

Planting the seeds of Magnolia:

Templater's quick start guide

for Magnolia 2.0

Version 2.0.1 / 28.12.04 / copyright 2003, 2004 obinary ltd.



Table of content

1	Introduction	3
2	Advertising the existence of a page template	4
3	Building a page template	5
4	Refining the page template - adding a custom property	7
4.1	Create a custom dialog	7
4.2	Create a paragraph definition pointing to our custom dialog	8
4.3	Adapt the page template	9
5	Adding paragraphs	9
6	Dissecting paragraph templates	12
7	Our first paragraph - a text paragraph with subtitle	14
8	Where to go from here	16

1

Introduction

Welcome to Magnolia, and congratulations to the aspiring Magnolia templater! (that would be you). Building templates in Magnolia is very simple once you have acquainted yourself with the basic mechanisms and structures involved. This document will try to make that process as painless as possible, with a focus on speed vs. depth.



The following examples assume that you have Magnolia installed and up and running inside tomcat, which is currently our default distribution. If you use a different J2EE server, adjust your thinking accordingly. For installation, follow the release notes available with the distribution of Magnolia.

Magnolia per default has installed two applications: authoring and public. The applications are basically identical, but can run on different instances or even different servers. It makes sense to start building templates inside the authoring application folder, so that you can try out your templates. Once your templates work as desired, you will have to copy these to the public application folder to see their output in the published website as well.

Inside of the webapps folder of tomcat, you will find the magnolia application (INSTALLDIR/magnolia/author/webapps/magnolia),

The folder contents are more or less as follows:

- WEB-INF/
- admindocroot/
- admintemplates/
- cache/
- docroot/
- logs/
- repository/
- templates/

For the course of this tutorial, all we really need to know is: templates go to INSTALLDIR/magnolia/author/webapps/magnolia/templates.

admindocroot and admintemplates contain files relevant for Magnolia administration, cache will contain a cache of our website. The docroot folder can be used to put files there that we would like to access via standard web mechanisms (e.g. static links). The repository will contain all content, as well as users, roles and custom configuration, including the information which templates are available. The templates directory contains the JSP templates that define the look (and functionality) of our website.

Templates exist on two levels - we create templates for pages, and we create templates for paragraphs. The page template will commonly contain things like the html head, css includes, Javascript and navigation.

Paragraph templates will be used to define the look (and functionality) of paragraphs. Examples might be an image paragraph, a text with heading and subtitle or a form designer to collect all those valuable tidbits your marketing department has been demanding for the last twenty-three early-morning strategy meetings.

One way to determine what goes where is that all the things that we only need once per page or have a fixed place on the page (e.g. a logo) could be part of the page template, and all the bits that we might move around on the

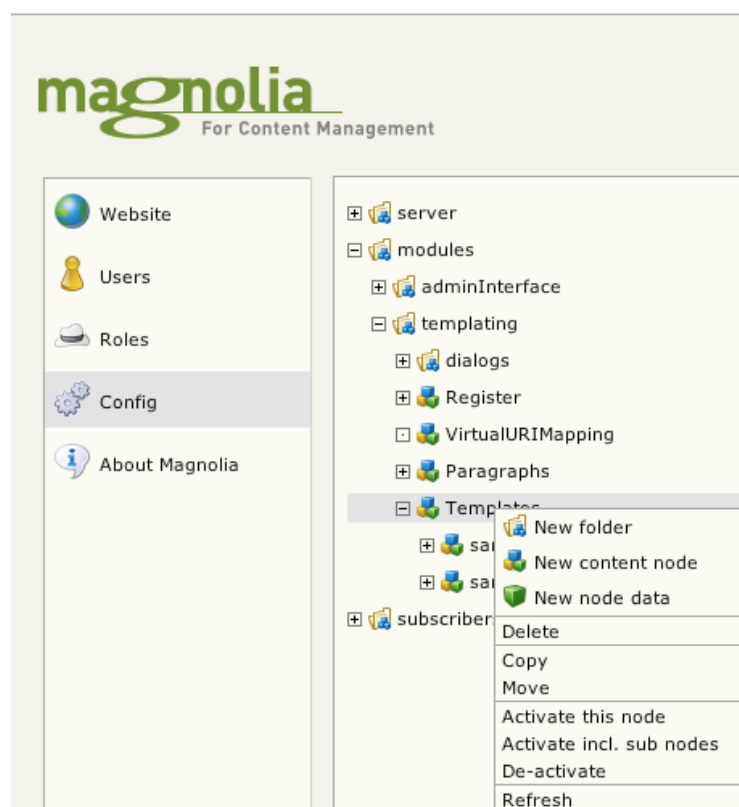
page or that occur multiple times (e.g. text blocks or images), will probably end up as some kind of paragraph.

From a different perspective, one can think of a paragraph definition as a mapping between the view and the model. The view is defined through a JSP template (or a servlet), and the model is defined through a dialog definition. More about that later.

We will first look at page templates, then at paragraph templates.

2 Advertising the existence of a page template

When a Magnolia user creates a new page, he will have a choice of available page templates in a dialog window. The available choices are a configuration issue and are defined in the “Config” repository. Point your browser to the authoring environment (something like <http://localhost:8081>), select “Config” and browse to /modules/templating/Templates. Right click on Templates and create a “New content node” (see screenshot below).



A node named “untitled” will show up; a double click on “untitled” lets us edit its name. We name it “quickstart” and add a couple of properties to describe our new page template. A right-click on “quickstart” lets us select “New Node Data”. “Node data” is a bit abstract depending on your background – it is simply an entry consisting of a name, a value, and a type. Let us add the following information:

Name	Value	Type
visible	true	Boolean
name	quickstart	String
path	/templates/jsp/quickstart/main.jsp	String
type	jsp	String

Our result should resemble the following:



Here is a short description of the properties:

Name	Description
visible	Show this template in the list of page templates. Admin templates generally set this property to false.
name	Template name. It will be stored with each page to define how the content is rendered. The name shows up in the template dropdown menu that lets you select which template a page is using.
path	The location of this page's starting template (which in turn may include further JSP's, see below)
type	Template type (currently jsp or servlet)

Restarting tomcat at this time will show up our entry in the dropdown menu of available templates. To verify, we select "Website" and for one of the pages, double click on the template name to get the drop down menu. It should look similar to the following screenshot:



Of course, before we create that new page, we first need to define the page template itself.

3

Building a page template

The page template itself is put under the path

`templates/jsp/quickstart/main.jsp`

since that is the path and file-name we have defined for the page template in the previous section. This directory can contain several templates, since main.jsp can include other templates. For example we could define menus, main content and side content boxes each in their own jsp. main.jsp would then include the other templates as appropriate.



WARNING: even though this file is in UTF8, be warned that template directories and names must be in ASCII.

What does a page template look like? First of all, we use jsp to define the look of the page. Second, we have defined a number of tags that will make life easier for you (more about these later).

So lets get started. Let us fire up that text editor and enter the magic words:

```
<%@ taglib uri="cms-taglib" prefix="cms" %>
<%@ taglib uri="cms-util-taglib" prefix="cmsu" %>
<%@ taglib uri="JSTL" prefix="c" %>
<cms:init/>
```

This will include the standard (JSTL) and magnolia tag-lib, and initialize the page.

Now lets start the html & head:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
<head>
  <title>
    <c:out value="${actpage.title}" />
  </title>
</head>
```

The actpage variable will have conveniently been set by the cms:init tag you have seen above. It contains the currently displayed page-object, and lets you access all properties of that page.

On we move to the body:

```
<body bgcolor="#ffffff">
<cms:mainBar paragraph="samplesPageProperties" />
<br/>
<h2>Hello World</h2>
</body>
</html>
```

Let us save the file in the template directory (in this case: templates/jsp/quickstart/main.jsp). You will need to create the directory first, as it is completely up to you how you name your templates, and "quickstart" is just an example for the purpose of this tutorial.

(No need to restart tomcat if you create or change jsp files). Create a new page, choose the freshly defined template, and view the result (by clicking on the newly created page). Here we have our first working page template, containing the Magnolia-specific "main bar" and our message to the world:



Congratulations! You have just created your first working template in Magnolia.

4 Refining the page template - adding a custom property

So let us explore some more details. First of all you will have noticed the

```
<cms:mainBar paragraph="samplesPageProperties"/>
```

tag. As you have seen when you viewed your first page, a green bar will be shown across the top of the page, and lets you access a number of admin options - at the moment, switch to preview mode, go to the administration environment or show the page properties dialog box.

-- short background intermission --

This bar will only be visible on admin pages. So what is the difference between an authoring page and a public page? Well, there really is none. Magnolia has the concept of stages, where we usually have at least an authoring and a public stage, each corresponding to its own instance of the application. So if this page is displayed on an authoring instance, the admin bars will be visible. If the page is displayed on a public instance, the admin bars will not be shown. If an instance is public or authoring is determined by the property Config: `"/server/admin"`

-- end of intermission --

So on we move. If you click on "page properties" in the mainBar, a page properties dialog will be fired up. A dialog defines the model (in MVC parlance). While the dialog defines which properties you want to collect and store in the repository, the template defines which of these will be rendered as output – and how.

Dialogs are defined through the administration GUI and reside under

Config: `"/modules/templating/dialogs"`

Which dialog is displayed when we click on the page-properties button? As stated earlier, a paragraph definition maps a template to a dialog. Paragraphs are defined in

Config: `"/modules/templating/Paragraphs"`

The mainBar tag points to a paragraph definition, and the paragraph entry points to the dialog being displayed. The idea behind this redirection is that different templates can use the same dialog definitions. This makes it easy to switch the look & feel of a site.

To define and use custom page properties we need to:

1. Create a custom dialog and define the desired custom property
2. Define a paragraph pointing to (1)
3. Adapt the page template pointing to (2) and displaying the custom property

4.1 Create a custom dialog

For our example, create a folder called quickstart through the admin GUI:

Config: `"/modules/templating/dialogs/quickstart"`

Copy the sample dialog

from

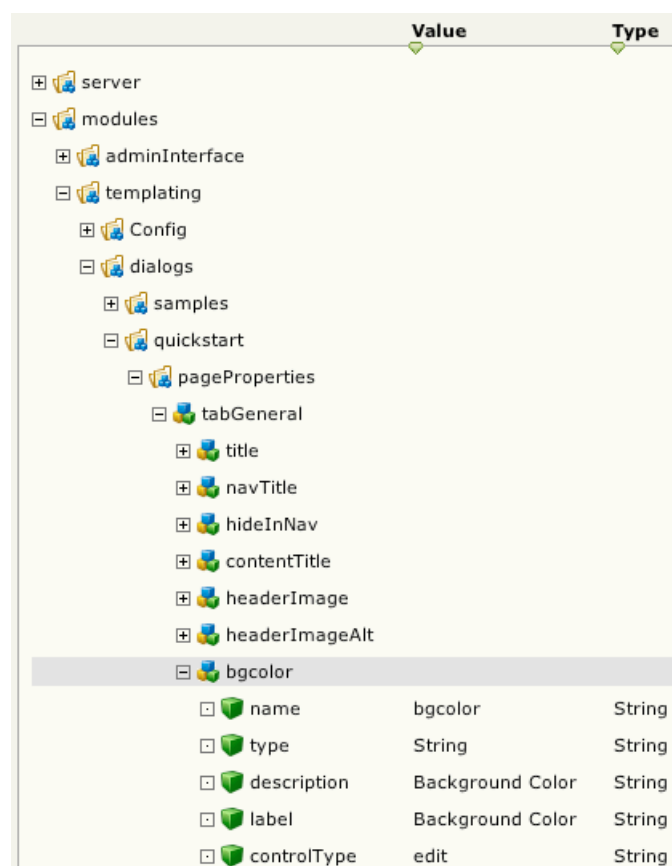
Config: "/modules/templating/dialogs/samples/pageProperties"

to

Config: "/modules/templating/dialogs/quickstart/pageProperties"

Note: you can paste a path into the path field located at the bottom of the admin screen to jump there.

As with page templates and paragraphs (as we will see later) the structure contains data nodes defining properties of the dialog. Let us add a content node for the background color of our page as shown below:



Thus we have a custom dialog for page properties that allows us to store our desired background color.

4.2 Create a paragraph definition pointing to our custom dialog

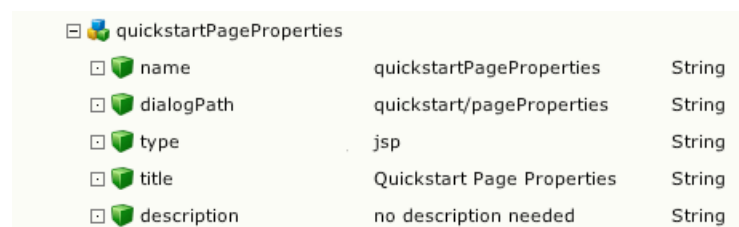
Copy

Config: "/modules/templating/Paragraphs/samplesPageProperties"

to

Config: "/modules/templating/Paragraphs/quickstartPageProperties"

and change name and dialogPath.



4.3 Adapt the page template

We need to change the page template's mainBar paragraph-Parameter to point to our paragraph definition and adapt the page template to use our new background color property.

We fire up the page template again (/templates/jsp/quickstart/main.jsp) and replace

```
<cms:mainBar paragraph="samplesPageProperties" />
```

with

```
<cms:mainBar paragraph="quickstartPageProperties" />
```

Since we now use our custom mapping between the jsp (view) and the dialog definition (model), let us use the background color property by changing

```
<body bgcolor="#ffffff">
```

to

```
<body bgcolor="<cms:out actpage="true"  
nodeDataName="bgcolor" />">
```

Save all files and restart tomcat. We will find a new entry called "Background Color" on our page properties dialog box. Let us try it out and enter a html-color-code (lets say #aabbcc) and save. After reloading the page, we will have a nice greyish-blue background.



The cms:out tag is from our cms-taglib that we have made available at the start of our page template:

```
<%@ taglib uri="cms-taglib" prefix="cms" %>
```

As we can see, cms:out allows us to access properties like the bgcolor above.

Congratulations! We have defined our first custom property for a page template.

5 Adding paragraphs

Within a page you will commonly find a number of differently formatted paragraphs. We may define any number of paragraph templates to be used within a page template.

Let us first create a new JSP to describe the content area of our page, where we will later add the paragraphs (while this is not really necessary, it will make things easier for you in the long run).

We replace "Hello World" line of code with

```
<c:import url="content.jsp"/>
```

in our page template, and create a second file called - you guessed it - content.jsp, which is stored in the same directory as our main.jsp (templates/jsp/quickstart/content.jsp)

We have to add the taglib definitions here as we did in main.jsp. However, we do not need to call the cms:init again (remember we have already initialized actpage in the importing main.jsp).

```
<%@ taglib uri="cms-taglib" prefix="cms" %>
<%@ taglib uri="cms-util-taglib" prefix="cmsu" %>
<%@ taglib uri="JSTL" prefix="c" %>
<h1>
  <c:out value="${actpage.title}"/>
</h1>
```

Reload the page - and you should see the page title (which you can edit in the page properties displayed from the main bar).



To access paragraphs, two more admin bars exist - the newbar and the editbar. Let us add a newbar to the content.jsp template. Add the following lines at the end of the file:

```
<cms:adminOnly>
<cms:newBar contentNodeCollectionName="contentParagraph"
paragraph="samplesTextImage"/>
</cms:adminOnly>
```

The contentNodeCollectionName will be used when we want to access the content we store via this dialog. It can be any name you want. The paragraph parameter allows you to add any number of paragraph definitions, separated by ','. If you have defined more than one, a selection box will be shown that allows you to choose which paragraph you wish to create (the same is true for page templates).

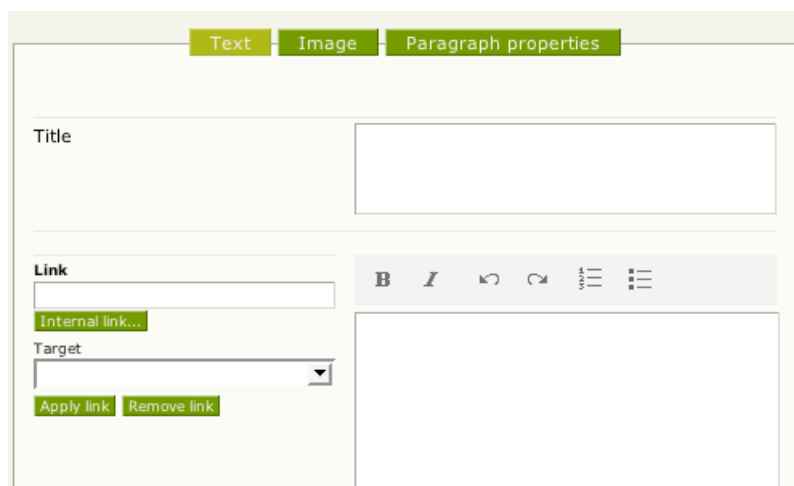


WARNING: there must be no spaces between the paragraph-names you provide.

The samplesTextImage-paragraph is provided with the Magnolia default installation, so let us see what it looks like: Reloading the page will show you an additional green bar at the bottom of the page, with a single new-button on it.



Click on "new" to get the paragraph dialog:



Enter some information, save and - you will see nothing, as we still have to include the content in our page template, which we shall do now. Complete content.jsp as follows (add the bold lines):

```
<%@ taglib uri="cms-taglib" prefix="cms" %>
<%@ taglib uri="cms-util-taglib" prefix="cmsu" %>
<%@ taglib uri="JSTL" prefix="c" %>

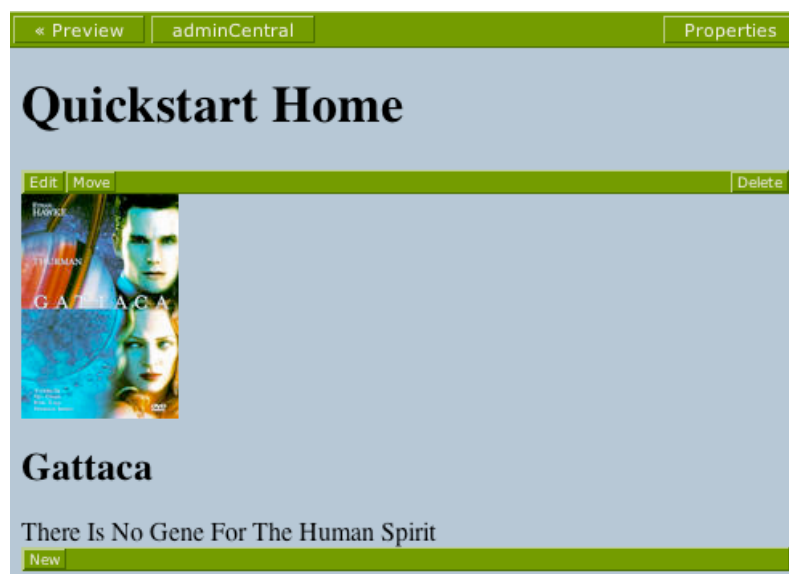
<h1>
  <c:out value="${actpage.title}"/>
</h1>

<cms:contentNodeIterator
contentNodeCollectionName="contentParagraph">
  <cms:adminOnly>
    <cms:editBar/>
  </cms:adminOnly>
  <cms:includeTemplate/>
</cms:contentNodeIterator>

<cms:adminOnly>
  <cms:newBar contentNodeCollectionName="contentParagraph"
paragraph="samplesTextImage"/>
</cms:adminOnly>
```

The contentNodeIterator tag will loop over the contentNode collection as defined in the contentNodeCollectionName attribute. This is the same contentNode collection that is used in the new bar. This mechanism allows you to create several individual lists on the same page. Within contentNodeIterator you find the includeTemplate-tag, which will render the content of the current paragraph. Adding an editBar within adminOnly-tags allows you to edit the paragraphs.

Your page now will look something like the following (provided you have added some text and image):



Now you can already add as many paragraphs as you want, move them around and delete them.

6 Dissecting paragraph templates

So let us see how and where the information for paragraphs is coming from. Like page templates, defining paragraph templates consist of three parts:

1. A dialog to define which properties we wish to provide.
2. A paragraph definition to map the template to the dialog and describe the template so that the author knows what it is meant for.
3. A template (jsp or servlet) to render the information stored via the dialog.

We already know that dialog definitions are located at

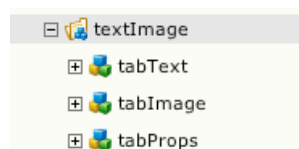
Config: `"/modules/templating/dialogs/"`

and that paragraph definition are located at

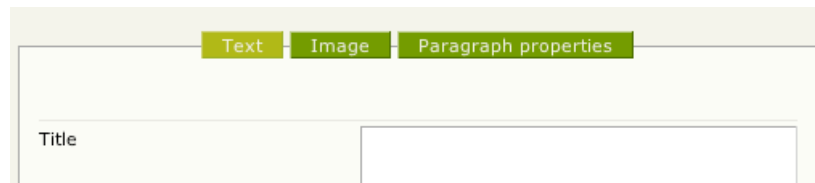
Config: `"/modules/templating/Paragraphs/"`

Finally, as a third part, you will need something to render that paragraph's information.

So let us have a look at the dialog definition at Config: `"/modules/templating/dialogs/samples/mainColumn/textImage/"`.

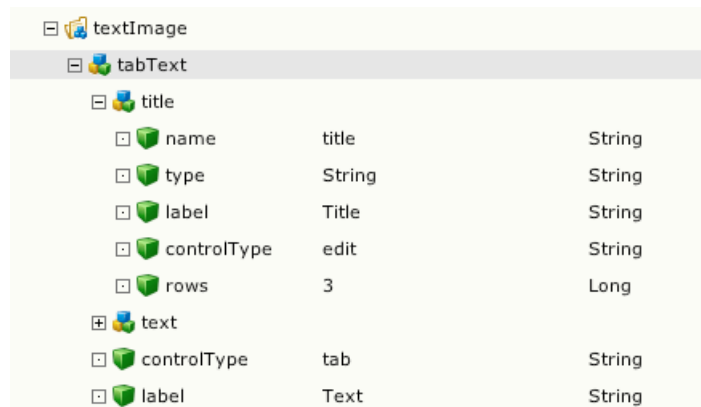


The top level of a dialog definition consists of the available tabs. Each tab can hold any number of properties, and will be rendered in the dialog box as shown below:



If we take a closer look at one of the tabs, we note that below the content node we find two data nodes describing the tab - its label and its (control-) type. Currently "tab" is the only available control type for tabs. You can provide custom controls and use these, if you wish.

At the same level the available properties for that tab are defined. This is shown in the next screenshot:



The properties work the same way we have already encountered them for the page properties, i.e. we define one content node per property we wish to add, and add data nodes that describe that property.

Second, look at the paragraph definition at

Config: `"/modules/templating/Paragraphs/samplesTextImage"`

It has the same format as the page properties definition, with one difference: an additional node data element called "templatePath" exists, which names the template that renders this paragraph.

This stems from the fact a page (template) has a single page-properties dialog, but usually several different paragraph dialogs. Consequently, a page template already knows which template will render the page properties: itself. There is no need to store that information in its page-properties paragraph definition.

On the other hand, for a paragraph we need to know which template will render it. Therefore, we define the template path of a paragraph in its definition.

As you can see from the templatePath data node, the sample paragraph we have been using so far has its jsp-file located at `/templates/jsp/samples/paragraphs/textImage.jsp`

Third, a JSP file exists to render the paragraph information. Let us have a look at a couple of tags we find in this jsp, for example:

```
<cms:ifNotEmpty nodeDataName="image">
```

Tests if a given nodeData has a value. The nodeDataName (in this case: image) is defined in the dialog-definition above. There is a corresponding cms:ifEmpty – Tag

(All tags are documented at www.magnolia.info)

So by now we should be able to implement our first custom paragraph. Lets do it!

7 Our first paragraph - a text paragraph with subtitle

Just to get you started, we will define a very simple text paragraph, consisting of Title, Subtitle and Content properties. We start with defining the dialog, then create the paragraph description and finally we will define what the output will look like using JSP. This what we will get:

Gattaca
There Is No Gene For The Human Spirit
Futuristic story of a genetically imperfect man and his seemingly unobtainable goal to travel in space.

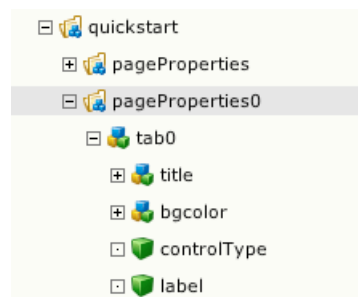
Copy the page properties dialog

Config: “/modules/templating/dialogs/quickstart/pageProperties”

to

Config: “/modules/templating/dialogs/quickstart/extendedText”

To do so, right-click on pageProperties, select “copy” and move the mouse pointer to the parent node (“quickstart”). The subtree will be copied to quickstart/pageProperties0; double click on the name to change it to “extendedText”:



(Alternatively we can create the structure from scratch)

Let us define the properties we wish to have available by deleting background color data node and copying the “title” node to “subtitle” and “content”, then renaming the entries to match the following screenshot:

extendedText			
tabGeneral			
title			
name	title		String
type	String		String
label	Title		String
description	The title of this paragraph		String
controlType	edit		String
subtitle			
name	subtitle		String
type	String		String
label	Subtitle		String
description	This text is shown below the title		String
controlType	edit		String
content			
name	content		String
type	String		String
label	Content		String
description	The body of the paragraph		String
controlType	edit		String
controlType	tab		String
label	General properties		String
label	Text		String
width	550		String
height	650		String

We have defined the dialog to enter paragraph properties. Now we add a paragraph definition to provide a mapping between the paragraph properties and a template.

Config: `/modules/templating/Paragraphs/quickstartExtendedText`

The simplest way is to copy an existing paragraph definition and adjust it to our needs. The result should look as follows:

quickstartExtendedText			
name	quickstartExtendedText		String
templatePath	/templates/jsp/quickstart/paragraphs/extendedText.jsp		String
dialogPath	quickstart/extendedText		String
type	jsp		String
title	Quickstart Extended Text		String
description	Our first custom paragraph		String

Now define the rendering in html by providing the jsp-file `/templates/jsp/quickstart/paragraphs/extendedText.jsp`. We keep the html intentionally simple so that we do not drown relevant parts in style & layout.

```
<%@ taglib uri="cms-taglib" prefix="cms" %>
<%@ taglib uri="cms-util-taglib" prefix="cmsu" %>
<%@ taglib uri="JSTL" prefix="c" %>

<cms:ifNotEmpty nodeDataName="title">
  <h2><cms:out nodeDataName="title" /></h2>
</cms:ifNotEmpty>
```

```
<cms:ifNotEmpty nodeDataName="subtitle">
  <b><cms:out nodeDataName="subtitle" /></b>
</cms:ifNotEmpty>
<br/>
<cms:ifNotEmpty nodeDataName="content">
  <cms:out nodeDataName="content" />
</cms:ifNotEmpty>
```

Finally, to make this paragraph available in our quickstart page template, we need to revisit `/templates/jsp/quickstart/content.jsp` and change the line

```
<cms:newBar contentNodeCollectionName="contentParagraph"
paragraph="samplesTextImage" />
```

to

```
<cms:newBar contentNodeCollectionName="contentParagraph"
paragraph="samplesTextImage,quickstartExtendedText" />
```

Make sure you have no space between the paragraph definitions. Restart tomcat and enjoy your first custom paragraph!

Extended Text	
Titel	<input type="text" value="Gattaca"/> The title of this paragraph
Subtitle	<input type="text" value="There Is No Gene For The Human Spirit"/> This text is shown below the title
Content	<input type="text" value="igly inobtaining goal to travel in space"/> The body of the paragraph

Save Cancel

Remember that we customized only one (the author-) instance in this tutorial. To make your changes available at other (public-) instances, you need to copy the new or modified jsp files to the other instances using the file system. All changes to the configuration can be activated directly through the administration GUI – simply click on the corresponding content nodes and choose “Activate incl. sub nodes”. In our case, we have modified “dialogs”, “Paragraphs”, “Templates”. As configuration can differ between your instances, you should generally not activate the whole Config: “/modules” tree unless you know what you are doing.

8

Where to go from here

Places to explore:

- The **samples-template** we provide will give you a lot of ideas how to proceed, as it is considerable more complex than the examples above, and includes amongst others a mail form paragraph that allows you to create arbitrary forms.

- The **tag libraries** - check out separate documentation or the source code to see which tags the Magnolia tag libraries provide to make templating in JSP easier. (Also check out JSTL).
- Check **www.magnolia.info** for updates and news as well as **wiki.magnolia.info** for community documentation
- Visit **jira.magnolia.info** to view reported bugs or feature requests and to submit your own.
- Get registered on the **mailing-list** to discuss with your fellow Magnolians (to subscribe, send an empty mail to **user-list-on@magnolia.info**).

For feedback and suggestions regarding this documentation, please contact the author directly: boris.kraft@obinary.com