Drupal Panels

Whilst Drupal itself is the most fantastic and powerful content management system / development platform, there are a few modules which are, in our view, almost as important as the core system itself. These have revolutionised the platform and the way we use it. The modules which go into nearly every install of Drupal we do are:

- CCK (Content Construction Kit)
- Views
- Panels

These modules are both incredibly powerful in their own right, but have also been built with the same principles of extensibility that Drupal core has. This makes them an invaluable tool when rapidly creating sites. We would, in particular, like to thank the developers of Drupal itself and of these modules (both Views and Panels are created by 'merlinofchaos'). Their work is amazing and when you look at their code, you cannot fail to be impressed.

What does Panels do

Panels is the ultimate way to structure pages of a website. You can respond almost immediately to the changing needs of navigation, presentation, and page layout – including on production sites (think changing the front page based on Google Analytic bounce rates). Essentially they let you move blocks of content around a page using a drag and drop interface – but it's more powerful than that. You can dynamically choose what is shown on a page based on data being passed to it or even render a page in two different ways. Once you have used Panels, you'll wonder how you ever did without it.

Some terminology to learn

We initially found Panels slightly difficult to learn because the terminology was unfamiliar. It doesn't take long to figure out, so here's a quick introduction (based on our non-official understanding).

Term	What is it
Custom Page	This is a standalone page which is just like any other node on Drupal. It has one specific URL and is listed in the Content List along with other nodes. The only difference is that it is a panel – made up of different bits of content from nodes, embedded content (which is content entered into the panel itself), blocks and views. It is also known as a 'landing page' in some navigation.
System Page	These are panels which override system pages – such as the Node or Taxonomy page. These in turn are referred to as templates.
Node Template	This is a template (System Page) which creates a specific layout for a given content type. If you have a content type 'Blog', then you can create a Panel layout for it. You can also do it via the standard Node Templates (described next)
Manage Pages > Templates	In the Panels Dashboard and the Page Manager (if that module is enabled) you will see a list of built in templates. These need to be enabled to use them. For example you can enable the 'Taxonomy Term Template' which overrides the way the standard Drupal taxonomy pages are laid out. Similarly you can do the same for a 'Node Template', so you can layout nodes in different ways depending on criteria that you set.
Variants	In the System Pages and Custom Pages you can have many different ways a node is laid out depending on criteria you set. Each of these ways is called a

	'variant', and it contains all the logic for when it is to be used and what the layout is going to be. Variants are most useful in System Pages (such as nodes, user or taxonomy pages) where different conditions can result in different layouts. We'll talk about this a lot further down.
Selection Rules	For each variant you create, you can set an enormous number of 'selection rules'. These are basically there to make a decision on whether it is that variant which will be used to display a page. For example, you could set a rule that says if a person is logged in then display a page one way, but if the person is anonymous then don't use this variant.
Contexts	This is one of the most powerful aspects of Panels and at the beginning unfamiliar, so maybe best to take an example. I have a node which has a taxonomy term 'Studio portraits' (as in photography), and I have a page which promotes how wonderful my photos are. I also have a bunch of testimonials from people saying so, which I've brought together in a 'view' (in amongst testimonials about all sorts of other stuff). I want the testimonials about 'Studio portraits' to show only on the 'Studio portrait' page. So essentially I need to know the taxonomy term of the main node to pass into the view which will also show on that page. That taxonomy term (and all the related data, like the Term ID, the name etc) becomes a 'context' – a grouping of data which is passed to each bit of the panel. Hopefully this will become clear later.
Layout	The layout is how the page is displayed – 1 column, 2 columns, 3 columns, biscuit, wafer – you name it. You can do almost anything (within the bounds of HTML and CSS), and this is entirely extensible within your theme. Amazing.
Content	This allows you to put content – node data, search forms, views, self created text, breadcrumbs, the logo – anywhere within the 'Layout' defined above. Any content created by Drupal (which eventually gets put into the traditional regions and blocks) can be put anywhere in a Panel Layout. This is incredibly powerful.
Mini-Panels	These are like 'Panels snippets' which can be reused inside other panels. It's a bit like creating blocks in standard Drupal which you can use around the place.

Installing Panels

Panels is made up of two parts – CTools (Chaos Tools – another merlinofchaos inspired contribution to Drupal) and Panels. Chaos Tools are the underpinnings – a library of helpful tools and frameworks – used by Panels and other modules (like Feeds). Panels is the other one.

Install from http://drupal.org/project/panels and http://drupal.org/project/ctools

Or alternatively use drush to download panels and ctools.

There are lots of submodules in these which we found a bit confusing at the beginning, so here I'll list the ones we commonly use/enable.

Chaos Tools (ctools)

[This needs to be changed as documentation for advanced help is done]

Module	Do We Use it?	Comments
Bulk Export	Not much	This allows exporting and importing structures you've created – a bit like Views Bulk Export. You develop on your dev site, export everything to code, then push your page manager pages with all their configuration to your live site just by updating the code.
Chaos Tools	Always	This is core CTools and Panels requires it
CTools Ajax		We haven't used this one yet

CTools Plugin Example	No	This is a fantastic 'tutorial' on using CTools functionality. We don't enable the module, but we look at the code in it quite a bit. If you plan to do customisation, look at this stuff.
Custom content panels / rulesets	Not Yet	These create reusable components, but we haven't used them yet.
Page Manager	Yes	This provides a navigation menu in Site Building > Pages which we find pretty useful to get around (it does duplicate the Panels Dashboard a bit which some people found a bit confusing)
Stylizer	Not Yet	
Views Content Panes	Always	This allows Views to be embedded into Panels – probably one of the best bits of content to embed. We use these in nearly every page!

Panels (panels)

Mini panels	Sometimes	As mentioned above this gives you the ability to create snippets to reuse in other panels or inside 'block' spaces. We find ourselves creating custom layouts instead of using these – not sure which is best or if there's just personal preference.
Panel Nodes	Not Yet	In general we're using panels selection rules to override the way nodes display, so we haven't yet found ourselves using this. In general, you will not use page manager AND panel nodes in the same application. If you find yourself wanting a truly panel-node solution, the new panelizer module is a better solution anyway. It is possible that as we move forward, panel nodes will be ejected from Panels into their own project and we will favor Panelizer as the solution.
Panels	Always	Core Panels
Panels In Place Editor	Not Yet	We tried to use the powerful drag and move features of this, but since we're a bit geeky we ended up just doing custom layouts instead.

The panels dashboard: This is really kind of an overview; because Panels is so spread out, this brings everything into one place primarily to help you find it. You don't actually truly administrate things from here, but it's a good starting location to get to what you do want to administer. As such, I wouldn't really recommend editing pages from the panels dashboard. What it shows you from page manager is a pretty small snippet of what's available.

An important note:
We got burnt by this once and shame on us for not following/reading instructions. I seem to remember reading somewhere that Panels and CTools have coordinated releases – so you should always be on the same version. I guess the latest one. I can't find where I read that.

Getting started

It's probably easiest to work through an example.

Take a look at the page of a typical photographers' website. Well, not so typical actually. The majority of data is highly dynamic – lots of content is only relevant to this particular offering (studio photography) and not to other offerings (schools photography for example). They want their blogs, articles, testimonials and promotions related to studio photography to show only on that page and not on other pages of the site. What's more their images are in unusual places and not in the normal page flow.

Before panels, you would have had to change Drupal's template files and write code to achieve this layout – probably a week's work. This site can be done in less than a day with Panels and Views – it's a doddle. We'd love to show you how!

Structuring the site content

We usually start a site by going to CCK. That's where you structure the content which is going to be stored on the website. After that (well iteratively) we go to Drupal's Taxonomy. That's where you create the inter-relationships between content on the site. With that in place and some test data we go to Views to create dynamic data and finally we make it over to Panels to lay it all out.

Site structure – CCK & Taxonomy.

We create 2 content types – 'Standard Page' and 'Testimonial'. The standard page is used for most pages, the blog and articles. It contains multiple image fields and text area/fields. The testimonial contains a single

image and single text area.

We then create 2 taxonomies – 'Page Type' and 'Offering'. The first is used on standard pages to identify if it is a normal page, a blog page or an article (we need this to lay them out differently in Panels later). The second is used by the studio for things like 'Studio Photography' or 'PR Photography' or 'Makeovers'. We also need this to link testimonials for each offering to the main pages.

There's so much about CCK and taxonomy out there, we won't go over the specifics here.

Site structure – Views

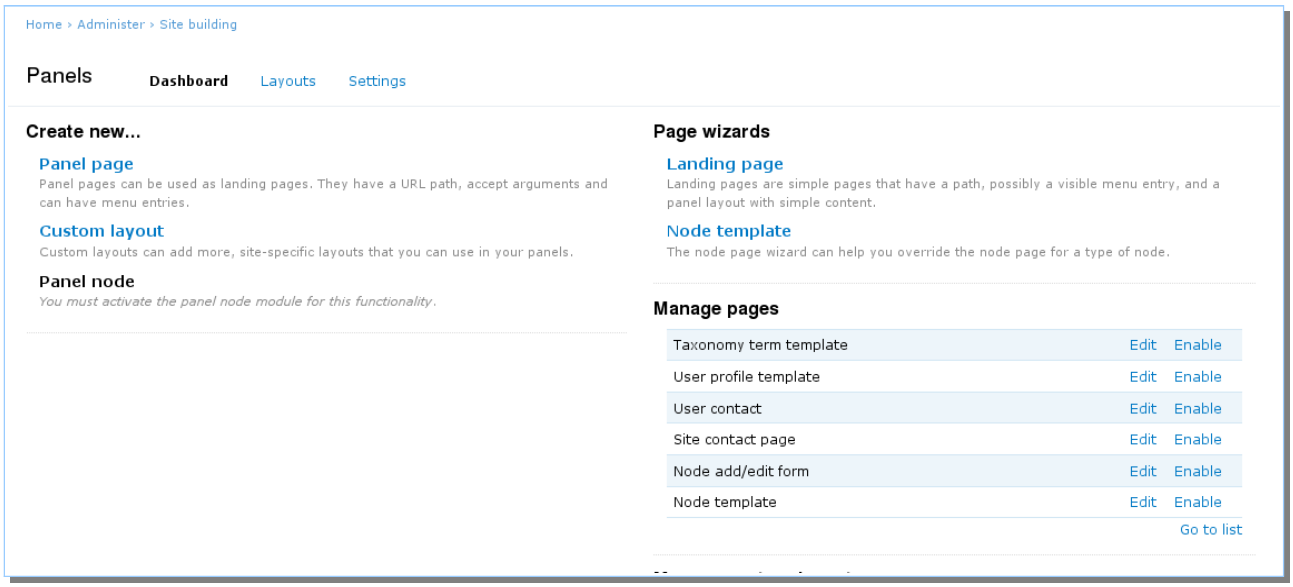
Our next step is to create a number of views to display the list of blog items, the list of articles and the testimonials. You could create one view with different 'displays' or multiple views. The most important thing is to ensure there is a way the view knows that a particular term in the 'Offering' taxonomy is selected (by being on a page which has that taxonomy). This is done via the 'argument'.

A screen shot from Views for the testimonial. The term ID is passed in as an argument, which acts as an additional filter (so the view will be filtered on Term ID AND Node Type testimonial)

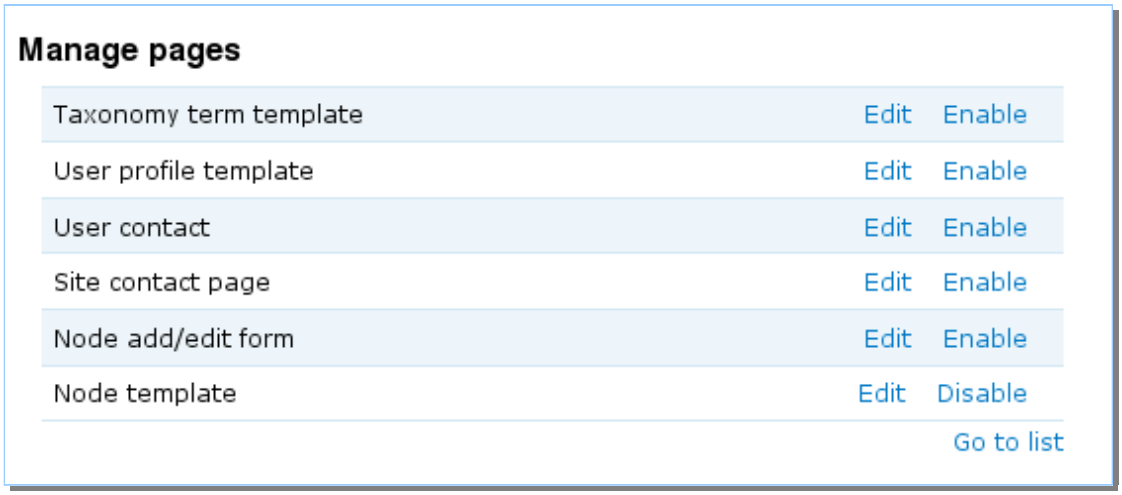
Site structure – Panels

We now have the pieces in place to create our page and we can create our panel. We want every site page to look the same or similar, so we'll get Panels to intercept every request for a node and then lay it out. You do this by enabling the 'Node Template'.

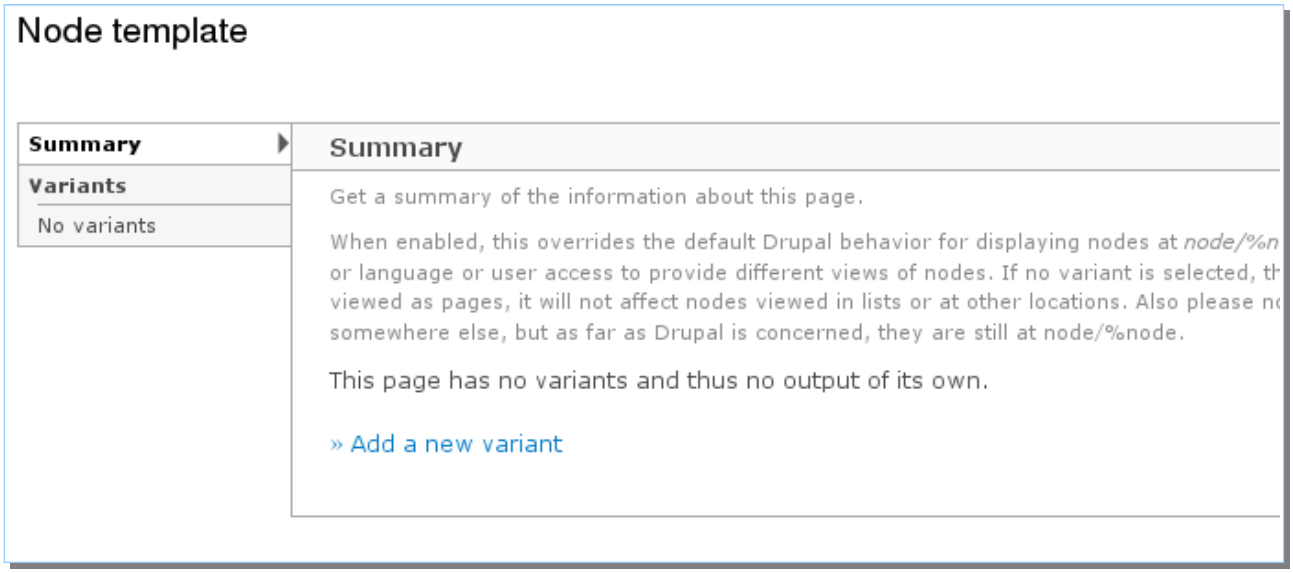
Go to Administer > Site Building > Panels > Dashboard.



Here, under 'Manage pages' you will see the templates. Enable the 'Node template' by clicking Enable.



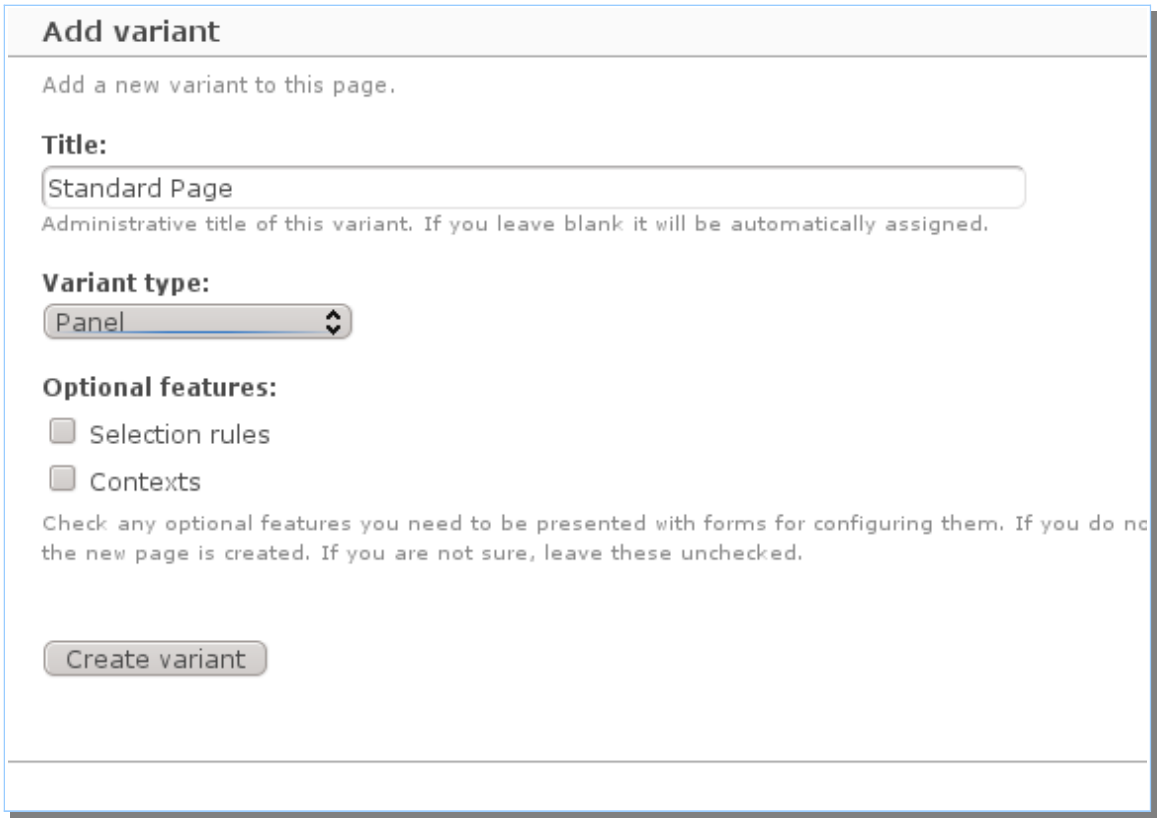
Then click 'Edit', and you'll be taken to a Panels page where you can manage exactly how the page should look.



In the templates, you'll come across the concept of 'variant'. A variant is a group of things which determines if a page will be rendered using it and how the page will be rendered. If there are no variants then that template will do nothing and the page will render in the standard Drupal way.

This will become clear as we create the variant. So we click 'Add a new variant'. This starts a little 4 step wizard

1. Wizard – basic information

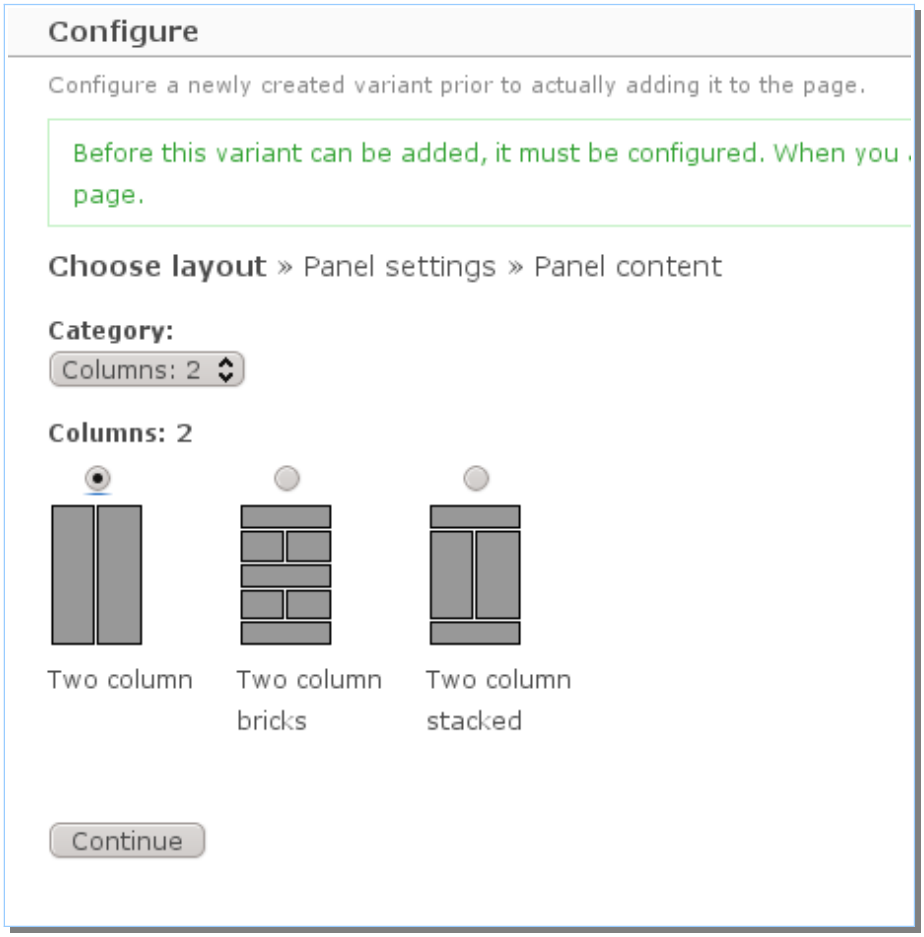


We get a form for basic variant information. All you need to do is enter a title – something which describes what this variant is for. In our case it is the rendering of the 'Standard Page' across the site, so we call it that.

Note that you can select optional features, but we tend to just get to the created variant where you can do all

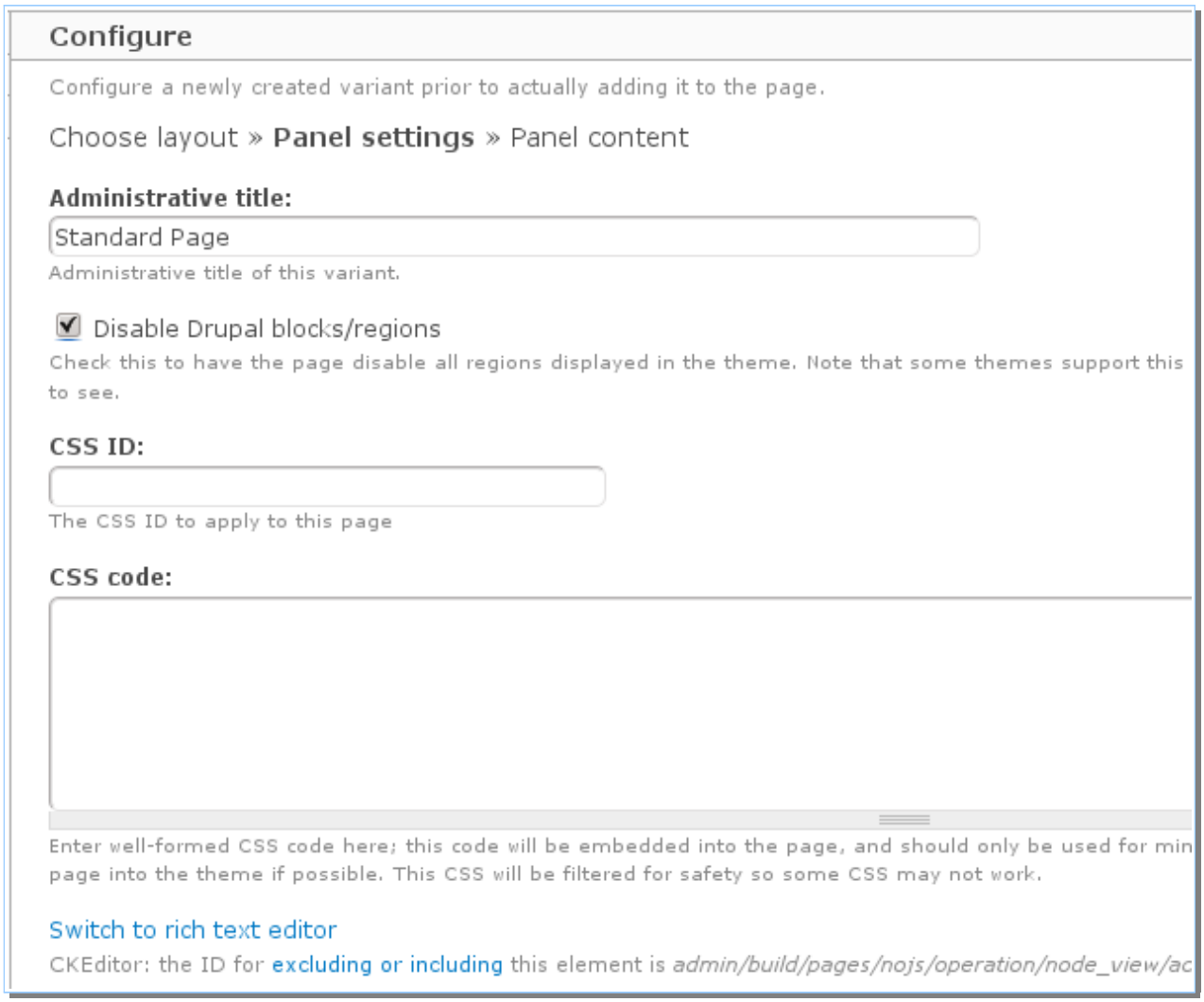
of this editing anyway. Click Create variant.
The next step is to choose a layout

2. Wizard – choose layout



Panels gives you huge ability to chop and change between layouts – 1, 2, 3 column layouts in brick or not. You'll want to choose a layout which suits your page. This can always be changed later (and in our case we will create a specific layout of our own). In the mean time we choose a 2 column simple layout.

3. Wizard – Layout Configuration

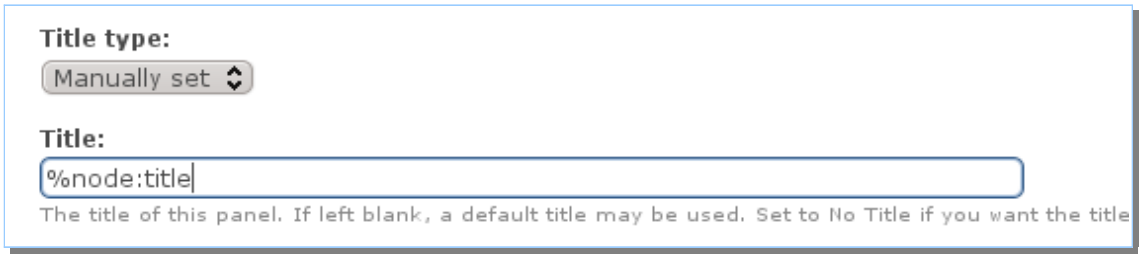


Here you can add useful information to the layout and CSS being output. We often choose to 'Disable Drupal blocks/regions', as we get Panels to do it all. For those using standard themes as a basis, this generally removes all the stuff you see as regions in the 'Blocks' setup – so no left and right sidebars (or blocks in the content region). In Garland based themes the header and footer blocks remain.

You can also add a specific CSS ID or your own CSS (if you don't have access to the stylesheets, I guess). It's probably better to keep all styling in one place (in the stylesheets), which is why you can add your own ID to style this particular page. This ID is added at the root of the panel HTML, not at the root of the page HTML (for those familiar with page.tpl.php, the page tags just inside the <body> often have styling classes and Ids which come from there. It's only within $content that the panel HTML appears).

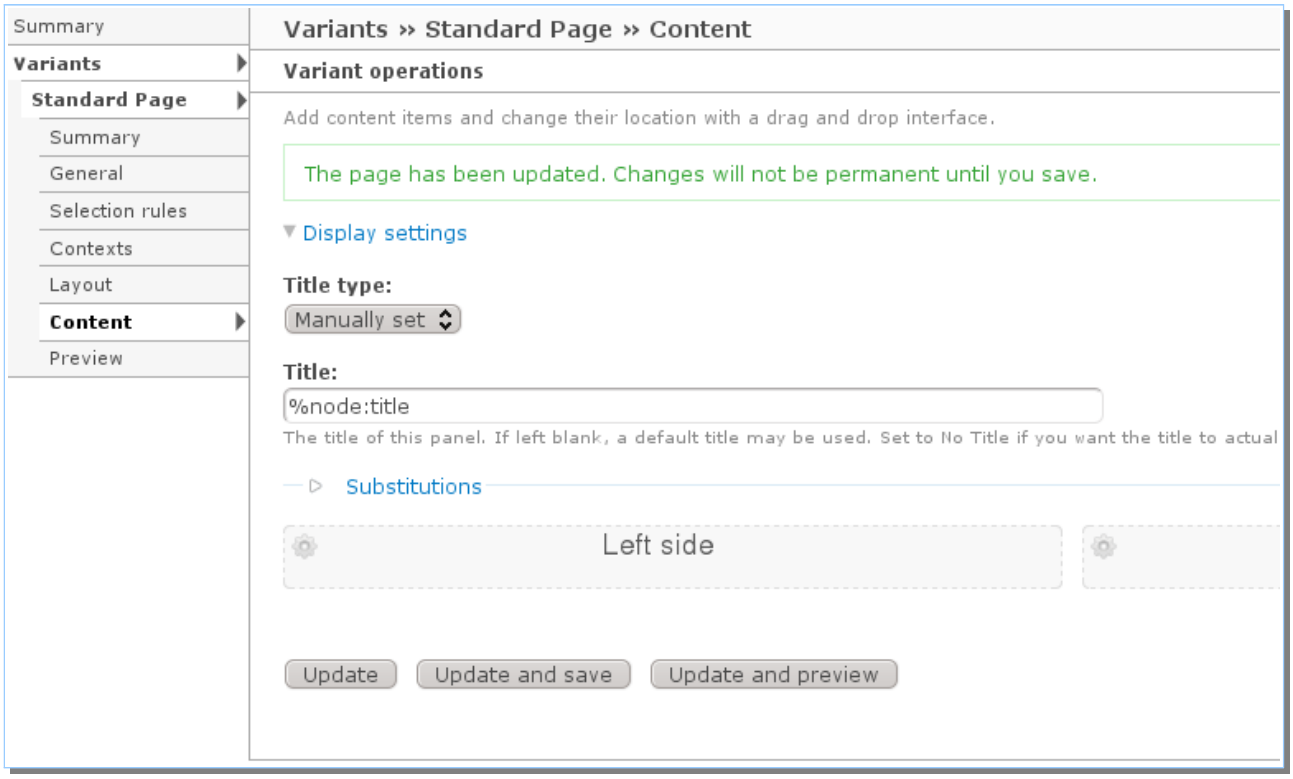
4. Wizard – Configure Content

Finally you get to the place you can add content to the page. We generally add just the basic stuff here at first, because you can always come back and edit any part of the variant which you've configured so far. So in this case we change the Title to take the title from the node.



We'll come to a longer explanation of what the %node:title means – but in summary it says take the %node context and use it's title.

We click Create Variant, and we can see the fruits of our labour!



Just a quick note – like Views, Panels doesn't save data (make it live) until you specifically ask it to. It makes sense when you have a production site, you don't want users to see your half-baked work until it's ready. So you simply use update. When ready, you must save. You'll see the following icon in the top right until you do.

The Variant Sections

Once created your templates will have one or more variants (named by you) and within each variant a number of tabs which allows you to continuously change and work on each one.

In our case we have a number of variants – each one a bit different. The Gallery of photos is laid out differently from the Blog or Standard Page.

Within the Standard Page (which we are currently editing, and which is therefore not collapsed) we see:

- Summary – not editable, just an overview of the panel
- General – the panel title, CSS code, disable blocks toggle
- Selection rules – conditions on which this variant is used versus others
- Contexts – data passed into and around the panel
- Layout – the visual structure of the page
- Content – the content within the layout.

You select each one to navigate around the panel definition. Remember to click 'update' or 'update and save' to store your changes.

Setting up the variant

This is an iterative process as you move from tab to tab – so your process/preference may be different. This is the process we find logical for us.

Setting up the Variant - Add content to the panel

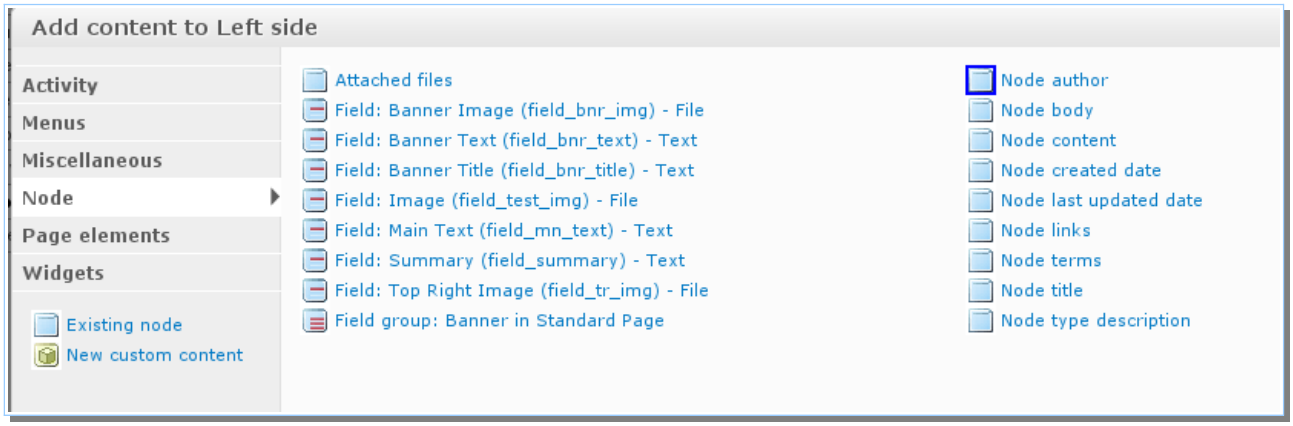
We go to the 'content' tab and all areas are shown in the layout as grey boxes. Each part of the panel layout has a name – so 'left side' or 'right side' or 'left top'. You add content separately to each part or region.

Note the little gear on the top left. By clicking that you will get a menu with which you can add content.

Select 'Add content' after clicking the gear. You'll want a decent modern browser because there's a lot of javascript stuff being used here. If it doesn't work, we've found Firefox to be consistently pretty good with Panels.

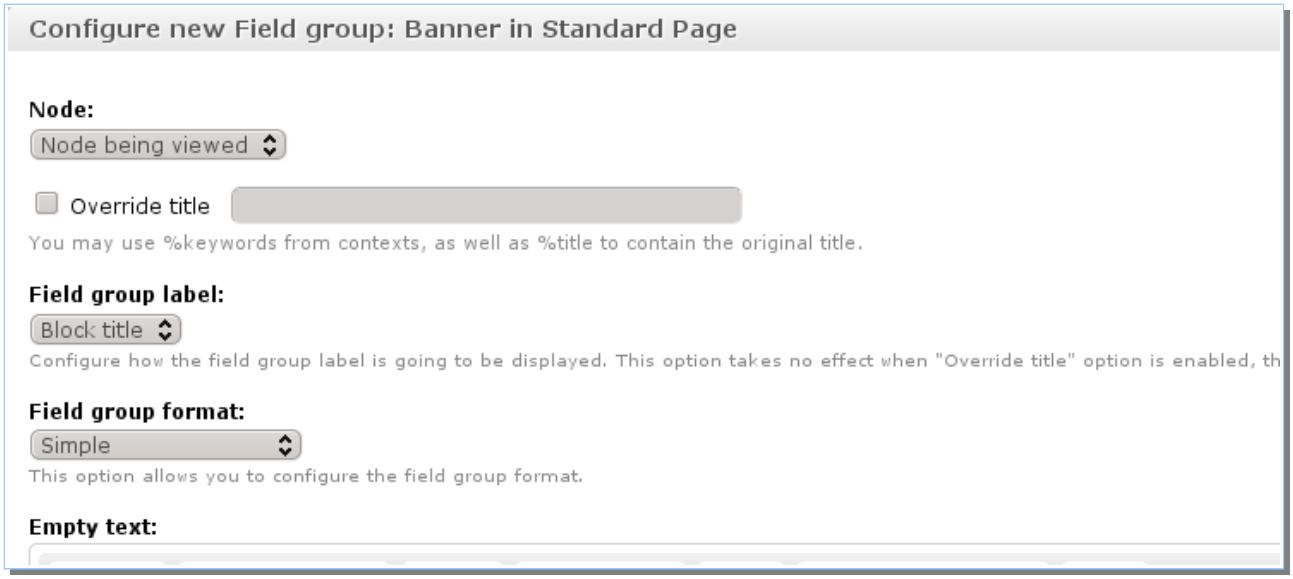
Once you've selected 'Add Content', a dialog box pops up offering a huge range of Drupal content to add to your area. The dialog box has a number of tabs on the left side to categorise the content for you.

There's an explanation later of each of these. For the moment, we simply want to add plain 'Node' data to the Panel. So we select 'Node'. This contains all the elements inside a node – the body, the title and if you're using CCK, each CCK field and group. You can pick and choose which one's go where on your layout. Amazingly powerful!

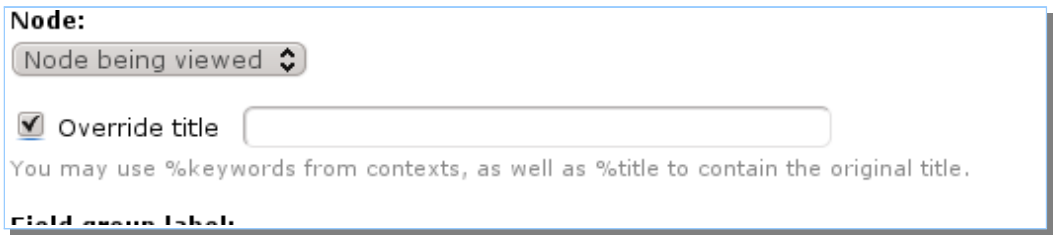


The Node elements are all listed. We have a CCK group called 'Banner' which pulls together the banner of the page. So we insert that by clicking on it. (Field group: Banner in Standard Page) – first column at the bottom.

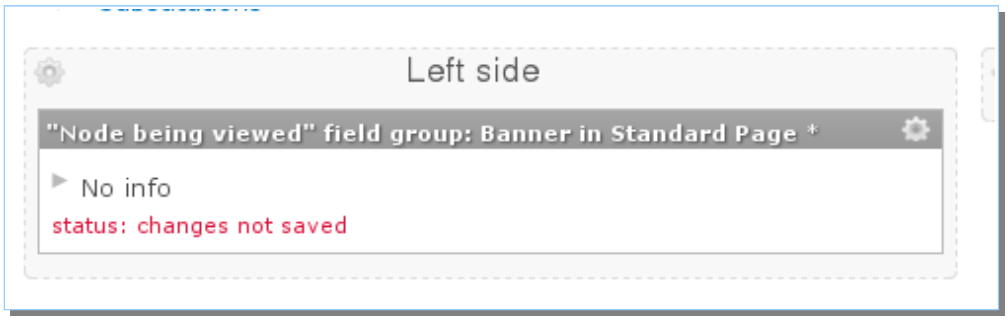
We then get taken to the settings dialog related specifically to the banner being inserted. Each bit of content you add to the panel has settings associated with them – depending on what that content is and is trying to do.



In this case the data will come from the current node (Node being viewed), but we don't want it to have the CCK Fieldgroup title. So we click override title and leave it blank.

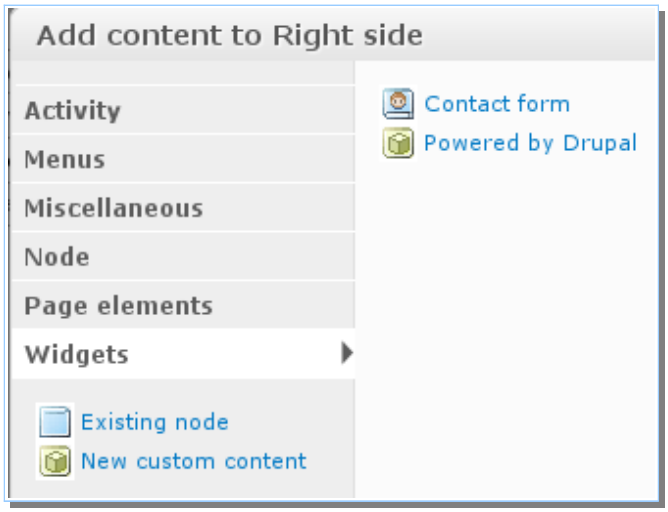


Then we scroll to the bottom and confirm. We return to our content layout and our new content is there.



You can always edit or change your settings for that particular bit of content by clicking the gear on it's top right and selecting 'Settings'.

We can do the same for the top right image, adding it to the right column, and the contact form. This time we select 'contact form' from the 'widgets'



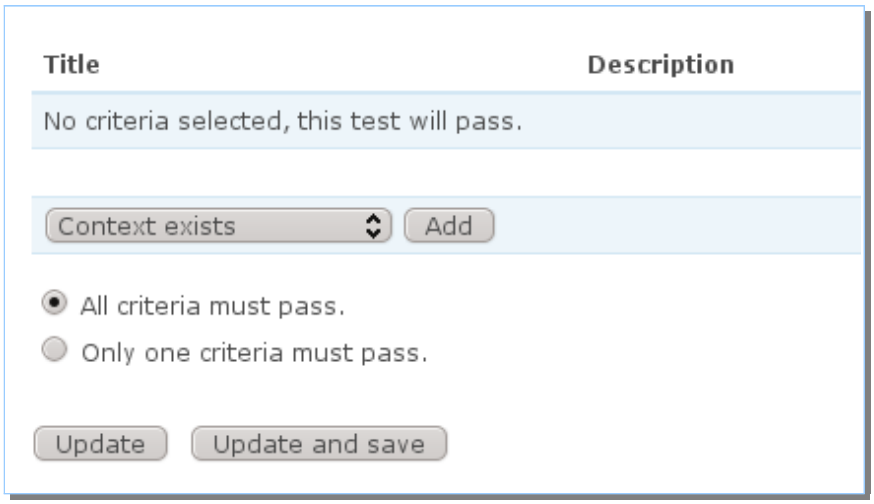
In this way we start filling out our 2 column layout. It's not quite what we wanted, so a little later we'll show you how to create your own custom layout.

Setting up the Variant - Setting the selection rules on the variant

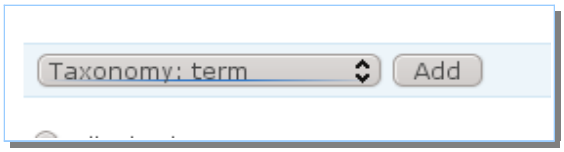
When you have a number of variants, Panels will go through each one in order (from top to bottom) and check the selection rules. If that rule matches, then that variant is chosen and the page is laid out according to it. So you want to create a logical set of selection rules from the first variant to the last. In our case we want to check the following:-

- If the node type is 'Gallery' then use the first Gallery variant
- If the page type as defined in our taxonomy Page Type is 'blog' then use the Blog variant (second)
- If the node type is 'Web Form' (using the web form module), then use that variant (third)
- Otherwise as a catch all use the 'Standard Page' variant (fourth).

This is how we set up the Blog selection rules.
We go to the Blog variant and select the tab 'Selection rules'. We get this form.

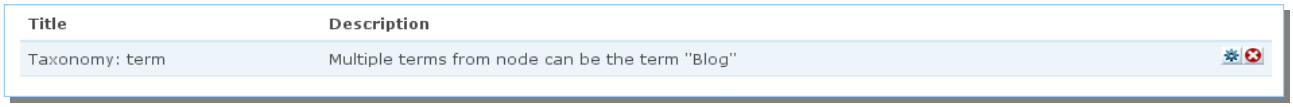


Here we need to choose which selection rule to use from the drop down menu. If you click on it you'll see all sorts of options – like node type or taxonomy term. We select taxonomy and click add.



We get a dialog box which pops up, allowing us to choose which vocabulary (Page Type) and what the term should be ('Blog').

And when we save we see that selection rule listed. It can always be deleted (x) or changed (gear).



So, in summary. When someone creates a new page and sets the 'Page type' to 'Blog' (done in the taxonomy section of the edit form), then this particular layout will be chosen.

Panels works through the variants from top to bottom, looking at each selection in turn. This is like an 'if... else if' statement. So,

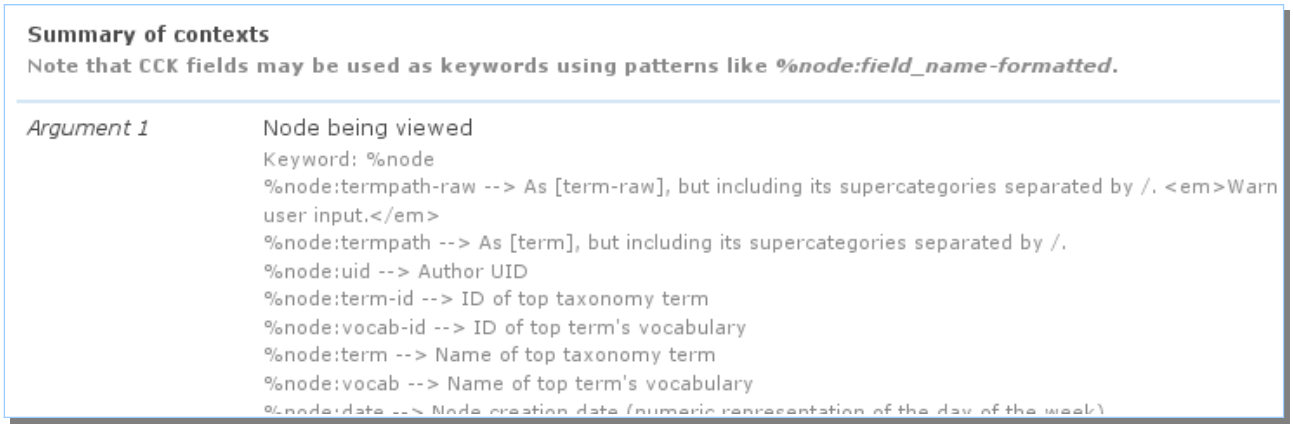
if (selection criteria from variant 1 is true) { display content from variant 1 }
else if (selection criteria from variant 2) { display content from 2 }

else { if there is a variant with no selection criteria, use this. If not, use normal Drupal page}

Setting up the Variant – Contexts

You may or may not need to add your own 'contexts' to the panel. Some context is already built in – so for example if you have a 'node template', then the 'node' context is already there in the panel.

The context is basically data which is available throughout the panels. If you go to the 'contexts' tab and look down at the bottom, you will see a huge list of data available.

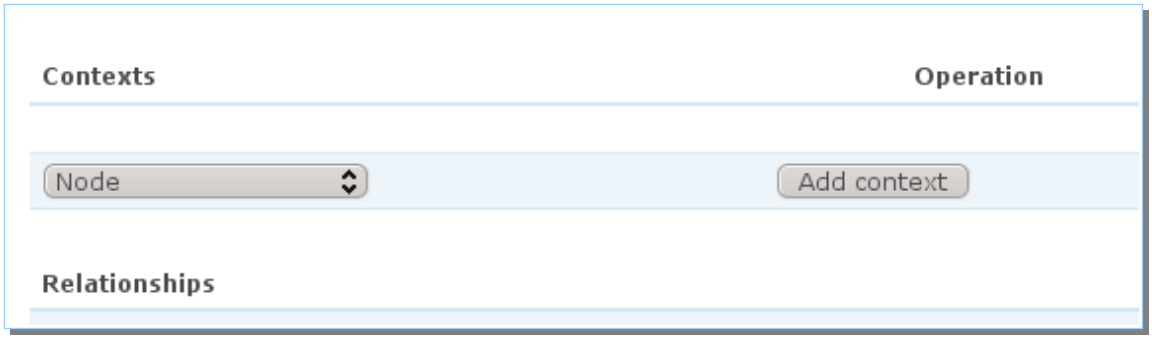


This summarises all the context data available. A context is given a keyword (like 'node') and you can reference it with %node. The data within it is referenced such: %node:title. This takes the title from the node.

You'll see quite a few places in panels which allow this kind of usage. You will see others (such as input to Views Arguments) where a drop down list of all the context data is available. Even more powerfully there are some places you can enter your own PHP code and access this context data.

In some cases you will find the standard context data isn't enough. For example we want to pass the all the taxonomy terms of the page to the view. The node has the standard %node:term-id for just the primary term. So we need to get the 'taxonomy term' context added into the panel.

Go to the 'Contexts' tab. Here you will see the following screen:



As a note: until you see a line with a specific context in it (not an 'Add context' button next to it), this context isn't actually there (unless it is a standard in built context). The lines with the buttons next to them are simply options to add a context.

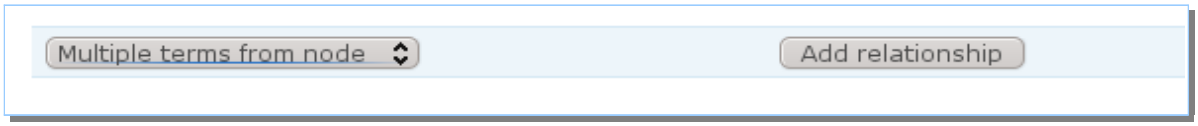
There are 2 categories of context – plain vanilla and relationships.

- Normal contexts are those which don't necessarily have a special relationship with the node – for example a 'user' context (which contains user data). We haven't used the contexts here much.
- Relationship contexts are those which have a special relationship with the node (in this case) – for

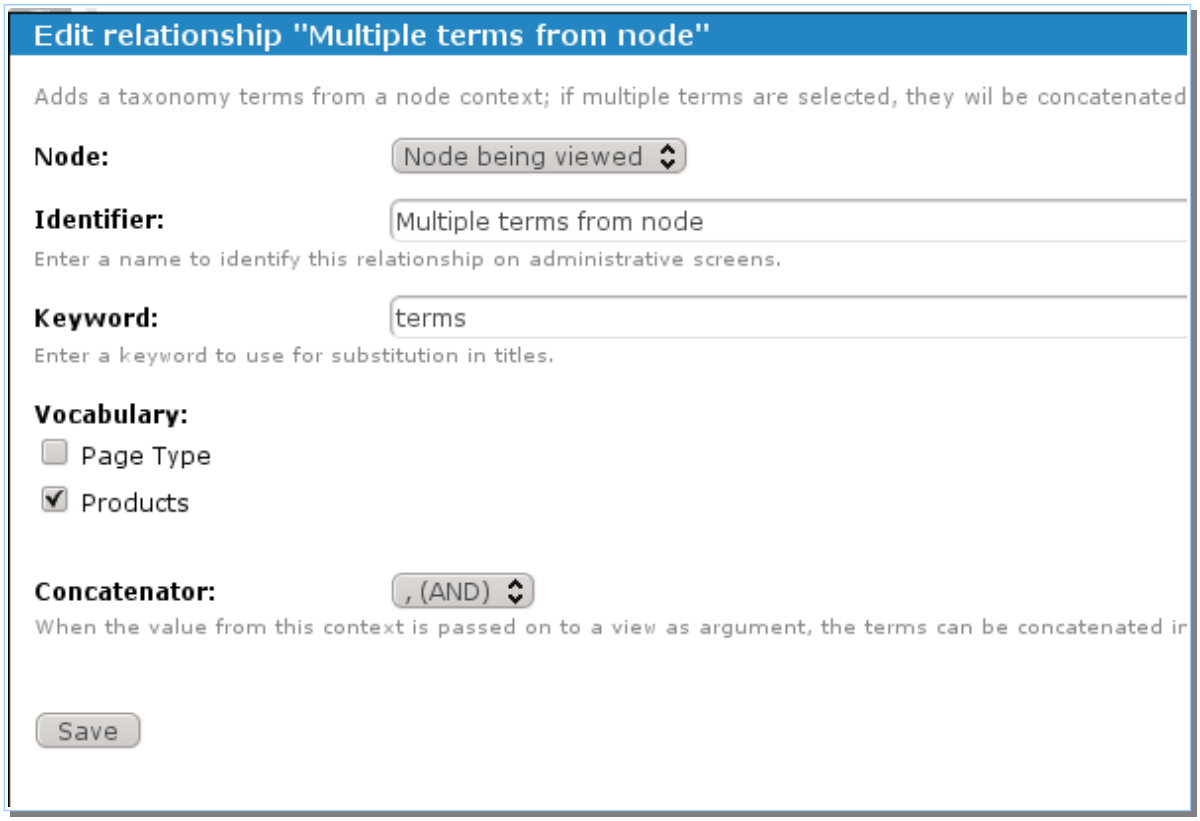
example the taxonomy term which is referenced on that node or the author.

Whichever you add, simply creates a new bit of contextual data to work with. The end result is the same – each is just a context.

We want to have the taxonomy terms related to our node, so we select (under the relationships heading), multiple terms from node (which are all the terms associated with the node)

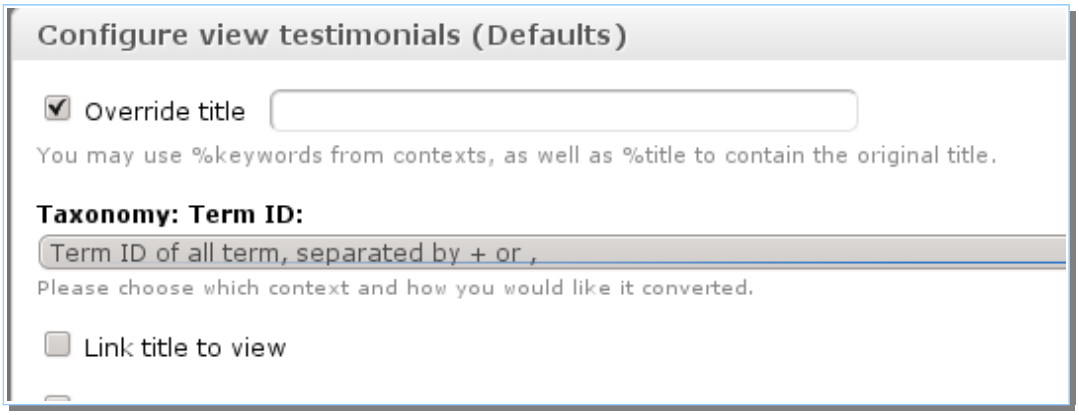


We then click 'add relationship' which just adds this as a context. You get a configuration dialog.



Apart from selecting which vocabulary you want the term data from, you can leave everything else as suggested. The keyword is as defined above (such as %terms:XXX to get a value from this context). Click save.

These values are now available as references (using %terms:XXXX) or in our case to our view. When we go back to our 'content' tab and choose to 'Add content' and add a view (in this case our testimonials view which takes an argument of taxonomy term id), we get this dialog box:



You can see (unlike in the other dialogs) that this has a drop down selection of all the available context data which can be passed into the view. As long as you have the 'CTools Views' module enabled, when you create an 'Argument' in your view, then this dialog box will ask you what to pass into that argument. Here we choose for the 'Term ID of all terms' to be passed in. Therefore when someone goes to a page about 'Studio Photography' which is categorised as such in our taxonomy, then only testimonial which are also categorised that way get shown in this panel.

That's pretty much it – your content is now very dynamic.

Caching

As you can imagine Panels layouts can create lots of queries against your database. However it comes with a fantastic caching feature. Each bit of content you add, you can decide to cache – do this by clicking on the gear by the content and adding simple caching. Do remember though that this content won't change as you edit until the cache is cleared.

Styling your panel

Once you have content flowing into a page, styling it is just back to CSS work. For example in Firebug you can see how panels creates the 2 column layout:-

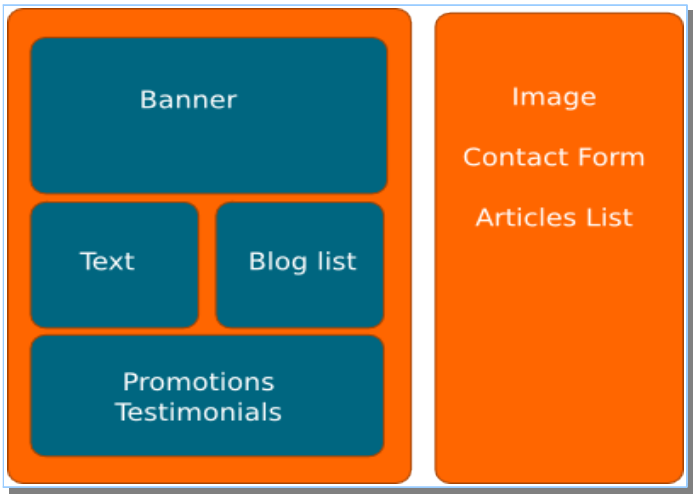
```
<div class="clear-block">
    <div class="panel-display panel-2col clear-block">
        <div class="panel-panel panel-col-first">
        <div class="panel-panel panel-col-last">
      </div>
  </div>
```

```
.panel-2col .panel-col-first {                    twocol.css?g (line 7)
    float: left;
    width: 50%;
}
```

Panels generates the HTML and has some standard stylesheets (2 columns of 50% width). You can override this in your own stylesheets. Or, if it doesn't suit you – you can create your own layout which we'll do next.

Creating your own layout in your theme

In our case we wanted something a bit unusual. 2 main columns and then within the first a brick effect. Like this:-

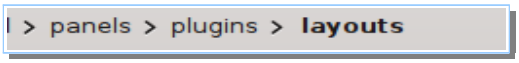


We could do it by creating a 'mini-panel' for the text and blog list in the middle, but we chose the other way. Creating our own.

Panels allows you to easily create your own layouts inside your theme. In order to do this, it's best to copy an example over from the Panels code. Here's what is needed:-

- Copy a layout from panels. It's inside panels > plugins > layouts. We took twocol
- Place this inside your theme in a folder called 'layouts' and give it a different name
- Tell your theme about it by updating you MY_THEME.info file
- Refresh your cache

Step by step, this is what we did



We went to layouts and copied the twocol layout folder to our theme

We then renamed all of the files to twocol-left-twocol (or whatever name you want), and we changed the image file to reflect our layout. That image file is shown in the layout dialogs in Panels.

The *.inc file is used to define this layout. You simply need to put in your relevant values:-

```
<?php

// Plugin definition
$plugin = array(
  'title' => t('Two column with first column two column'),
  'category' => t('Columns: 2'),
  'icon' => 'twocol-left-twocol.png',
  'theme' => 'twocol-left-twocol',
  'css' => 'twocol-left-twocol.css',
  'panels' => array(
    'left-top' => t('Left side top'),
    'left-middle-left' => t('Left side middle left'),
    'left-middle-right' => t('Left side middle right'),
    'left-bottom' => t('Left side bottom'),
    'right' => t('Right side')
  ),
);
```

The $plugin definition (I think) is standard of Chaos Tools and allows you to define the necessary information for this layout. It's just a simple PHP array. You can change the title, icon, theme and css to reflect your layout and layout files.

The **'panels' array** defines each of the regions/areas within the panel. Here we added left-top, left-middle-left, left-middle-right and left-bottom. These become variables (inside $content) used in your *.tpl.php file which is shown next.

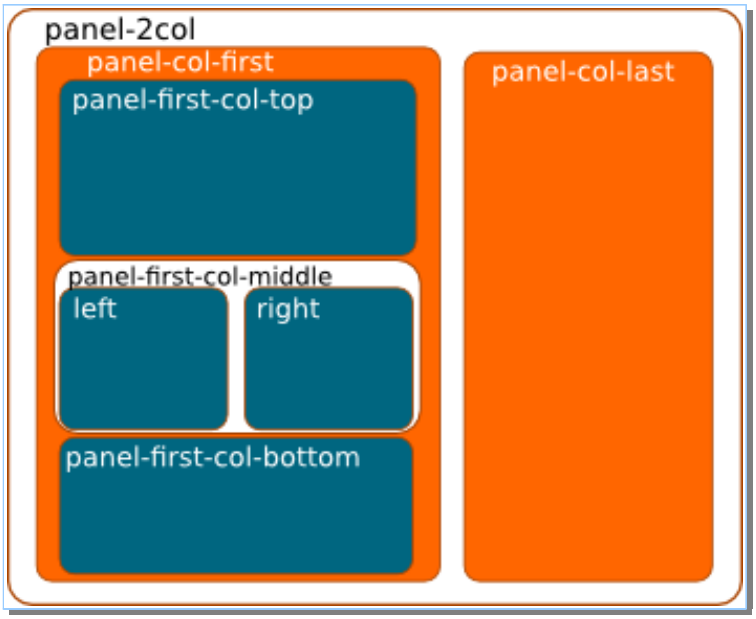
We then edit the layout file which is the twocol-left-twocol.tpl.php file. It looks like this:-

```
<div class="panel-display panel-2col clear-block" <?php if (!empty($css_id)) { print
"id=\"$css_id\""; } ?>>
  <div class="panel-panel panel-col-first">
    <div class="inside">
      <div class="panel-first-col-top"><?php print $content['left-top']; ?></div>
      <div class="panel-first-col-middle">
        <div class="left"><?php print $content['left-middle-left']; ?></div>
        <div class="right"><?php print $content['left-middle-right']; ?></div>
      </div>
      <div class="panel-first-col-bottom"><?php print $content['left-bottom']; ?></div>
    </div>
  </div>

  <div class="panel-panel panel-col-last">
    <div class="inside"><?php print $content['right']; ?></div>
  </div>
</div>
```

This is just the HTML layout of the panel. You can see the variables for left-top, left-middle-left etc. coming in as values inside $content (e.g. $content['left-top']). So using HTML floating DIVs you can create a layout of your choosing with the content coming from the panels.

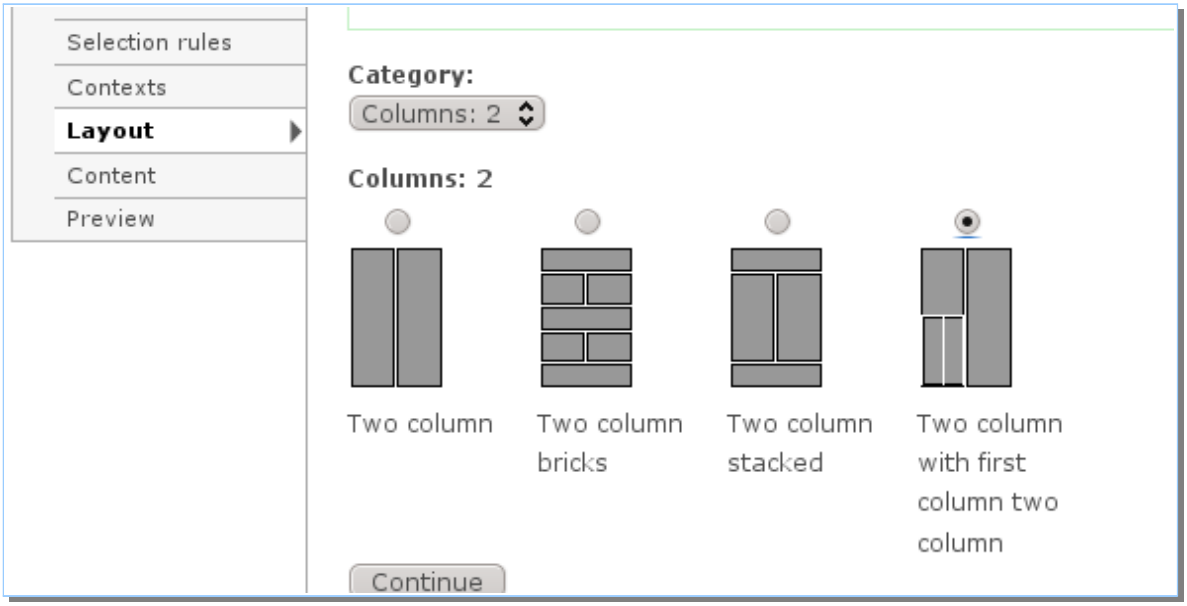
To help you along this is what it looks like visually (with the CSS classes shown in the DIVs above)



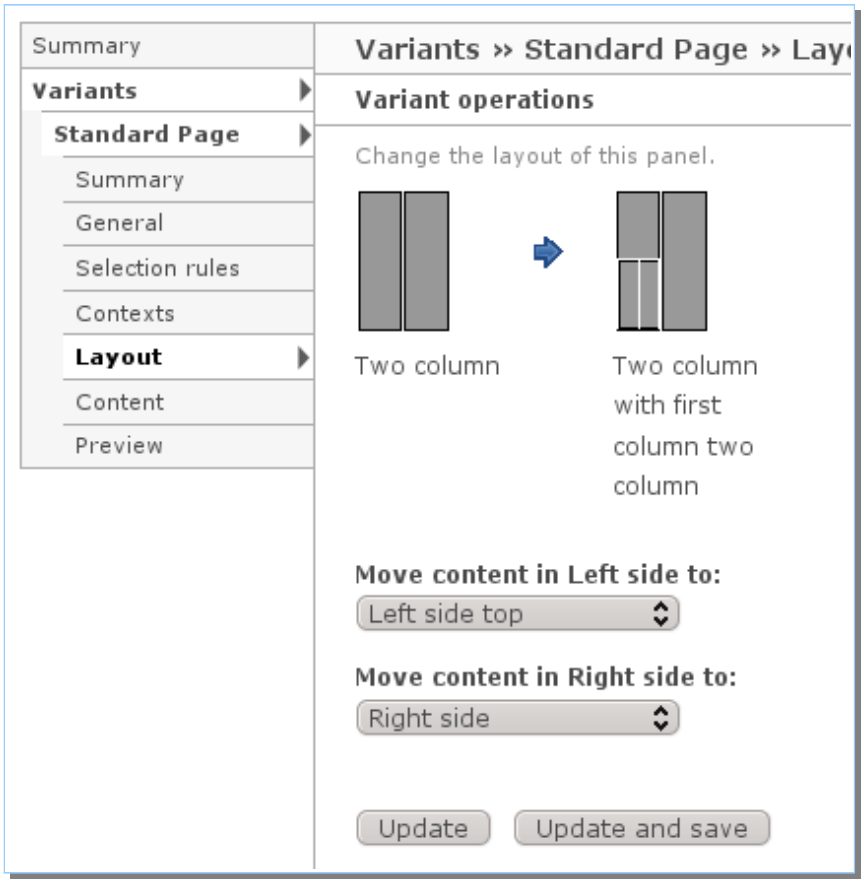
Once you're done, update your MY_THEME.info file to tell panels where to find the layouts

```
; Panels layouts. You can place multiple layouts under the "layouts" folder.
plugins[panels][layouts] = layouts
```

Flush your cache, and you'll see your template right there inside 'layouts' (inside the panel variant tabs as described above):-



You can select it and click continue. Panels will take all your current content (which was in Two column) and ask you where you want it in your new layout:-



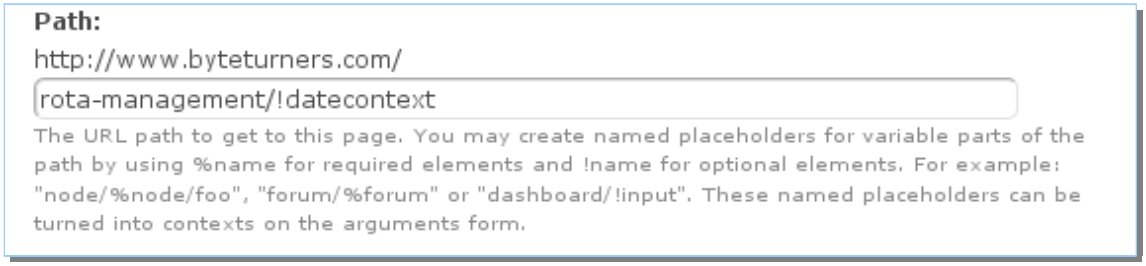
You can always go into Content afterwards and move it around – dragging and dropping from any content area to any other content area.

Passing in arguments from the URL

Sometimes, instead of your context coming from data within the panel, you'll specifically want to pass in arguments via your URL. So for example if I want to pass in a specific date to my page (or calendar), I could put http://www.mydomain.com/path/to/page/2011-12-01 (where the last argument is my date).

In order to have this date passed into my panel as a context, I need to specifically tell the panel that I have an argument coming in and that it should be a context. This is done in Settings > Basic (on a normal panel page. I'm not sure if this is possible for templates).

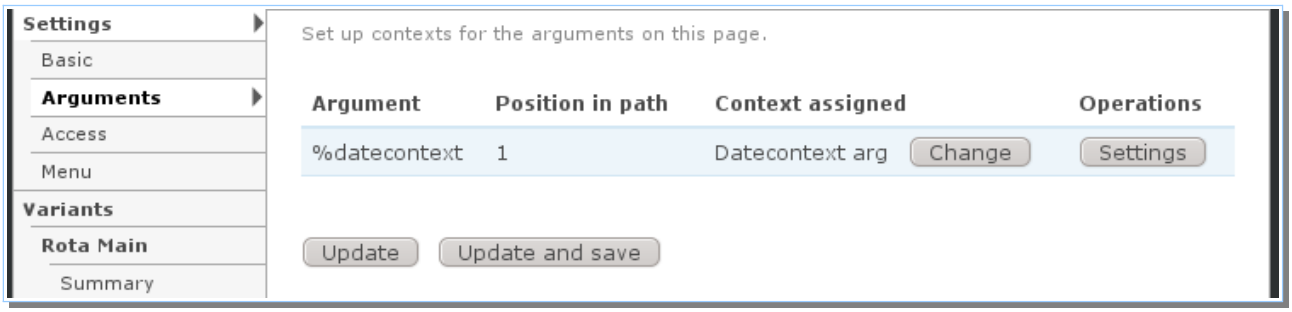
Here you will find a path to enter



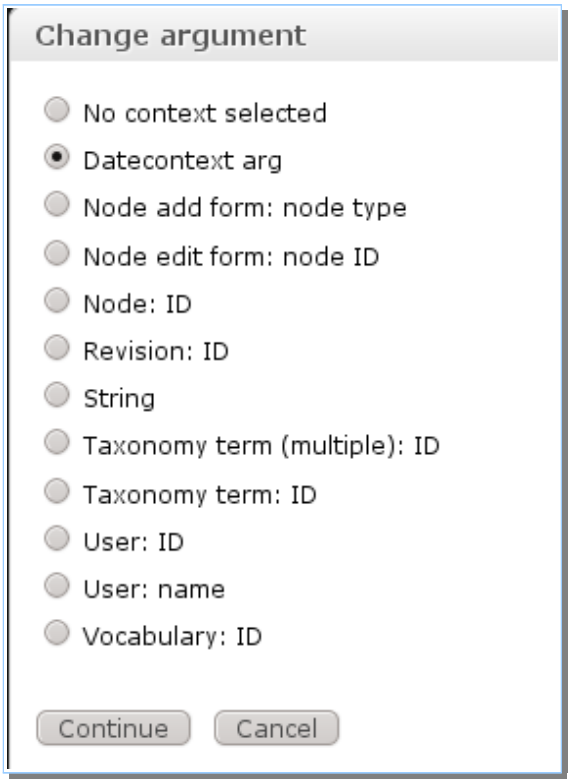
You enter your 'arguments' as a name prepended by % or ! - if you want the date to be required in the URL you us %datecontext. If it is not required, then use !datecontext.

Once you've saved this pane, then you can step over to the 'Arguments' pane where you can tell Panels

which context to assign this argument to. Click Change to select the assigned context.



You will get a list of available contexts, which you can select. Each context may or may not have a settings form which you can also complete with further related information. (See below for programming your own context arguments).



Creating your own contexts

Every so often you find yourself wanting data passed around your panels which you can't easily do out of the box. We had one example with Calendars (another fantastic module), where we wanted to pass dates around in a specific format. To do that we were able to create our own context for 'date'. You will need to write a module to do this and tell panels you have a context for it.
You need to:-

- Implement hook_ctools_plugin_directory (to tell it where the plugins are)
- Create a plugins directory and inside it (optionally) a contexts directory
- Define the context through code (as described below).

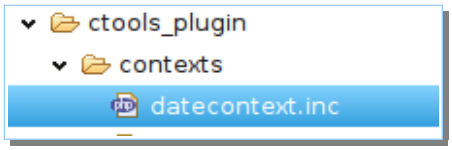
To implement hook_ctools_plugin_directory

```
function MYMODULE_ctools_plugin_directory($module, $plugin) {
  if (!empty($plugin)) {
```

```
    return "ctools_plugin/$plugin";
  }
}
```

This says that we have stuff inside the directory ctools_plugin/contexts inside our own module. $plugin is the type of plugin being searched for, which is 'contexts' in this case. You don't have to do it exactly this way, but this is how Panels examples does it.

We then create our directory structure and put a file 'datecontext.inc' in there, which represents the 'datecontext' we want to build.



datecontext.inc needs this following:-

- $plugin array which is used to pick up all the details about this plugin
- functions (which are defined in the plugin array).

Our plugin array looks like this

```
$plugin = array(
  'title' => t("Date context"),
  'description' => t('A single date which is derived from PHP.'),
  'context' => 'olamalu_rota_context_create_datecontext',  // creates context
  'context name' => 'datecontext',
  'settings form' => 'datecontext_settings_form',
  'keyword' => 'datecontext',

  // Provides a list of items which are exposed as keywords.
  'convert list' => 'datecontext_convert_list',
  // Convert keywords into data.
  'convert' => 'datecontext_convert',

  'placeholder form' => array(
    '#type' => 'textfield',
    '#description' => t('Enter data understood by create_date'),
  ),
);
```

The title & description are self-explanatory. The rest is

- context: a function in the file which creates the context. It must return a context object
- context name: the context name used in the system
- settings form: a function which creates a standard drupal $form array
- keyword: as seen above
- convert list: provides a list of sub key-words which are used by the convert function below
- convert: converts the context into a string which is used in panel titles etc. This uses options from the convert list. For example we want the output of a date by 'day' (2011-01-01), 'month' (2011-01) or 'year' (2011).

Code to create the context:-

```
function olamalu_rota_context_create_datecontext($empty, $data = NULL, $conf =
FALSE) {
      $context = new ctools_context('datecontext');
      $context->plugin = 'datecontext';

      if ($empty) {
      return $context;
      }
```

```
      $context->data = new stdClass();
      if ($conf) {
            $context->data->date = check_plain($data['date_setting']);
      } else {
            $context->data->date = 'now';
      }
      return $context;
}
```

Here we create an object 'ctools_context' and then depending on the settings form (which takes an argument about the date) we create a date string inside $context->data. This string is used by the PHP create_date function.

Code to create the settings form:-

```
function datecontext_settings_form($conf, $external = FALSE) {
  if (empty($conf)) {
    $conf = array(
      'date_setting' => 'now',
    );
  }
  $form = array();
  $form['date_setting'] = array(
    '#type' => 'textfield',
    '#title' => t('Setting for Date'),
    '#size' => 50,
    '#description' => t('A setting understood by create_date'),
    '#default_value' => $conf['date_setting'],
    '#prefix' => '<div class="clear-block no-float">',
    '#suffix' => '</div>',
  );
  return $form;
}
```

Just like any Drupal form we allow the user to enter the string understood by PHP create_date – or if none we just take 'now'.

Code to create the sub-keyword list:-

```
function datecontext_convert_list() {
  return array(
    'day' => t('Day as Y-m-d'),
    'month' => t('Month'),
    'year' => t('Year'),
  );
}
```

Here we list out the options for converting date to day, month or year values. This is simply a list given to the person creating the panel to choose from when they are working with our context.

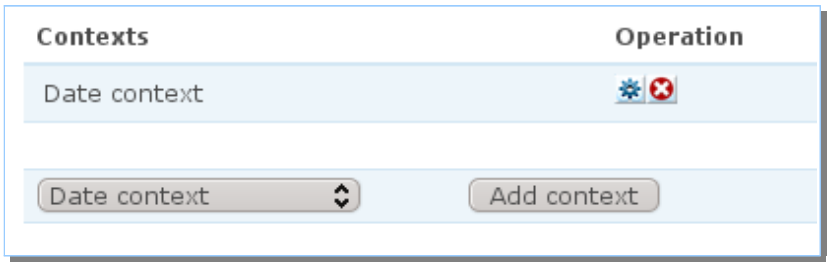
Code to create the output of the conversion list:-

```
function datecontext_convert($context, $type) {
  switch ($type) {
    case 'day':
      return date_format(date_create($context->data->date), 'Y-m-d');
    case 'month':
      return date_format(date_create($context->data->date), 'Y-m');
    case 'year':
      return  date_format(date_create($context->data->date), 'Y');
  }
}
```

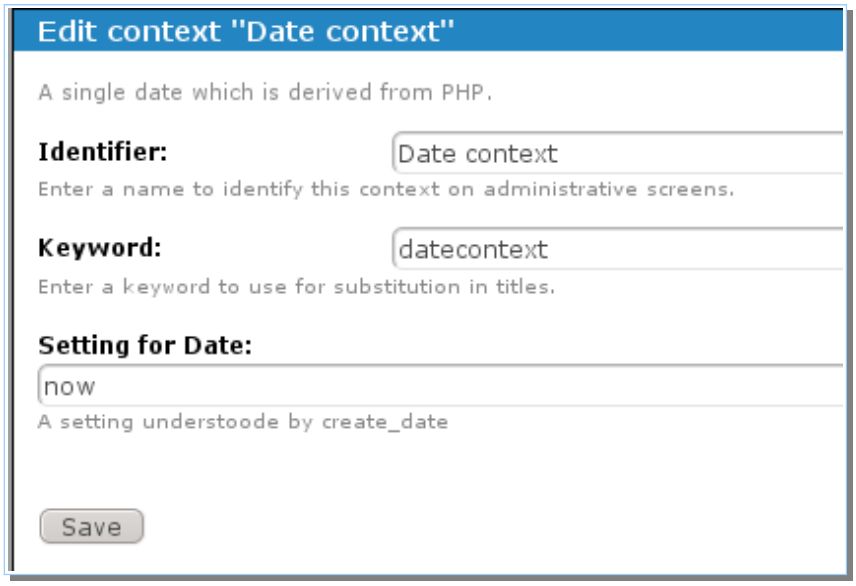
And here based on the selections above, the value is converted. Note the switch is on the same values as in

the options above.

And voila! This context is now available in our panels. (Remember to flush the cache).



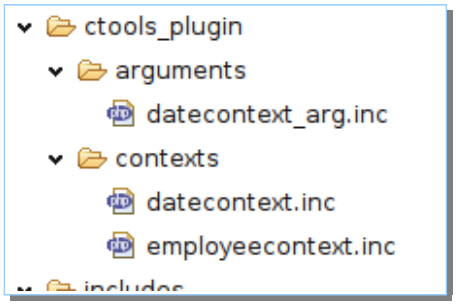
And when you click edit or add, you'll see the settings form we created



These examples were figured out from the Ctools Plugin Examples module inside the core Ctools package.

Arguments passed into your own contexts

If you want URL argument data passed into your contexts, you need to create a specific plugin for that argument and then link it to your context. Our new folder layout looks like this.



We don't need to change our MYMODULE_ctools_plugin_directory($module, $plugin) – because we are getting the value of $plugin as 'arguments' , which if you see above points to this directory.

We now create the necessary code for datecontext_arg.inc

```
<?php

/**
 * Plugins are described by creating a $plugin array which will be used
 * by the system that includes this file.
 */
$plugin = array(
  'title' => t("Datecontext arg"),
  // keyword to use for %substitution
  'keyword' => 'datecontext',
  'description' => t('Creates a date context from the argument'),
  'context' => 'olamalu_rota_context_create_datecontext_arg',
  // 'settings form' => 'simplecontext_arg_settings_form',

  // placeholder_form is used in panels preview, for example, so we can
  // preview without getting the arg from a URL
  'placeholder form' => array(
    '#type' => 'textfield',
    '#description' => t('Enter the datecontext argument'),
  ),
);

/**
 * Get the simplecontext context using the arg. In this case we're just going
 * to manufacture the context from the data in the arg, but normally it would
 * be an API call, db lookup, etc.
 */
function olamalu_rota_context_create_datecontext_arg($arg = NULL, $conf = NULL,
$empty = FALSE) {
  // If $empty == TRUE it wants a generic, unfilled context.
  if ($empty) {
    return ctools_context_create_empty('datecontext');
  }
  // Do whatever error checking is required, returning FALSE if it fails the
test
  if (empty($arg)) {
    return FALSE;
  }
  try {
      date_create($arg);
  } catch (Exception $e) {
      return FALSE;
  }

  return ctools_context_create('datecontext', $arg);
}
```

(You'll see we copied this from the ctools plugin examples).

Basically stepping through this code – the first array defines the plugin ($plugin). The 'context' key points to the function olamalu_rota_context_create_datecontext_arg, which is where the context data is created.

There's not much to do, as we already have a context (datecontext created above), so we simply use ctools functions to create our context using that. Note that we need a date to be created from the argument so we don't validate (i.e. we return FALSE) if we can't create a date.

Flush the cache and you should be able to assign arguments to this context.

Writing your own content types

Content types are the snippets of content which you put into your panels. These are big things like views, CCK elements and widgets like the 'contact form' provided by the Contact module.

Not all modules integrate with panels – in other words, they don't necessarily provide all their bits of content as snippets to panels. In this case you can write your own plugin to do this.

Lets take a really simple example. There is in fact a 'logo' content type in the main ctools/panels offering, but in our case we like to wrap it in <h1> tags and put a few keywords as the alternative text. Normally you'd do that in your page.tpl.php – but if you are inserting the logo in a panel which is completely in charge of the page, then you can't do that. So in this case this code is copied from the ctools > plugins > content_types > page folder – the file is page_logo.inc.

First you need to make sure you create a folder called 'content_types' inside your ctools folder and implement hook_ctools_plugin_directory (see the context stuff for that). If this is in the same module as the contexts, all you need to do is create the folder.

Then you create your file called 'MYPLUGIN.inc' where MYPLUGIN is the name of your plugin. Ours is called 'olamalu_extra_logo.inc'.

```
$plugin = array(
  'title' => t('Olamalu adapted logo'),
  'single' => TRUE,
  'icon' => 'icon_page.png',
  'description' => t('Add the logo trail as content - ours is wrapped in H1.'),
  'category' => t('Page elements'),
  'render last' => TRUE,
  // Constructor.
  'content_types' => array('olamalu_panels_extra_logo'),
  // Name of a function which will render the block.
  'render callback' => 'olamalu_panels_extra_logo_render',
);

/**
 * Output function for the 'olamalu_panels_extra_logo' content type.
 *
 * Outputs the logo for the current page with the site name and slogan as title.
 */
function olamalu_panels_extra_logo_render($subtype, $conf, $panel_args) {
  $logo = theme_get_setting('logo');
  if (!empty($logo)) {
    $block = new stdClass();
    $title = check_plain(variable_get('site_name', "Home")) . ' '.
check_plain(variable_get('site_slogan', ""));
    $image = '<img src="' . $logo . '" alt="' . $title . '" />';
    $block->content = '<h1>' . l($image, '', array('html' => TRUE, 'attributes'
=> array('rel' => 'home', 'id' => 'logo', 'title' => $title))) . '</h1>';
  }

  return $block;
}
```

The $plugins array defines this according to the ctools pattern – you need a title and description. The category is the tab on the side of adding content types into which your widget will go. Ours goes into the 'page elements' with the other page related stuff. The content_types is your content type unique name and

the render callback is the function which will return a rendered HTML string.

This is implemented just below. In our case it simply takes the logo, gets the site slogan (where we store keywords) and wraps the logo in H1 tags with the alternative text being the site slogan. It sets the $block->content to that HTML output and returns the $block.

Flush the cache and you'll see your content type!

Little note: If you really want to see how plugins are programmed in lots of ways, then the best place is to look in the ctools code. Go to the ctools > plugins folder – in there you will see all the plugins in all the different guises.

Advanced Information and Use Cases

Ctools Plugin array

Every ctools plugin is defined by an array – so if you're building your own content type, arguments, contexts etc. you'll find yourself defining this array at the beginning of your plugin

```
$plugin = array(
      'key' => value,
}
```

There are multiple values and variables and so this is just the beginning of the documentation of ones we have found

Key	Values explained	Plugin Type	E.g.
title	Title of the plugin – use t('XXXX') to translate		t('My title')
description	Description of the plugin – use t('xxx') to translate		t('My desc')
icon	The related icon shown to the user (*.png file)	content_type	form_icon.png
category	The category shown to the user	content_type	
render callback	The callback function to use in order to render this content type. See content.inc->ctools_content_render.	content_type	
content_types	The constructor function for the content type	content_type	
defaults	The default context.		array()
edit form	The configuration form function	content_type	
all contexts	TRUE or FALSE. Tells ctools if all contexts should be passed to this plugin. See content.inc->ctools_content_select_context	content_type	TRUE
required context	Tells ctools which contexts are required by this plugin. Should be a ctools_required_context object	content_type	new ctools_context_required(t('Term'), array('term', 'terms')) This creates a required context called Term – with an array of context types to match (term, terms)
render last	TRUE or FALSE		
single	TRUE or FALSE		

Use Case: Putting administration forms into Panels

Whilst developing a new module which had quite a few administration forms for it (there are a bunch of our

own tables we've added to the site install), we wondered if there was a way to create the entire administration interface via panels.

In order to do that, we needed to get all the administration forms we created into a form acceptable to panels – which could have meant creating a content type for each one. However, we found another more generic way of doing that. By exposing a hook which looked for definitions of all forms which could be shown in a panel, we were able to create just one form content type which can be inserted into a panel.

Document developed by Olamalu Web Development

We're a small web development company based in Oxfordshire, England. We work exclusively on Drupal for our websites. We'd like to see it as the defacto standard for small and medium sized business websites. Having programmed in Java and PHP for a while, we're always amazed at how flexibly and powerfully Drupal and modules like Panels are architected.

For the moment we'll keep an updated version of this guide on our website at http://www.olamalu.com/content/our-guide-drupal-panels until we can find the best way to make it generally available.

Version control

Version	Changes
1.0	Initial document
1.1	Added arguments as contexts and programming context arguments
1.2	Added writing content types (simple)
1.3	Added panels plugin definitions and advanced use cases – flexible form
1.4	Started incorporating feedback from issue log and rewriting for automatic creation into advanced help.