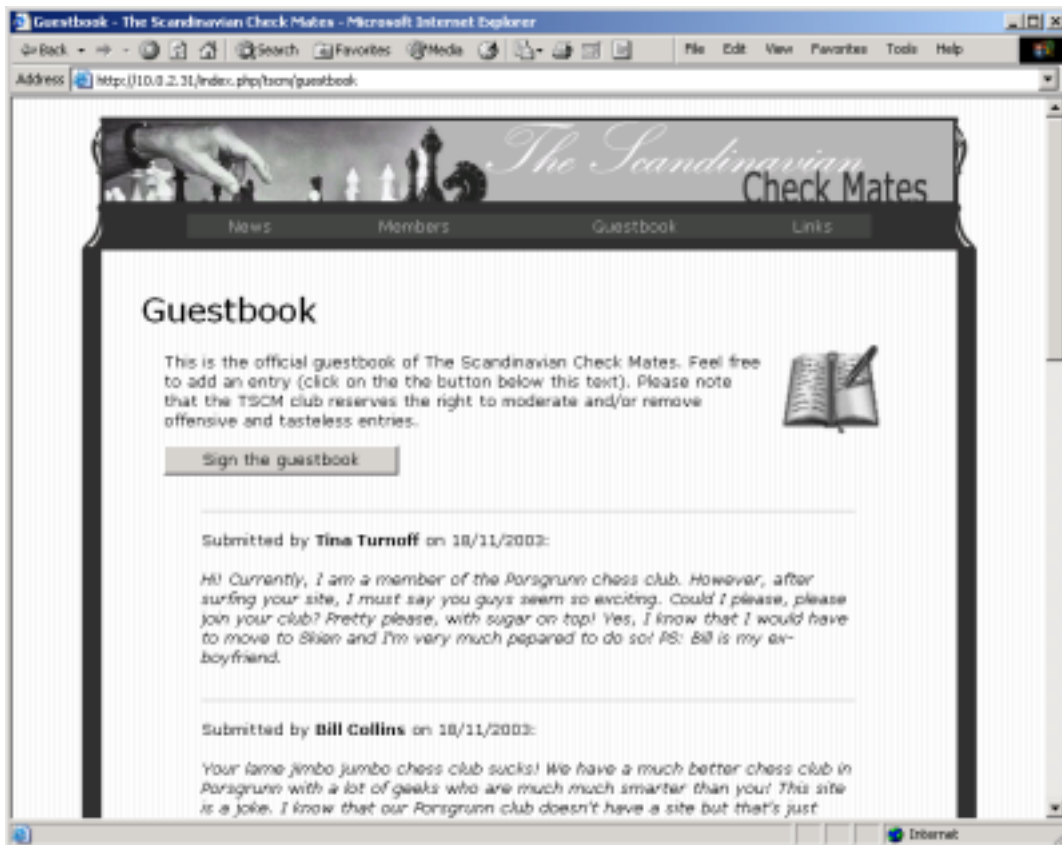
<input type="hidden"
      name="ClassID"
      value="__REPLACE_WITH_CLASS_ID__" />

<input type="hidden"
      name="NodeID"
      value="{ $node.node_id }" />
</form>

```

Make sure that you replace the "\_\_REPLACE\_WITH\_CLASS\_ID\_\_" part with the identification number of the "Guestbook entry" class. Hint: a class' identification number can be acquired by browsing "Classes" from within the "Setup" part of the administration interface.

Attempt to reload the guestbook page. You should now be able to see the "Sign the guestbook" button below the description text. By now, the page should look something like this:



Let's just take a moment and look at the code that was presented above. It is nothing more than a standard HTML form that uses the POST method to send data back to the web server. The "ezurl" operator will take care of translating the action string into "index.php/tscm/content/action/" - this will become the requested URL. For information about eZ publish URLs, please refer to the ["eZ publish URLs" section](#) within the [eZ publish basics](#) chapter. In this particular case, the URL will tell eZ publish that it should attempt to execute the



"action" function within the "content" module using the "tscm" siteaccess. Behind the curtains, eZ publish will actually run the "kernel/content/action.php" file.

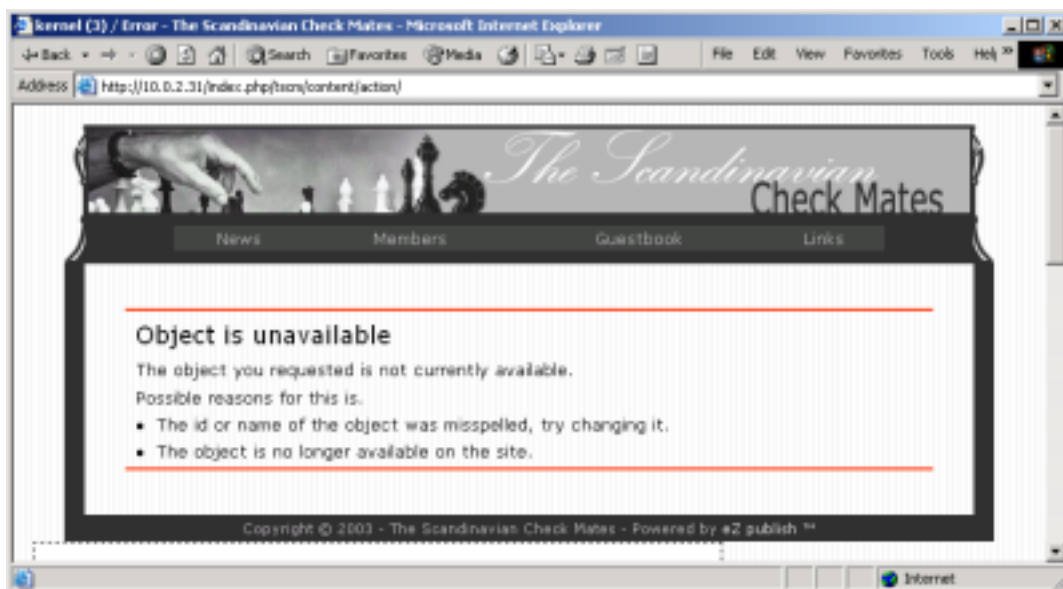
The second part/line instructs the browser to display a button labelled "Sign the guestbook". In addition, this line also contains the name of the eZ publish action that we wish to execute when the button is pushed. So, what should eZ publish do? Well, we wish to create a new guestbook entry object. In other words, we have to tell eZ publish that it should create a new object. The "NewButton" string instructs eZ publish to start the action-process of creating a new object.

The third part/line is used to inform eZ publish about the type of the object we wish to create. In this case, eZ publish should create a guestbook entry object (an instance of the guestbook entry class). In order to be able to create the desired class, eZ publish needs to know the identification number of the class that we wish to instantiate an object of.

The fourth part/line is used to provide eZ publish with an identification number of a node. This is simply done in order to pinpoint a location (of some node) within the content node tree. During a publishing process, eZ publish will place the newly created object in a new node. The new node will automatically become the child of the node that was revealed/pinpointed. In this particular case, it is the node containing the guestbook folder object that will be the parent node.

## Making the button work

If you try to click the "Sign the guestbook" button at this point, you'll probably meet a rude page that resembles the following screenshot.



That's right buddy, the object is unavailable! This phenomenon is perfectly normal; and the explanation is fairly simple. When clicked, the "Sign the guestbook" button that we just put in initiates an action within the content management system. eZ publish will attempt to create a new object using the current user, which in this case (since nobody is logged in) is the built-in

"Anonymous" user. Not surprisingly, the default security settings do not allow anonymous users to mess around with content. In order to make the button work, we'll simply have to let anonymous users to be able to do two things:  
Create guestbook entries within the guestbook section  
Edit the contents of guestbook entries

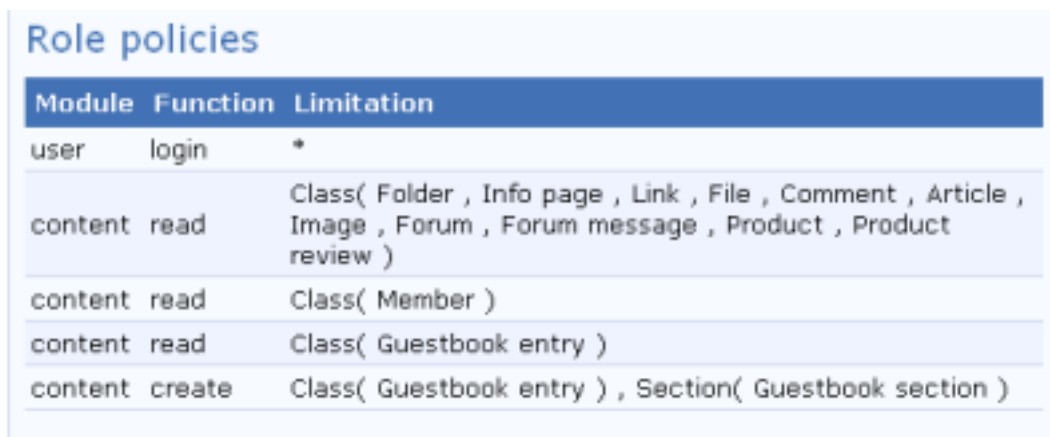
The following step-by-step guide explains how to enable this functionality.  
Make sure that you're logged into the administration interface.  
Click on the "Users" tab.  
A list of users/groups will appear.

Click on "Roles" (from within the menu on the left hand side).  
A list of roles will appear.

Click on the "Anonymous" user.  
The properties of the Anonymous user will appear.

Click on the "Edit" button.  
Click the "New" button.  
Select "Content" from the "Give access to module" dropdown box.  
Click the "Allow limited" button.  
Select "Create" from the "Function" dropdown box.  
Click the "Allow limited" button.  
Set the class to "Guestbook entry", the section to "Guestbook" and the Parent class to "Any".  
Click the "OK" button.  
Click the "Store" button.

At this point, the list of role policies for the anonymous user should look something like this:



Module	Function	Limitation
user	login	*
content	read	Class( Folder , Info page , Link , File , Comment , Article , Image , Forum , Forum message , Product , Product review )
content	read	Class( Member )
content	read	Class( Guestbook entry )
content	create	Class( Guestbook entry ) , Section( Guestbook section )

The previous list of steps took care of adding "Create" privileges to the anonymous role. In addition, we'll also have to add "edit" privileges to this role. Repeat the previous steps, but this time make sure that you select the "Edit" function (at step two) and that you select "Self" from within the owner list (at step three). The rest of the selections should be the same as in the list of steps presented above.

**IMPORTANT!**

When finished, make sure that you press the store button.

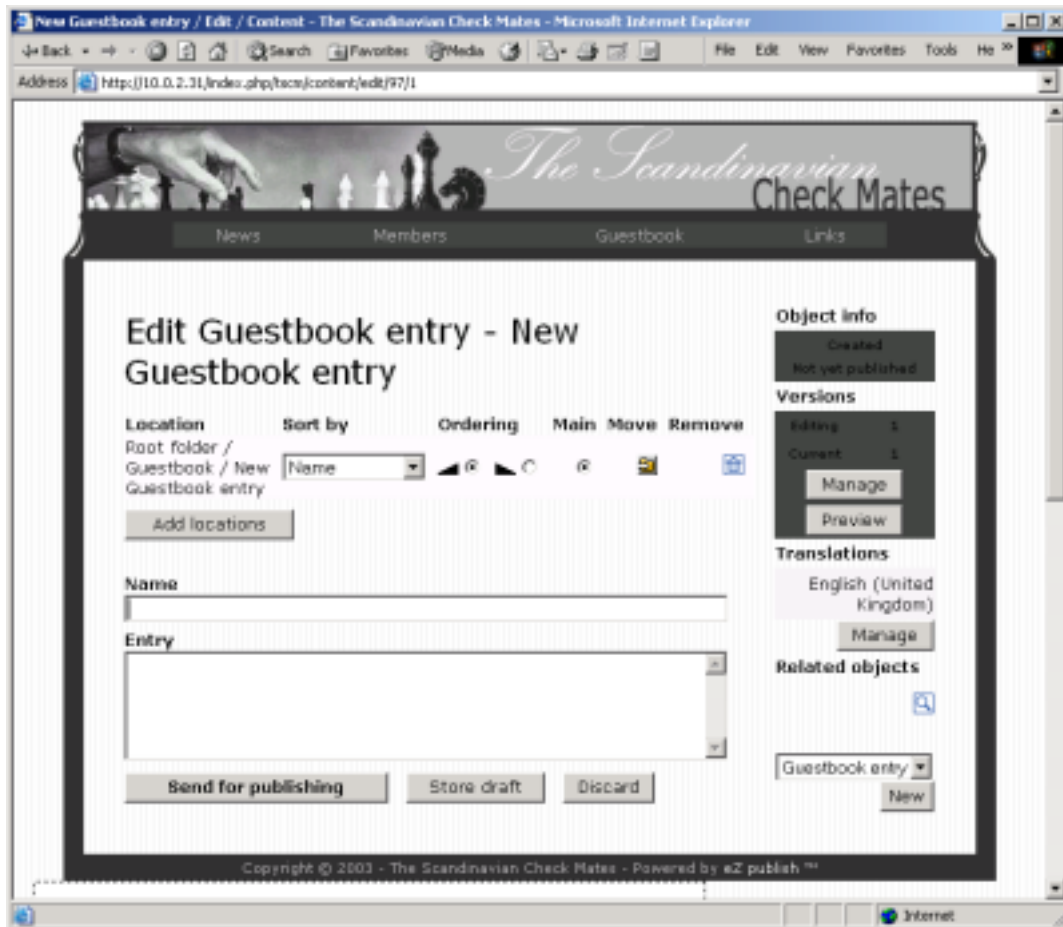
At this point, the list of role policies for the anonymous user should look something like this:

## Role policies

Module	Function	Limitation
user	login	*
content	read	Class( Folder , Info page , Link , File , Comment , Article , Image , Forum , Forum message , Product , Product review )
content	read	Class( Member )
content	read	Class( Guestbook entry )
content	create	Class( Guestbook entry ) , Section( Guestbook section )
content	edit	Class( Guestbook entry ) , Section( Guestbook section ) , Owner( Self )

## The input template

Go ahead and surf the "Guestbook" page again. Attempt to press the "Sign the guestbook" button (which at this point should work). Instead of the "Object unavailable" message, you should be presented with a page that resembles the following screenshot:



If it worked, it means that eZ publish allowed the anonymous user to create a new guestbook entry object. What you're looking at is the default edit template, which resides in the "design/standard/templates/content/edit.tpl" file. This template contains a lot of functionality that we don't need (version control, set.). What we need is a template that shows two fields (the name and the entry field) and two buttons (Submit and Cancel). The following text explains how to create a template override and a custom template for guestbook entries.

Until now, we've been doing template overrides by using the web based administration interface. The administration interface can only be used to create basic template override rules. Complex/advanced rules must be input manually to the override file of the target siteaccess. What the administration interface does is that it simply manipulates this file when you create/edit/delete template overrides. Let's create the guestbook-entry template override manually. The following list of steps should guide you through it.

Acquire and make a note of the identification number of the "Guestbook entry" content class. (Hint: Admin->Setup->Classes->Content)

Acquire and make a note of the identification number of the guestbook section. (Hint: Admin->Setup->Sections)

Edit the "settings/siteaccess/tscm/override.ini.append" file.

Notice that the file contains the overrides that we've added so far.

Scroll down to the end of the file and add the following lines to it:

```
[edit_guestbook_entry]
Source=content/edit.tpl
```

```
MatchFile=edit_guestbook_entry.tpl
Subdir=templates
Match[class]=__REPLACE_WITH_CLASS_ID__
Match[section]=__REPLACE_WITH_SECTION_ID__
```

Make sure that you replace the "\_\_REPLACE\_WITH\_CLASS\_ID\_\_" with the identification number of the "Guestbook entry" content class. You'll also need to replace the "\_\_REPLACE\_WITH\_SECTION\_ID\_\_" with the identification number of the guestbook section. When finished, save the file, clear all caches, surf the guestbook and press the "Sign the guestbook" button again. At this point, the system should not display anything. That is because the "edit\_guestbook\_entry.tpl" file doesn't exist. Let's fix this problem. Create the new template file ("design/tscm/override/templates/edit\_guestbook\_entry.tpl") and put the following lines of code into it:

```
<div class="pagetitle">
    Add a guestbook entry
</div>

<form enctype="multipart/form-data"
    method="post"
    action={concat( "/content/edit/" ,
                    $object.id,
                    "/" ,
                    $edit_version) |ezurl}>

<table class="layout">
    <tr><td>

        {include uri="design:content/edit_validation.tpl"}
        {include uri="design:content/edit_attribute.tpl"}

        <div class="buttonblock">
            <input class="defaultbutton"
                type="submit"
                name="PublishButton"
                value="Submit" />

            <input class="button"
                type="submit"
                name="DiscardButton"
                value="Cancel" />

            <input type="hidden"
                name="MainNodeID"
                value="{ $main_node_id}" />
        </div>
    </td></tr>
</table>
</form>
```

The action URL is constructed by concatenating several strings. The final string is translated using the "ezurl" operator. The URL reveals the module that we wish to access (content), the function that should be executed (edit), the identification of the object (\$object.id) and the version (\$edit\_version) that should be edited. Since this interface is only meant to be used to create new nodes (and not edit existing nodes), the version number will always be one.

The "edit\_validation.tpl" takes care of validating the input once it is submitted. Omitting this line will still not allow the submission of partially or completely empty entries. The system will simply not display the warning. Since we've set both the entries to be required (when we created the "Guestbook entry" class), eZ publish will not allow empty entries.

The "edit\_attribute.tpl" takes care of displaying the actual input fields and their names.

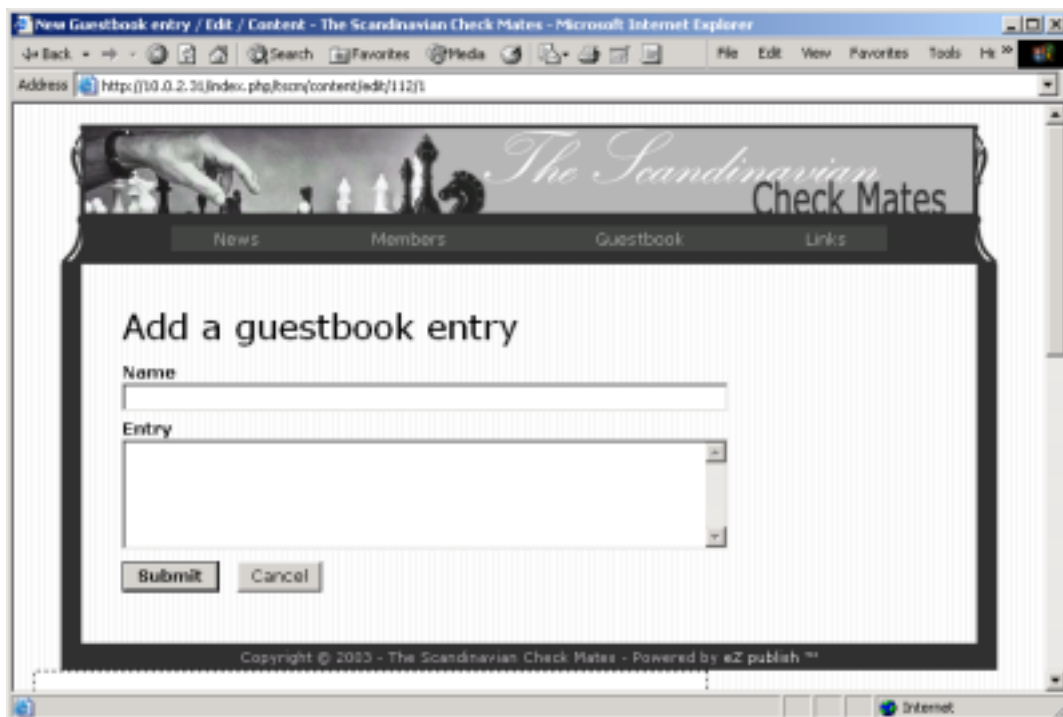
The "PublishButton" string (name of the "Submit" button) instructs the system to publish the object that is being edited.

The "DiscardButton" string (name of the "Cancel" button) instructs the system to discard the object that is being edited.

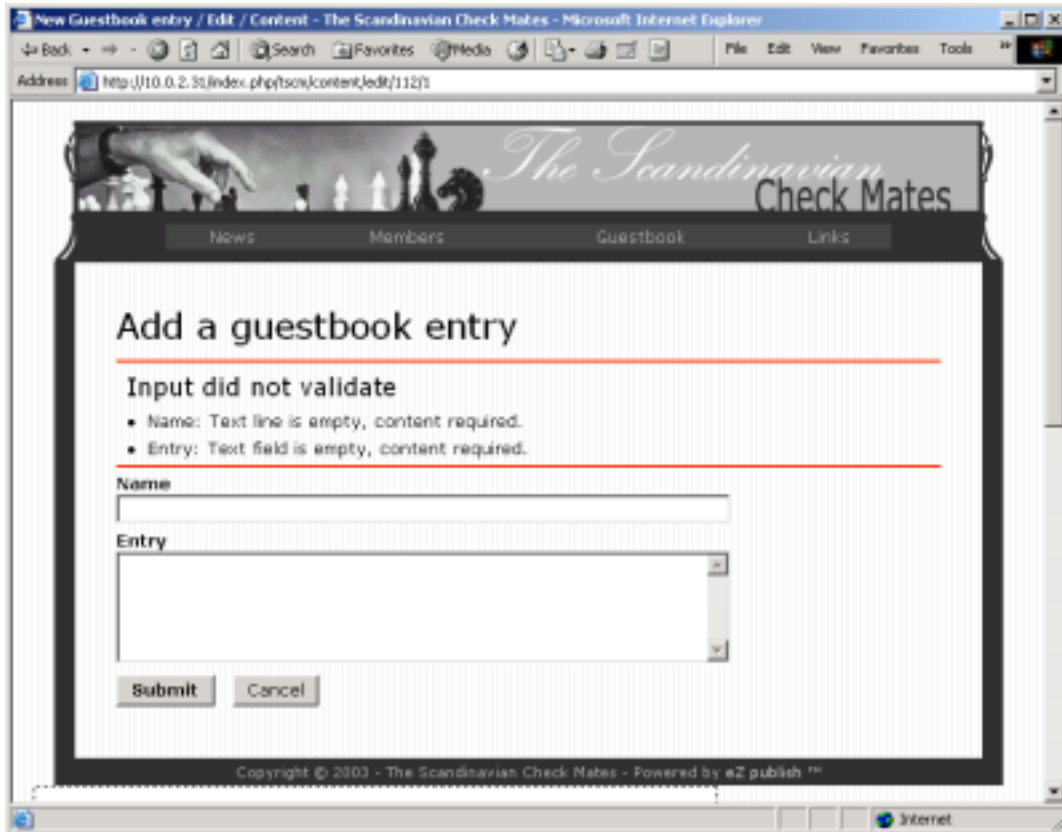
The hidden "MainNodeID" field reveals the identification number of the node that will be the parent of the node that is being published. This is the main location of the node that is being published. Without this field, eZ publish will not be able to publish the object.

## Testing the guestbook

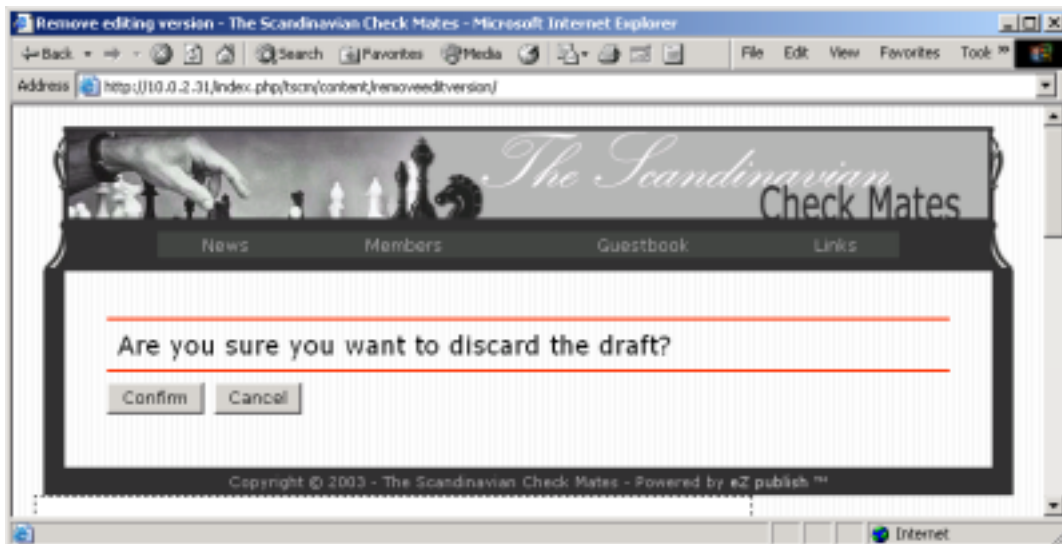
Try pressing the "Sign the guestbook" button again. The system should then generate a page that looks something like the following screenshot:



Try to submit an empty entry, eZ publish should not allow this and it will present you with a message that resembles the one in the following screenshot:



Now, put some text into the fields and simulate a user error by hitting the "Cancel" button (instead of "Submit"). The system should respond with the following page:



As you can see, eZ publish will kindly ask if you're sure about the cancel operation. Since we pressed "Cancel" by accident, let's inform the system about this (hit "Cancel" again). You should be back at the "Add a guestbook entry" page with the content that you just typed in preserved in the input fields. Press "Submit". eZ publish will then add the entry to the guestbook. The system should bring you back to the guestbook page and you should be able to see the new entry at the top of the list. This concludes the basic guestbook functionality.

# Implementing an approval mechanism

The guestbook that we've created so far allows anonymous people to "publish" arbitrary information on the TSCM website. Sometimes this is a good idea, sometimes it is not. A classic example is that malicious people misuse such functionality to post tasteless and/or offensive text. To prevent this problem, we could make use of some sort of a mechanism that ensures the approval of content before it is published. eZ publish provides a built-in workflow system that is designed to deliver such functionality (and a lot more). Please note that the workflow system is huge and complex. In this part of the tutorial, we will only use it to create a simple workflow for the purpose of demonstration and learning. However, I'm sure that you'll quickly understand and discover the wonderful and waste possibilities of the eZ publish workflow engine.

## Modifying the entry page

From now on, our policy is that all guestbook entries will have to be approved by the site administrator. New entries will not pop up instantly in the list of entries. We should inform the users about this. Let's modify the "Add an entry" page. Go ahead and edit the template that is used to input/post a guestbook entry and add the following text:

*Please note that your signature/entry will not be visible right away. Our policy is that all content (guestbook entries included) has to be approved by the the site administrator. Please check back within a couple of hours/days to see if your entry was approved or not.*

At this point, the guestbook entry page should look something like this:

## Creating a workflow

We'll create a simple workflow. It will be used to ensure the approval of guestbook entries made by anonymous users. The following list of steps should take you through it.

Log into the administration interface.

Click on the "Setup" tab.

Click on "Workflows" (from within the menu on the left hand side).

A list of workflow groups will appear.

Click on the "Standard" group.

An empty list should appear.

Press the "New workflow" button.



The "Editing workflow" page will appear.

Provide a custom name for this new workflow (fill out the name field). For example, you could call it "Guestbook entry approval".

In the events section, set the dropdown box to "Event/Approve".

Click the "New" button.

A new event will be added to this workflow.

Feel free to type something into the description field. You may also leave it blank.

Set the "Editor" list to "admin". This field is used to define the target user (the user that will be doing the approval).

Select the "Guestbook section" (set it in the sections field).

Set the "Users without approval" to "Administrator users".

Press the "Store" button.

That's it. You have now created a workflow definition.

## Connecting the workflow to a trigger function

What we have done so far is that we have created a new workflow called "Guestbook entry approval". This workflow alone doesn't do anything. It has to be assigned/connected to a trigger function. Without a trigger function, the workflow definition will simply sit there and do nothing. Go ahead and carry out the following steps:

Make sure that you're logged into the administration interface.

Click on the "Setup" tab.

Click on "Triggers" (from within the menu on the left hand side).

A list of triggers will appear.

Locate the line that says "Content - Publish - Before" and set the corresponding dropdown box to "Guestbook entry approval".

Press the "Store" button.

Now, surf the TSCM site again and attempt to submit/add a new guestbook entry. You'll quickly discover that the entry will not be visible on the guestbook page. So, where did it go? At this point the object that contains the entry is in an unpublished state. It is waiting for an approval by the administrator user.

## Approving entries

At this point, we have an unpublished object that awaits approval. Let's check it out and approve it.

Make sure that you're logged into the administration interface.

Click the "Personal" tab.

The administrator user's personal page will appear.

Click on "Collaboration" (from within the menu on the left hand side).  
Click on the item that says "New guestbook entry awaits your approval".

At this point, you should be looking at a page that resembles the following screenshot:

This is the current collaboration frontend of the eZ publish administration interface. It can be used to review/approve/deny/etc. objects, collaborate with other users and so on. Actually, there is a lot of functionality in here.

Press the "Approve" button. The entry will be taken to a temporary publishing queue. If you surf/refresh the TSCM guestbook page again, you will still not be able to see the new (by now approved) guestbook entry. The queue must first be flushed/processed. The following section explains how to do this.

## The "runcronjob" script

The execution/processing of workflows, notifications and other tasks that should be run periodically must be carried out by an external (non-user/web-dependant) mechanism. This is the job of the "runcronjobs.php" file that resides inside the root of the eZ publish directory. This file takes care of processing the mentioned things and it must be executed periodically. The most common way to do this is to set up a cronjob that runs every 5 minutes or so. The script can also be executed manually.

Let's keep things at a simple level and run the "runcronjobs.php" file manually. You can do this easily by executing the script from within a shell using PHP:

```
[path_to_dir_containing_php]/php runcronjobs.php
```

If "php" is available directly (not in the path environment variable), replace the "[path\_to\_dir\_containing\_php]" with the path to the directory that contains the PHP executable. On most UNIX/Linux systems the path is "/usr/bin/". Hint: use the "whereis" command to locate the PHP executable if you're unsure about the location. Windows users can use the built-in search utility to search for "php.exe".

Once the "runcronjobs.php" script is executed, the pending/approved object(s) will be published. Try to surf the TSCM guestbook page again and you'll see that the new (approved) guestbook entry now appears in the list of entries.

## The links page

The links page

The links part of the site will basically be a page that displays a collection of links that we wish to make available for our visitors. This will not be just a boring list of links. First of all, we'll make use of the CMS engine of eZ publish to make a simple category-structure. Secondly, we'll create a neat template that displays the links section of the content node tree in a fancy tree-style view. The following mock-up shows this:

```
-----  
| Banner: THE SCANDINAVIAN CHECK MATES  
-----
```

LINKS

```
(+)-Chess sites  
  
(-)-Educational sites  
  |-Berkeley  
  |-Stanford  
  |-University of Oslo  
  
(+)-Local sites
```

## Adding content

In order to be able to test the "Links" page properly, we'll need to add some content. Use the administration interface to create a folder called "Links" (within the root of the content node tree). Feel free to add the following description for this folder:

*The links folder contains all the links that we wish to share with the rest of the world. Use the tree-menu on the left hand side to navigate the links collection. Each and every link belongs to at least one category. The categories are folders. Click on a folder to expand/view it's contents.*

Don't forget to assign the "Links" folder to the correct/corresponding section (the "Links section"). As before, the section will be used in conjunction with the template override. Now, create a couple of sub-folders within the "Links" folder. The sub-folders will act as categories. Feel free to create the following sub-folders:

**Name:**        **Description:**

Chess sites    This folder contains links to a couple of very popular chess sites on the internet.

Educational sites    This folder contains links to popular academic institutions. Stay in school; don't do drugs!

Local sites This folder contains links to some local sites. Warning! Local sites tend to be boring and lame.

News sites This folder contains links to news sites. Don't believe everything you read. Politicians just love to brainwash the masses.

Each link will be put inside one of the sub-folders. Use the built-in "Link" content class to create a couple of nodes containing link objects. Make sure that you type something into the "Title" and the "URL" fields (in this particular case, the rest of the fields can be safely ignored). Feel free to use the following example data:

**Chess sites:**

Chess Doctor - <http://www.chessdoctor.com>

Chess King - <http://www.chessking.com>

Chess Kit - <http://www.chesskit.com>

Test Your Chess - <http://www.testyourchess.com>

**Educational sites:**

Berkeley - <http://www.berkeley.edu>

Stanford - <http://www.stanford.edu>

University of Twente - <http://www.utwente.nl>

University of Oslo - <http://www.uio.no>

**Local sites:**

eZ systems - <http://www.ez.no>

Skien Kommune - <http://www.skien.no>

Telemark Newspaper - <http://www.ta.no>

Varden Newspaper - <http://www.varden.no>

**News sites:**

BBC News - <http://www.bbcnews.co.uk>

Sky News - <http://www.sky.com/skynews/home>

Slashdot - <http://www.slashdot.org>

The Register - <http://www.theregister.co.uk>

## Creating the template

Make sure that the "Links" link from within the main menu actually works (edit the pagelayout template file and fix it). Now, if you attempt to access the link, you'll probably be presented with a page that looks something like this:

If you have been paying attention during the earlier stages of the tutorial, you should be familiar with the stuff that comes next. What you're looking at is the default/standard full view template. The next natural step is to change this. Use the administration interface to create a template override. If you're still unsure about this procedure, refer to the following hints:

Override this file: "/node/view/full.tpl"

Name of new template: "full\_view\_links\_folder.tpl"

Matching keys: folder class, links section

Base on: empty file

Now that we've created the override, let's start thinking about the template itself. On the left hand side, we could display a list of the sub-folders that are inside the main "Links" folder. When a sub-folder is accessed (the user clicks on it within the browser), eZ publish should list the sub-folder's contents. In this case, the sub-folders contain hyperlinks. Each of these should be presented as a link. When clicked/accessed, the browser should follow the link by opening it in a new window/tab. On the right hand side, we could for example display the name and the description of the folder that was accessed. All this may sound a bit difficult at first. However, it is actually fairly simple. We'll put it together step by step, one piece at a time.

We'll need a bit of CSS, so let's start by adding the following lines to the "tscm.css" file:

```
.links
{
    font-size: 80%;
    vertical-align: top;
    width: 200px;
}
```

Now, edit the template file that was created when you added the template override (should be "design/tscm/override/templates/full\_view\_links\_class.tpl") and put in the following lines:

```
<div class="pagetitle">
Links
</div>
<table>
  <tr>
    <td class="links">
      { * Include the tree menu. * }
      {include uri="design:links_tree_menu.tpl"}
    </td>
    <td valign="top">
      { * Display the folder's name. * }
      { $node.name }
      <br />
      <br />
      { * Display the description of the folder. * }
      {attribute_view_gui
attribute=$node.object.data_map.description}
      </td>
  </tr>
</table>
```

As you can see, we are splitting the page into two parts (vertical split) using a table. The left portion will be used to display the tree menu, the right portion will be used to display information

about the folder that is being accessed. When the "Links" link is accessed (from within the main menu), the right portion will automatically display its name and description (the "Links" folder is simply just another folder from which we can extract the name and the description fields). If you attempt to browse the "Links" page at this point, eZ publish will generate something that looks like this:

## Displaying the sub-folders

The `{include uri="design:links_tree_menu.tpl"}` line within the template code presented in the previous section instructs eZ publish to include a file called "links\_tree\_menu.tpl". The "design:" prefix specifically states that it should look inside the "templates" and the "override/templates" sub-folders of the current (tscm) design folder. Obviously, eZ publish will first try to include the "design/tscm/templates/links\_tree\_menu.tpl" file. Create this file and attempt to write a piece of template code that *always* loops through and displays the sub-folders of the "Links" folder. You'll need to acquire and specify the identification number of the "Links"-node. The folders should be displayed as links. A link should point to the full view of the node that contains the actual folder object. Feel free to peek at/use the following template code:

```
{* Loop through all sub-folders within the "Links" folder. *}
{section loop=fetch( content,
                    list,
                    hash( parent_node_id,    __NODE_ID_OF_LINKS__,
                          class_filter_type, include,
                          class_filter_array, array( 'folder' ),
                          sort_by,          $node.sort_array ) ) }

    {* Display the folder-name as a link to a full view of the node. *}

    <a href={concat( "/content/view/full/", $:item.node_id, "/"
)|ezurl}>
        { $:item.name }
    </a>

    <br />

{* End of sub-folder loop. *}
{/section}
```

If you're using the code that was presented above, make sure that you replace the "\_\_NODE\_ID\_OF\_LINKS\_\_" part with the identification number of the "Links"-folder node. Notice that we're using the fetch function directly inside a section (without doing a "let" first). Refresh the links page. At this point, you should be looking at a page that looks something like this:

When you click on the name of the different folders, the template will display information about the folder that was accessed (on the right hand side). However, regardless of the folder that was accessed, the template will always display the entire list of sub-folders within the "Links" folder. This happens because we hardcoded the identification number of the

"Links" node.

## Displaying the contents of a folder

Whenever a folder is accessed, its contents should be listed directly underneath the name of the folder. This is similar to the visual navigation of a filesystem through a graphical user interface. To achieve this, we must first add some sort of logic that "detects" the folder that is being accessed. We must check if the identification number of the node being listed matches the identification number of the node that is being accessed. We'll use the "section" function combined with the "show" parameter to create an IF-THEN-ELSE like mechanism:

```
{* Check if this node(folder) is the one that was accessed: *}
{section show=eq( $:item.node_id, $node.node_id ) }

    TEST<br />

{* End of if-section. *}
{/section}
```

When a folder is accessed (clicked), this code will print a "TEST" string directly underneath the target folder.

The "\$:item.node\_id" variable contains the identification number of the node that is being listed. The "\$node.node\_id" variable contains the identification number of the node that is being accessed. The "eq" operator is used to compare the two identification numbers. This operator returns true if the numbers are equal (false otherwise). The boolean output from the eq operator is handled by the "show" parameter of the section function. If the show parameter is set to true, the template parser will process the section. If the show parameter is set to false, the section will be skipped. Insert these lines right before the "{\* End of sub-folder loop. \*}" comment of the code within the "links\_tree\_menu.tpl" file. At this point, the "links\_tree\_menu.tpl" file should contain the following lines:

```
{* Loop through all sub-folders within the "Links" folder. *}
{section loop=fetch( content,
                    list,
                    hash( parent_node_id,    __NODE_ID_OF_LINKS__,
                          class_filter_type, include,
                          class_filter_array, array( 'folder' ),
                          sort_by,          $node.sort_array ) ) }

    {* Display the folder-name as a link to a full view of the node. *}

        <a href={concat( "/content/view/full/", $:item.node_id, "/"
)|ezurl}>
            { $:item.name }
        </a>
```

```

<br />

{* Check if this node(folder) is the one that was accessed: *}
{section show=eq( $:item.node_id, $node.node_id )}

TEST
<br />

{* End of if/else structure. *}
{/section}

{* End of sub-folder loop. *}
{/section}

```

Refresh the "Links" page and access/click on the different folders. The "TEST" string will magically appear beneath the folder that was accessed/clicked. Instead of outputting "TEST", we should be looking at the contents of the folder that was accessed. We'll simply loop through and display the contents of the folder. The following template code takes care of this:

```

{* Loop through all the nodes(links) within this folder. *}
{section loop=fetch( content,
                    list,
                    hash( parent_node_id,    $:item.node_id,
                          class_filter_type, include,
                          class_filter_array, array( 'link' ),
                          sort_by,          $:item.sort_array ) ) }

    {* Display the name of each link as a link. *}
    <a href={$item.object.data_map.link.content} target="_blank">
        {$:item.name}
    </a>
    <br />

{* End of link-list loop. *}
{/section}

```

Simply replace the "TEST" string (and the line-break) with the code that was presented above. Refresh the "Links" page and try to click on one of the folders again. The folder's contents should be listed. The following screenshot shows what happens when the "Educational sites" folder is clicked:

## Creating a tree-style appearance

What we have done so far is that we've created a dynamic list. This list can be easily altered so that it appears like a tree-style view. In order to do this, we will add some indentation and a fancy set of icons.

Download the following images and put them inside the "design/tscm/images/" directory:



[folder\\_closed.png](#)  
[folder\\_open.png](#)  
[link.png](#)

The "folder\_open" icon should be displayed in front of the folder that is being accessed. The "folder\_closed" icon should be displayed in front of all the other folders (the ones that are not being accessed). The "link.png" should be displayed in front of the links. In addition, the links should be slightly indented. All this can be added within a matter of seconds by slightly altering the "links\_tree\_menu.tpl" file. At this point, this template file should look something like this:

```
{* Loop through all sub-folders within the "Links" folder. *}
{section loop=fetch( content,
                    list,
                    hash( parent_node_id,    __NODE_ID_OF_LINKS__,
                          class_filter_type, include,
                          class_filter_array, array( 'folder' ),
                          sort_by,          $node.sort_array ) ) }

    {* Check if this node(folder) is the one that was accessed: *}
    {section show=eq( $:item.node_id, $node.node_id )}

        {* Display the folder as "open". *}

        <a href={concat( "/content/view/full/", $:item.node_id, "/"
)|ezurl}>
            <img src={"folder_open.png"|ezimage} alt="" />
        { $:item.name }
        </a>

        <br />

        {* Loop through all the nodes(links) within this folder. *}
        {section loop=fetch( content,
                            list,
                            hash( parent_node_id,    $:item.node_id,
                                  class_filter_type, include,
                                  class_filter_array, array( 'link' ),
                                  sort_by,
                            $:item.sort_array ) ) }

            {* Indent the stuff that comes next (creates a sub-level
effect). *}

            {
                {* Display the name of each link as a link. *}
                <a href="{ $:item.object.data_map.link.content }"
target="_blank">
                    <img src={"link.png"|ezimage} alt="" /> { $:item.name }
                </a>
                <br />
            }

            {* End of link-list loop. *}
            {/section}

            {* Else: this is not the node(folder) that was accessed. *}
            {section-else}

                {* Display the folder as "closed" (and don't list the
contents). *}
            }
        }
    }
}
```

```

) |ezurl}>
    <a href={concat( "/content/view/full/", $:item.node_id, "/"
{$:item.name}
    <img src={"folder_closed.png"|ezimage} alt="" />
    </a>

    <br />

    { * End of if/else structure. * }
    {/section}

{ * End of sub-folder loop. * }
{/section}

```

As you can see, we've added the icons and indented the links. In addition, we've duplicated the code that displays the name (and the icon) of the folder that is being listed by the outer loop. This is done in order to be able to display different icons depending on the state of the folder (the "folder\_open" icon will be displayed adjacent the folder being accessed, the "folder\_closed" icon will be displayed next to all other folders). The "section-else" statement acts as an ELSE branch. At this point, when the the "Educational sites" is clicked, the browser should display something that resembles the following screenshot:

That's it. We've created a simple/generic tree navigator that could be easily altered to function in other kind of contexts.

## Same content at different locations

Two of the links that were put inside the "Local sites" folder are links to the websites of some local newspapers. It would be nice to have these available under the "News sites" folder. This can be easily achieved without duplicating content. Because of the unique structure of eZ publish, a node can appear at several locations within the content node tree. The following list of steps explain how to make the "Telemark Newspaper" node to appear both in the "Local sites" and the "News sites" folder.

Make sure that you're logged into the administration interface.

Navigate the content tree and find the node that is titled "Telemark Newspaper".

Edit the node.

(Click on the nodes name and then the "Edit" button.)

Click the "Add locations" button.

A navigation page will appear.

Use the navigator to locate and select the "News sites" folder.

Click the "Select" button.

If you browse the "Links" part of the site and check out the "Local sites" and then the "News sites" folder, you'll see that "Telemark Newspaper" appears under both categories. This concludes the first part of the "Building an eZ publish site" tutorial.

