

# TUTORIAL - Personalizzare i template di eZpublish

Autore: Maurizio Betti <m.betti@arsweb.it>

Nell'articolo che segue vedremo come personalizzare in modo **rapido, semplice ed efficace** i template di un sito in eZpublish. Partiremo da una installazione predefinita di eZ (corporate design, dal wizard di installazione) e da un template html statico già predisposto che dovrà essere implementato su eZ per renderlo dinamico.

## PREMESSA

Come mi sembra di capire dalle questioni poste nel forum molte persone cercano di modificare i template predefiniti che vengono creati dal wizard di installazione. Tale procedimento oltre che molto dispendioso (è più facile dinamizzare un template statico intero che cercare di inserire codice personalizzato in template già esistenti e strutturati per altro) risulta anche molto difficile per due motivi:

1. I template creati dal wizard di installazione utilizzano alcune funzioni che nelle recenti versioni di eZ publish sono state DEPRECATED (proprio perché troppo poco intuitive :-)) e di fatto, su queste non esiste alcuna documentazione. Un classico esempio è la funzione "section" ormai non più utilizzata.
2. I template creati dal wizard seguono una logica estremamente generica e parametrica cercando di astrarre il più possibile. Questo ovviamente rende molto difficoltosa l'operazione di customizzazione con codice specifico.

Per questo motivo l'approccio della customizzazione sull'esistente è vivamente sconsigliato (soprattutto per chi non ha perfettamente chiari i meccanismi alla base di eZ).

In questo tutorial seguiremo un'altro approccio, **estremamente semplice e lineare**, che prevede di creare il proprio sito rendendo dinamico un template html statico pre-esistente. Man mano che verranno affrontate le varie problematiche di questo processo si cercherà di evidenziare, accanto alla soluzione pratica, anche il meccanismo generale che sta dietro la soluzione stessa.

E' da notare che il linguaggio di templating di eZ è costituito da relativamente poche funzioni facili da apprendere per chi abbia un minimo di base di programmazione. Non è richiesta invece alcuna conoscenza di php.

## PASSO 1 - Note introduttive

In questo tutorial daremo per scontato l'aver completato correttamente una installazione di eZpublish attraverso il wizard on-line. Per quello che ci proponiamo di fare una scelta di template vale l'altra. Per maggiore comodità tuttavia utilizzeremo l'installazione " **corporate**" selezionata dal wizard della versione 3.8.3 di eZpublish.

Nota: eZpublish permette di modificare i template grafici anche online. Tuttavia per maggiore comodità e rapidità di sviluppo in questo tutorial ci avvarremo di un accesso ftp che ci consenta di muoverci rapidamente sui vari file (.tpl) e soprattutto che consenta l'utilizzo di un editor che supporti il coloring del codice di eZ. Per questo raccomando Jedit (<http://www.jedit.org/>), un ottimo editor per i file .tpl grazie all'utilizzo della relativa estensione ([http://ez.no/community/contribs/3rd\\_party/ez\\_tpl\\_highlight\\_support\\_edit\\_mode\\_for\\_jedit](http://ez.no/community/contribs/3rd_party/ez_tpl_highlight_support_edit_mode_for_jedit)).

Una ulteriore nota prima di iniziare a lavorare: tenete sempre ben presente che all'interno di una installazione di default di eZpublish ci sono due directory molto importanti:

- /design: contiene tutti i file .tpl relativi ai template delle pagine
- settings: contiene tutte le informazioni di configurazione del sistema (tra cui anche quelle che indicano in che modo applicare i template alla varie classi)

**Nota:** per maggiore comodità in tutti i paragrafi che seguono i path vengono indicati in modo assoluto rispetto alla cartella di installazione di eZpublish. Se allora la mia istanza sarà installata sotto la cartella /data/www/myezpublish/ questa diventa /. Per indicare ad esempio la cartella "bin" scriverò direttamente /bin.

Tenendo ben presente queste 2 cose cominciamo a vedere come personalizzare l'home page

## PASSO 2 - Personalizzazione dell'home page

Cominciamo quindi ad addentrarci nella personalizzazione dei template in eZ. Per maggiore comodità andremo ad utilizzare un semplice html statico che potete vedere all'url <http://www.oswd.org/design/download/id/2199> e scaricare qui: <http://www.oswd.org/design/download/id/2199>.

Il template statico da rendere dinamico si presenta come segue:

In questo passo vedremo allora come "importare" questo template dinamico in eZpublish in particolare attraverso i seguenti passaggi successivi:

1. Impostare il template generale (pagelayout.tpl)
2. Inserire titolo, citazione (in alto a destra) ed immagine
3. Rendere dinamico il menu pescando dalle cartelle inserite in eZpublish da interfaccia amministrativa
4. Inserire elenco di news e links (rispettivamente a sinistra e a destra)
5. Inserire il corpo centrale (ed introduzione al concetto di **vista** in eZpublish)

Vediamo ad uno ad uno

### 1. Impostare il template generale (pagelayout.tpl)

Nel caso pi<sup>1</sup> classico un sito web  $\tilde{}$  costituito da una "gabbia grafica" generale che rimane pressoch $\tilde{}$  costante attraverso tutte le pagine, ed un contenuto interno che cambia a seconda della pagina in cui mi trovo. eZpublish gestisce questa gabbia grafica attraverso un particolare file chiamato **pagelayout.tpl**. Per creare un proprio pagelayout personalizzato  $\tilde{}$  sufficiente creare un file vuoto di nome **pagelayout.tpl** che sovrascriva quello generico di eZpublish. Per fare questo andiamo nella cartella **/design/corporate\_site/override/templates** e creiamo il file pagelayout.tpl.

Attenzione: **corporate\_site**  $\tilde{}$  il nome del "site access" utilizzato di default da eZpublish quando importa da wizard il modello "corporate". Importando un sito di tipo "shop" si avr $\tilde{}$  un site access di nome "shop\_site".

Attenzione: perch $\tilde{}$  **override**?. eZpublish prevede un meccanismo "a cascata" per la gestione dei template. In pratica quando gli viene richiesta la visualizzazione di un nodo prima controllo se per quel nodo  $\tilde{}$  stato definito un template specifico (di override appunto), se non lo trova applica il template standard. Per questo buona norma, quando si deve modificare un template di default (sotto **/design/standard**), non apportare le modifiche direttamente sul file interessato, ma creare un override. Vedremo come.

Intanto abbiamo creato un override del pagelayout.tpl. Per vedere se funziona Ã sufficiente, **dopo aver svuotato la cache**, andare a vedere la pagina principale del sito (il link pubblico che appare nel wizard una volta completata l'installazione). Se abbiamo applicato correttamente, al posto della pagina di avvio standard dovremmo vedere... **una pagina bianca!** Funziona? Spero di sÌ. Nel caso provate a rileggere attentamente e a controllare i permessi sul filesystem (se lavorate in ambiente linux Ã sufficiente dare il comando ./bin/modfix.sh).

## 2. Inserire titolo, citazione (in alto a destra) ed immagine

Al passo successivo dovremo copiare il codice del template statico all'interno del file pagelayout.tpl appena creato. Apriamo dunque il nostro file **pagelayout.tpl** ed inseriamo il nostro codice statico:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta name="description" content="_your description goes here_" />
<meta name="keywords" content="_your,keywords,goes,here_" />
<meta name="author" content="_your name goes here_ / Original design:
Andreas Viklund - http://andreasviklund.com/" />
<link rel="stylesheet" type="text/css" href="andreas01.css"
media="screen" title="andreas01 (screen)" />
<link rel="stylesheet" type="text/css" href="print.css" media="print" />
<title>andreas01</title>
</head>

<body><div id="wrap">

<div id="header">
<h1>andreas01</h1>

<p><strong>"I can see you fly. You are an angel with wings, high above
the ground!"</strong><br />(traditional haiku poem)</p>
</div>



<div id="avmenu">
<h2 class="hide">Menu:</h2>
<ul>
<li><a href="#">Welcome!</a></li>
<li><a href="#">Current events</a></li>
<li><a href="#">Downloads</a></li>

<li><a href="#">Art gallery</a></li>
<li><a href="#">Collections</a></li>
<li><a href="#">Litterature</a></li>
<li><a href="#">Newsletter</a></li>
```

</ul>

<div class="announce">

<h3>Latest news:</h3>

<p><strong>June 25, 2005:</strong><br />

New design and layout finished and uploaded to OSWD. Since it is my first contribution to this site, it has been given the title "andreas01".</p>

<p class="textright"><a href="index.html">Read more...</a></p>

</div>

</div>

<div id="extras">

<h3>More info:</h3>

<p>This is the third column, which can be used in many different ways. For example, it can be used for comments, site news, external links, ads or for more navigation links. It is all up to you!</p>

<h3>Links:</h3>

<p>- <a href="http://andreasviklund.com/">My website</a><br />

- <a href="http://oswd.org/">OSWD.org</a><br />

- <a href="http://validator.w3.org/check/referer">Valid XHTML</a><br />

- <a href="http://jigsaw.w3.org/css-validator/check/referer">Valid CSS</a></p>

<p class="small">Version: 1.0<br />(July 25, 2005)</p>

</div>

<div id="content">

<h2>Welcome to andreas01!</h2>

<p>This is a very simple (but also very useful) standard web layout: a fixed-width 3-column page with header and footer. There are thousands of variations of this layout, and this is no perfect solution. But it should be a great starting layout that you can customize in many ways and easily give a unique touch to. The layout is made of valid XHTML 1.0 Strict and CSS only, and it has no tables. If you remove all CSS, the page still works great in text mode (I've tested it on my old mobile phone, and it works like a charm).</p>

<p>The layout features one big image (front.jpg) which you can easily replace with your own header. The default image shows a snowy street in Porjus (northern Sweden), where I come from. The haiku can easily be replaced with an ad banner. There are also CSS classes for aligning text paragraphs and images.</p>

<p>This layout also has a print stylesheet, that you can customize in any way you want to. Per default, navigation menus and images are removed, and only the main content text is printed. <a href="index2.html">Look here</a> for an example of how the print layout looks, or print this page to see it in reality.</p>

<h3>The menubar</h3><p>The main navigation menu is an unsorted list, where each list item has been styled to be a button. There are lots of ways to style lists for navigation, but I've kept things simple for this layout. Adding "that little extra" is up to you! </p>

```
<p>If you like this layout and would like to use it in any way, you are free to do so. This is an <a href="http://oswd.org">open source web design</a>, and all I ask for is that you leave the "Design by Andreas Viklund" link in the footer of the site. If you would like to remove that line, or if you would like professional help with anything related to this layout (such as custom design, branding, scripting or programming), please contact me <a href="http://oswd.org/email.phtml?user=Andreas">through OSWD.org</a> or through my website for more information.</p>
```

```
<p>Good luck with your new website!</p>
</div>
```

```
<div id="footer">
Copyright &copy; 2005 (_your name_). Design by <a href="http://andreasviklund.com">Andreas Viklund</a>.
</div>
```

```
</div>
</body>
</html>
```

Cominciamo quindi, pezzo per pezzo a "dinamizzare" questo template statico.

La prima cosa da sistemare sono i collegamenti ai CSS: salvate i file **andreas01.css** e **print.css** nella cartella **/design/corporate\_site/stylesheets**. Cambiate quindi nel file pagelayout.tpl le righe di collegamento ai css in questo modo:

```
<link rel="stylesheet" type="text/css" href="andreas01.css"
media="screen" title="andreas01 (screen)" />
<link rel="stylesheet" type="text/css" href="print.css" media="print" />
```

DIVENTA

```
<link rel="stylesheet" type="text/css"
href={'stylesheets/andreas01.css'|ezdesign} media="screen"
title="andreas01 (screen)" />
<link rel="stylesheet" type="text/css"
href={'stylesheets/print.css'|ezdesign} media="print" />
```

Come potete notare abbiamo sostituito il nome semplice del css ( **attenzione anche le virgolette scompaiono!**) con il seguente:

```
{ 'stylesheets/<nomedelcss>.css' | ezdesign}
```

richiamando la funzione **ezdesign** che ricostruisce il path relativo a partire dalla nostra cartella design. Dando in input alla funzione la cartella di riferimento "stylesheets" ed il nome del file "<nomedelcss>.css" abbiamo correttamente collegato il file css dal file html. Il risultato ( **ricordatevi di svuotare la cache**) dovrebbe essere quello del template statico di partenza senza le immagini.

Allo stesso modo andiamo a modificare l'html per far puntare in modo corretto le immagini. Copiate all'interno della cartella **/design/corporate\_site/images** tutte le immagini che dovete richiamare da template (i file **front.jpg, test.jpg, bg.gif**) e modificate i path relativi nell'html come da seguente (l'esempio Ã" riportato per l'immagine dell'intestazione), attenzione che anche in questo caso spariscono le virgolette:

```

```

DIVENTA

```
<img id="frontphoto" src={'front.jpg' | ezimage} width="760" height="175" alt="" />
```

La funzione **ezimage** ricostruisce il path della cartella images per il file dato in input. In questo modo. Salvando il tutto e svuotando nuovamente la cache dovremmo ottenere il template statico di partenza (replicate per tutte le immagini presenti nel template)!

Nota: eventuali immagini che puntano dal file .css dovranno essere inserite utilizzando il percorso relativo a partire dalla cartella stylesheets, quindi: **"../images/<nomimmagine>"**

### 3. Rendere dinamico il menu principale

Eccoci dunque al passo successivo che ci permette di esplorare una delle funzioni probabilmente piÃ¹ utilizzate di eZpublish nei template. La funzione **fecth** (dall'inglese *to fetch = prendere*) serve a prelevare un elenco di contenuti memorizzandoli in una lista (array). Il tipo di contenuti puÃ² essere di vario tipo (sezioni, classi, nodi, oggetti). Nel caso piÃ¹ comune (come nel nostro) viene utilizzata per "prendere" dall'albero dei contenuti uno specifico insieme di nodi (a

seconda dei parametri passati al momento dell'utilizzo) e visualizzarli nel sito pubblico. Ma facciamo subito un esempio chiarificatore.

Entrare nell'area di amministrazione ed **eliminate tutti i nodi presenti dall'albero dei contenuti** (tranne ovviamente il nodo di root).

Create sempre nell'area di amministrazione, a partire dalla root del sito, un certo numero di cartelle a piacere, ciascuna delle quali rappresenterà una voce di menu principale. Avremo quindi per esempio le cartelle: info, chi siamo, prodotti, novità, contatti. Questi saranno i nodi che dovranno comparire come menu principale a sinistra nel template del nostro sito.

A questo punto dobbiamo far apparire questo elenco di cartelle come menu principale al posto di quello statico presente nel template principale. Aprite nuovamente il file pagelayout.tpl, individuate la porzione di codice che contiene il menu ovvero:

```
<h2 class="hide">Menu:</h2>
<ul>
<li><a href="#">Welcome!</a></li>
<li><a href="#">Current events</a></li>
<li><a href="#">Downloads</a></li>

<li><a href="#">Art gallery</a></li>
<li><a href="#">Collections</a></li>
<li><a href="#">Litterature</a></li>
<li><a href="#">Newsletter</a></li>
</ul>
```

Adesso: ricaveremo dal pagelayout.tpl l'elenco delle cartelle appena inserite con la funzione fetch e con un loop sull'array restituito da tale funzione andremo a creare il nostro menu dinamico. Vediamo come:

```
{* ricavo un elenco semplice di nodi figli del nodo principale che ha
sempre id=2 e
memorizzo tale elenco nella variabile $items*}
{def $items=fetch( 'content', 'list',hash( 'parent_node_id', 2))}

<h2 class="hide">Menu:</h2>
<ul>
{* scorro la variabile $items e ad ogni iterazione stampo il nome del
```

```
nodo ed il collegamento a questo *}
{foreach $items as $item}
  <li><a href={$item.url_alias|ezurl}>{$item.name|wash}</a></li>
{/foreach}
</ul>
```

Salviamo il pagelayout.tpl, svuotiamo la cache e, se abbiamo fatto tutto correttamente, dovremo vedere l'elenco statico di prima scomparire per visualizzare lo stesso menu ma con il nome delle nostre cartelle. Superfluo aggiungere che a questo punto tutti i contenuti che aggiungeremo direttamente sotto la root saranno visualizzati nella colonna di sinistra.

Ricapitoliamo brevemente le funzionalità usate per generare questo menu ed il loro utilizzo in modo più generale:

- definizione di variabile: la sintassi **{def \$variabile=<valore>}**. Nel caso dovessimo rivalorizzare la variabile si userà invece la sintassi **{set \$variabile=<valore>}** **funzione fetch**: come anticipato la funzione **fetch** è utile a ricavare una collezione di nodi a partire da una serie di parametri di ingresso. L'uso fatto nell'esempio è molto semplice (elenco semplice dei nodi figli di un nodo padre), ma la funzione permette di decidere quanti nodi recuperare, fino a che profondità, quale tipo di nodi (di quale/quali specifica/e classi), con quale particolare attributo, con quale ordine, etc. La funzione fetch è molto ricca e molto potente e viene utilizzata nei casi più variegati. Vi suggerisco pertanto di [leggere attentamente la documentazione](#) relativa e postare eventuali domande.
  - foreach: il costrutto **foreach** è anche tra quelli più spesso utilizzati e serve a scorrere un array definito in precedenza con la funzione fetch. La sintassi del foreach valorizza ad ogni interazione una variabile che conterrà tutti i dati utili per quella interazione (nell'esempio è la variabile \$node).
  - accesso alle proprietà del nodo: per visualizzare il nome del nodo ed il suo puntamento abbiamo fatto accesso nell'esempio a due semplici proprietà. Per avere l'elenco completo delle proprietà di un nodo potete fare [riferimento a questo url](#). Nell'esempio la proprietà **.url\_alias** e **.name** ci permette di ricavare il collegamento ed il nome del nodo di riferimento.

NOTA: un semplice ma efficace modo per "esplorare" il contenuto di un oggetto (classe, nodo, array, e quanto altro) è quello di utilizzare la sintassi **{\$variabiledaesplorare|attribute(show)}**. In questo modo verranno stampate a video l'elenco completo delle proprietà dell'oggetto di riferimento assieme al loro valore.

#### 4. Inserire l'elenco di news e links

Proseguendo nel voler rendere interamente dinamico il template statico di cui sopra, ci imbattiamo nella necessità di dover pescare gli elenchi di news e links rispettivamente a destra e sinistra del template principale. Lo scopo di questa operazione è evidente: spesso l'utente ha la necessità di gestire un elenco di news che venga automaticamente visualizzato in bella evidenza nel sito online. Vediamo in prima battuta come pescare l'elenco di news, ordinarlo per data di pubblicazione e visualizzarlo in home page. Seguirà una breve indicazione di come

ricavare e visualizzare l'elenco dei links.

La prima cosa da fare Ã" ovviamente quella di creare dall'area amministrativa una cartella con il nome "News" in cui andremo ad inserire tutte le news del sito. Creiamo anche due o tre news (per questo scopo va benissimo la classe "article" o una qualunque altra classe con nome, data della news (creiamo per l'occasione un attributo di nome "date"), descrizione breve, descrizione). Nell'esempio che useremo la classe di riferimento per le news sarÃ la classe con identificatore "article"

Posizioniamoci quindi sul codice relativo alle news:

```
<h3>Latest news:</h3>
<p><strong>June 25, 2005:</strong><br />
New design and layout finished and uploaded to OSWD. Since it is my
first contribution to this site, it has been given the title
"andreas01".</p><p class="textright"><a href="index.html">Read
more...</a></p>
```

e sostituiamolo con il seguente (a seguire la spiegazione delle funzionalitÃ utilizzate)

```
<h3>Latest news:</h3>
{* ricavo elenco delle news "complesso" *}
{def $news=fetch( 'content', 'tree', hash( 'parent_node_id', 2,
'sort_by', array('attribute', false(), 'article/date'),
'class_filter_type', 'include', 'class_filter_array', array('article')
)}}
{* scorro la variabile $news e ad ogni iterazione visualizzo gli
elementi della singola novitÃ *}
{foreach $news as $new}

<p><strong>{$new.object.data_map.date.timestamp|l10n('shortdate'
ime')}</strong><br />
{attribute_view_gui
attribute=$new.object.data_map.short_description}</p>
<p class="textright"><a href={$new.url_alias|ezurl}>Read
more...</a></p>
{/foreach}
```

Come prima ricordiamoci di salvare il pagelayout.tpl e svuotare la cache. Aggiornando il sito pubblico dovrebbero apparire l'elenco delle news inserite dall'area di amministrazione. Alcune note riguardo le funzioni utilizzate:

- Anche in questo caso abbiamo usato la funzione **fetch**, se pur in modo leggermente diverso. Anzitutto l'abbiamo richiamata utilizzando il valore **tree** (secondo parametro). Utilizzando **tree** invece che **list** la funzione prende tutti i nodi contenuti a partire dal nodo padre indicato, ma cercando non solo tra i figli ma **fra tutti i discendenti!** In questo modo verranno presi tutti i nodi di tipo article presenti nel sito, indipendentemente dalla loro posizione. Questo metodo ci permette inoltre di non dover specificare nel template un particolare nodo padre per le news che potrebbe anche cambiare in seguito a modifiche nell'albero dei contenuti. Ancora per fetch: in questo caso abbiamo utilizzato la funzione **fetch** in maniera pi complessa di quella vista in precedenza per i menu. Per ricavare le news infatti abbiamo specificato di ricercare solo fra i nodi di classe "article" e di ordinare l'array risultante secondo l'attributo "date" in modo decrescente. Per queste ed altri parametri utilizzabili con la funzione fetch vi invito a leggervi attentamente la documentazione relativa sul sito ufficiale. Accesso alle propriet del nodo: di nuovo utilizziamo la lettura delle propriet del nodo per leggere il valore dell'attributo "date" del nodo e stamparlo a video. La sintassi **\$new.object.data\_map.date.content.timestamp** indica al sistema di leggere la propriet "date" dell'oggetto contenuto nel nodo "new" e di visualizzarne il contenuto ".content" nella forma unix ".timestamp". Tale valore viene quindi passato attraverso il simbolo di "pipe" ("|") alla funzione **l10n** per un rendering ottimale nella forma voluta (in questo caso **shortdatetime** che la visualizza nel formato data e tempo brevi). Per altri rendering della funzione vi invito a leggere la [relativa documentazione](#). **NOTA BENE: IN GENERALE** per leggere il valore inserito nell'attributo di un oggetto  possibile utilizzare la sintassi vista del tipo `<nodo>.object.data_map.<identificatore_attributo>.content.[<formato>]` (l'ultima voce  facoltativa e dipende dal tipo do attributo). E' possibile visualizzare il tipo di output prodotto dalla funzione ".content" leggendo la [documentazione relativa al tipo di attributo](#) su cui si sta operando. Trovate l'elenco di attributi (datatypes) possibili a [questo indirizzo](#). Selezionate l'attributo che vi interessa e scorrete la pagina fino alla voce **Raw output**, quindi seguite il link relativo all'oggetto ritornato dalla funzione .content. Nel caso di un attributo (datatype) di tipo Time ( il nostro esempio) l'output ritornato  un oggetto "[eztime object](#)". Per l'output dei contenuti potete in generale riferirvi a questo [mini-tutorial](#). **{attribute\_view\_gui attribute=\$new.object.data\_map.short\_description}**: questa sintassi fornisce un metodo generico per stampare il contenuto di un attributo senza conoscerne le propriet come indicato sopra. In pratica eZpublish fornisce questa funzione che permette di specificare l'attributo da visualizzare. Vantaggio: l'output  prodotto rapidamente. Svantaggio: molte volte l'output prodotto non  come lo vorremmo visualizzare.

Per generare l' **elenco dei links** sulla destra si procede nello stesso identico modo visto per news, ovviamente modificando la classe filtrata (che sar di tipo link). Il codice riportato  gi personalizzato sul template di esempio:

```
{* Ricavo elenco dei links *}
{def $links=fetch( 'content', 'tree', hash( 'parent_node_id', 2,
'class_filter_type', 'include', 'class_filter_array', array('link'),
'sort_by', array('priority', false()) ) )}{* Visualizzo elenco dei
links solo se l'array ritornato NON  vuoto *}
{if $links}
<h3>Links:</h3>
<p>
    {foreach $links as $link}
```

```

href="{ $link.object.data_map.location.content }">{ $link.name }</a><br/>
    { /foreach }
</p>
{ /if }

```

Unica nota aggiuntiva rispetto all'elenco delle news Ã l'utilizzo della condizione `{if $links}...{/if}` che verifica, prima di stampare l'elenco dei links, che l'array non sia vuoto. Questo metodo Ã molto utile, nel nostro caso, ad evitare che venga stampata l'intestazione `<h3>Links</h3>` se poi l'elenco stesso risulta vuoto.

#### 4. Inserire il corpo centrale (ed introduzione al concetto di vista in eZpublish)

Una breve nota introduttiva: la parte che segue Ã fondamentale per la comprensione dei template in eZpublish e vi consiglio pertanto di leggerla attentamente cercando di comprenderne i concetti spiegati anche oltre gli esempi fatti, prima di procedere nelle sezioni successive.

Fino a questo momento abbiamo lavorato sul file `pagelayout.tpl`. Siamo arrivati tuttavia ad un punto che ci obbliga ad introdurre un concetto che ricorre spessissimo in eZpublish nella costruzione dei template, il concetto di **vista (view)**. Introduciamolo passo per passo.

Cominciamo col dire che abbiamo completato l'implementazione della gabbia grafica che rimane piÃ¹ o meno costante in tutto il sito (nei casi piÃ¹ semplici). Quello che cambia in un sito Ã tipicamente il corpo centrale che dovrÃ visualizzare di volta in volta il dettaglio di una pagina, un elenco di oggetti contenuti nella cartella corrente, e cosÃ¬ via. eZpublish risolve questa necessitÃ in modo semplice, elegante e potente utilizzando il concetto di **vista**.

La **vista** in eZpublish non Ã altro (come suggerisce la parola stessa) una modalitÃ di volta in volta differente di visualizzare lo stesso oggetto in posizioni diverse. Facciamo un semplice esempio: il caso di una **news** (ovvero di un'oggetto di tipo `article` creato nell'area di amministrazione). Per lo stesso oggetto potrÃ avere piÃ¹ viste o visualizzazioni:

1. La news contenuta nell'elenco di destra in home page
2. La news contenuta nell'elenco centrale quando ad esempio si visualizza un archivio delle news
3. Il dettaglio della news quando si clicca su "dettagli"
4. ....

Ad ogni vista di eZpublish Ã associato un particolare template (file .tpl). In generale (ma Ã solo indicativo in quanto le viste possono essere personalizzate) avremo quindi le seguenti associazioni:

1. Elenco nella colonna di destra: template contenuti nella cartella **/design/corporate\_site/override/templates/listitem/<identificatoreclasse>.tpl**
2. Elenco nel corpo centrale: template contenuti nella cartella **/design/corporate\_site/override/templates/line/<identificatoreclasse>.tpl**
3. Vista di dettaglio per il nodo corrente: template contenuti nella cartella **/design/corporate\_site/override/templates/full/<identificatoreclasse>.tpl**

Questo concetto di vista Ã ovviamente applicabile a **tutti gli oggetti** caricati nell'area riservata (cartelle, immagini, links e quanto altro).

Tenendo questa cosa ben in mente sostituiamo la parte centrale del nostro template con la funzione:

```
{ $module_result.content }
```

diventerÃ quindi..

```
<div id="content">  
    { $module_result.content }  
</div>
```

La funzione **{ \$module\_result.content }** visualizza la vista di dettaglio del nodo corrente (quindi il template relativo alla classe corrente contenuto nella cartella **full**).

eZpublish quindi carica il pagelayout.tpl e sostituisce al posto di **{ \$module\_result.content }** il template di tipo **full** definito per il nodo corrente. Per fare questa verifica consideriamo la root del sito. La classe relativa alla root del sito Ã di tipo "folder". Apriamo quindi il template **/design/corporate\_site/override/templates/full/folder.tpl** cancelliamo il contenuto esistente e

scrivamoci un testo qualunque ad esempio: "template vista full per la classe folder".

Salviamo il tutto, svuotiamo la cache e puntiamo il browser alla root del sito (sarÃ  qualcosa del tipo: <http://localhost/miosito/index.php>) e, se abbiamo fatto tutto correttamente dovremo vedere il testo appena inserito nel template che appare nell'home page al posto del precedente!

**Nota:** questo meccanismo vale per TUTTI gli oggetti nell'albero dei contenuti. Se proviamo a visualizzare il dettaglio di un articolo (facendo ad esempio click sulla scritta Read more dell'elenco delle news), eZpublish caricherÃ  la gabbia grafica "incastrando" all'interno la vista di dettaglio per l'articolo che corrisponderÃ  al template **/design/corporate\_site/override/templates/full/article.tpl**.

Vediamo adesso, secondo questa spiegazione e tenendo ben a mente il concetto di **view** (vista) come facciamo a far apparire, secondo il modello statico HTML, un elenco di paragrafi (con o senza foto), preceduti dal campo titolo e descrizione della cartella corrente.

Siccome la root del sito Ã  una cartella, per modificare il contenuto che appare in home page dovemo necessariamente inserire il nostro codice all'interno del template di dettaglio per la classe folder.

Apriamo dunque in modifica il file **/design/corporate\_site/override/templates/full/folder.tpl** cancelliamo il testo creato in precedenza ed inseriamo il seguente:

```
<h2>{$node.name|wash()}</h2>
{attribute_view_gui attribute=$node.object.data_map.description}
```

Salviamo il file, svuotiamo la cache ed aggiorniamo il browser e cosa troviamo? Nella vista del nodo corrente comparirÃ , all'interno della zona centrale del template, il **titolo** e la **descrizione** inseriti da area di amministrazione per la cartella corrente (in questo caso la cartella root dell'home page).

**Complichiamo ulteriormente le cose:** sotto il titolo e la descrizione vogliamo far apparire un **elenco di paragrafi** (con o senza immagine). Come fare? La soluzione piÃ¹ semplice (ma poco corretta) sarebbe quella di inserire tutto l'html all'interno del campo html "descrizione" dell'home page. In questo modo perÃ² costringeremmo l'utente finale ad "inventarsi" volta per volta il layout della pagina. La potenza di eZpublish invece ci permette di creare nuove classi/oggetti personalizzati che l'utente finale puÃ² utilizzare (creare, modificare, eliminare) preoccupandosi solo del contenuto e non della formattazione. Vediamo come.

Creiamo anzitutto una nuova classe di tipo "paragrafo". Per fare questo dovremo:

1. Loggarci nell'area di amministrazione con un account di admin
2. Selezionare dal menu in alto la voce "Impostazioni"
3. Selezionare dal menu laterale la voce "Classi"
4. Dal corpo centrale selezionare il gruppo di classi "Content"
5. Scorrere l'elenco di classi fino in fondo e premere il pulsante "Nuova classe"
6. Assegnare il nome "Paragrafo", l'identificatore "paragrafo" e il parametro del nome dell'oggetto <name>
7. Creiamo quindi 3 attributi: una Linea di testo con nome "Nome" ed identificatore "nome", un campo descrittivo Blocco XML di nome "Descrizione" ed identificatore "descrizione" ed un attributo di tipo immagine con nome "Immagine" ed identificatore "immagine"
8. Salviamo il tutto e torniamo all'elenco precedente.

Alcune **note** in merito alla creazione di nuove classi/oggetti: come avete potuto vedere uno degli aspetti piÃ¹ interessanti di eZpublish Ã¨ proprio quello relativo alla facilitÃ  con cui le classi base (news, faq, gallery, article) possano essere estese con nuovi attributi (ed anche qui la scelta di possibili attributi Ã¨ molto ampia) o addirittura create ex-novo.

Nel nostro esempio non abbiamo fatto altro che creare una nuova semplice classe di tipo paragrafo utilizzando attributi ( **datatypes**) esistenti. Con lo stesso meccanismo Ã¨ possibile perÃ² modellare realtÃ  complesse esattamente come faremo dovendo stendere un diagramma EntitÃ  /Relazioni.

Torniamo a noi. Ritornate, sempre in area di amministrazione, all'albero dei contenuti. Nell'elenco degli oggetti che potete inserire dal menu a tendina troverete adesso anche la nuova classe paragrafo appena creata. Posizionatevi sul nodo di root (cartella) e create tre nuovi paragrafi inserendo, per ciascuno di essi, un titolo, una breve descrizione ed una immagine (potete utilizzare per questo scopo quelli del template statico). I tre paragrafi creati saranno figli del nodo

principale.

Fatto? Quello che ci manca ora Ã di visualizzare i contenuti dei paragrafi inseriti in home page. Si puÃ fare questo in due modi:

1. Richiamando **direttamente** i tre paragrafi ed i loro attributi dal template di dettaglio della cartella
2. Richiamando **indirettamente** i tre paragrafi con il meccanismo delle **view** di eZpublish (che Ã poi lo scopo di questa sezione)

Quale approccio usare? Da un punto di vista del risultato finale i due approcci sono equivalenti. Da un punto di vista dell'eleganza e soprattutto della riusabilitÃ del codice Ã decisamente preferibile utilizzare il secondo approccio. In questo esempio mostreremo entrambe le tecniche (la seconda non Ã altro che una estensione della prima).

Con la **prima tecnica** diretta dovremo compiere i seguenti passaggi:

1. Aprire il file **/design/corporate\_site/override/templates/full/folder.tpl**
2. Posizionarsi al di sotto del codice scritto in precedenza
3. Recuperare l'elenco dei figli di tipo "paragrafo" del nodo corrente
4. Scorrere l'array risultante
5. Stampare a video, per ogni nodo figlio, la serie di attributi Titolo, Descrizione, Immagine

```
{* recupero l'elenco dei paragrafi contenuti nella cartella corrente *}
{set $items=fetch( 'content', 'list', hash( 'parent_node_id',
$node.node_id, 'sort_by', array(array('priority', true))),
'class_filter_type', 'include', 'class_filter_array', array('paragrafo')
) )}

{* verifico se ci sono paragrafi nella qual caso di visualizzo
stampando per ciascuno il contenuto degli attributi *}
{if $items}
  {foreach $items as $item}
    {* per ogni paragrafo stampo i 3 attributi (nome, descrizione,
immagine) *}
    <p>
      {attribute_view_gui attribute=$item.data_map.nome}<br/>
      {attribute_view_gui
attribute=$item.data_map.descrizione}<br/>
      {attribute_view_gui attribute=$item.data_map.immagine}
    </p>
  {/foreach}
```

```
{/if}
```

Con la **seconda tecnica** (indiretta) il codice viene invece modificato in questo modo:

```
{* recupero l'elenco dei paragrafi contenuti nella cartella corrente *}
{set $items=fetch( 'content', 'list', hash( 'parent_node_id',
$node.node_id, 'sort_by', array(array('priority', true())),
'class_filter_type', 'include', 'class_filter_array', array('paragrafo')
) )}{* verifico se ci sono paragrafi nella qual caso di visualizzo
stampando per ciascuno il contenuto degli attributi *}
{if $items}
{foreach $items as $item}
{* per ogni paragrafo stampa NON gli attributi, ma la vista di
tipo "line"*}
{node_view_gui view=line content_node=$item}
{/foreach}
{/if}
```

Cosa Ã" cambiato? Abbiamo appena introdotto nel template il concetto di **vista (view)** che in eZpublish ricorre ovunque. Il codice inserito al posto della stampa dei singoli attributi:

```
{node_view_gui view=line content_node=$item}
```

dice ad eZpublish: "attenzione: invece che stampare direttamente gli attributi per il nodo corrente, visualizza il nodo corrente con la vista di tipo **line**".

Chiaramente a questo punto dobbiamo DEFINIRE in qualche modo tale vista. Come? Molto facilmente con i seguenti passi:

1. Spostiamoci nella cartella Aprire il file **/design/corporate\_site/override/templates/line**

2. Creiamo il file vuoto paragrafo.tpl inserendo all'interno le seguenti:

```
<p>
{attribute_view_gui attribute=$node.data_map.nome}<br/>
{attribute_view_gui attribute=$node.data_map.descrizione}<br/>
```

```
</p> {attribute_view_gui attribute=$node.data_map.immagine}
```

3. Referenziamo la nuova vista nel file override.ini del nostro siteaccess. Apriamo e modifichiamo il file **/settings/siteaccess/<nomesiteaccess>/override.ini.append.php** aggiungendo le seguenti:

```
[line_paragrafo]
Source=node/view/line.tpl
MatchFile=line/paragrafo.tpl
Subdir=templates
Match[class_identifier]=paragrafo
```

4. Salviamo e chiudiamo tutto ricordandoci di svuotare le cache

Se abbiamo operato correttamente vedremo che le due tecniche producono lo **stesso risultato** (vengono stampati tutti i paragrafi contenuti nella cartella corrente) ma con una importante differenza: mentre nel primo caso il codice prodotto deve essere duplicato ogni volta che devo stampare un paragrafo in una certa posizione, nel secondo caso "**incapsulo**" il template, specifico per una vista e per una classe, all'interno di un sotto template che posso richiamare ogni volta che voglio anche da posizioni diverse.

**ALCUNE NOTE INTERESSANTI:** nell'esempio fatto Ã stato abbozzato il concetto di vista. Per un utilizzo proficuo di tale meccanismo Ã importante notare che:

1. eZpublish definisce come standard a cui Ã opportuno attenersi alcune viste tipiche: **full** per il dettaglio, **line** per le liste nel corpo centrale, **listitem** per box laterali, **embed** per le liste degli oggetti correlati, **galleryline** per gli elenchi immagine nelle gallerie.

2. lo sviluppatore puÃ² creare nuove viste a piacimento, avendo l'unica accortezza di definirle a livello di **override.ini**. Personalmente ad esempio utilizzo spesso una vista personale che chiamo **homeline** in cui inserisco i template per gli elenchi di oggetti in homepage

3. Il concetto di vista Ã utilizzato anche agli **attributi** che possono essere stampati a video utilizzando i template standard di eZpublish per quell'attributo (con la formula giÃ vista **{attribute\_view\_gui attribute=\$node.data\_map.<nome\_attributo>}**), ma il cui contenuto puÃ² anche essere richiamato direttamente senza passare attraverso il template (la forma piÃ¹ comune Ã **{ \$node.object.data\_map.<nome\_attributo>.content }**). In questo ultimo caso tuttavia la sintassi dipende dal tipo di attributo utilizzato. Per maggiori approfondimenti si veda la [relativa sezione](#) nella documentazione ufficiale (*Object Attributes*).

E con questo abbiamo introdotto ed utilizzato il concetto di vista che riprenderemo anche negli esempi successivi.