Contents

[1. Introduction to Layouts 4](#_Toc445330798)

[Defining Layouts 4](#_Toc445330799)

[Orchard and Layouts 4](#_Toc445330800)

[Layout and Elements 5](#_Toc445330801)

[When to use Orchard.Layouts? 6](#_Toc445330802)

[Option 1 - Direct HTML manipulation 7](#_Toc445330803)

[Option 2 - Widgets and Zones 7](#_Toc445330804)

[Option 3 - Content Fields and Placement.info 8](#_Toc445330805)

[Enter Orchard.Layouts 9](#_Toc445330806)

[Where did the Body Part go? 10](#_Toc445330807)

[What happens to my existing site and its contents when upgrading to Orchard 1.9? 10](#_Toc445330808)

[The Nature of Elements 11](#_Toc445330809)

[Does Orchard.Layouts work with grid systems such as Bootstrap? 11](#_Toc445330810)

[Summary 12](#_Toc445330811)

[2. First Look 13](#_Toc445330812)

[The Main Players 13](#_Toc445330813)

[The Layouts Feature 13](#_Toc445330814)

[The Layout Part 13](#_Toc445330815)

[Elements 13](#_Toc445330816)

[The Layout Editor 14](#_Toc445330817)

[Working with the Layout Editor 16](#_Toc445330818)

[Element Editor Controls 16](#_Toc445330819)

[Keyboard Support 19](#_Toc445330820)

[Moving Elements within its Container 22](#_Toc445330821)

[Moving Elements across Containers 22](#_Toc445330822)

[Re-sizing Columns 22](#_Toc445330823)

[Layouts on the Front-end 22](#_Toc445330824)

[Summary 23](#_Toc445330825)

[3. Meet the Elements 24](#_Toc445330826)

[Element Categories 24](#_Toc445330827)

[Layout 26](#_Toc445330828)

[Grid 26](#_Toc445330829)

[Row 27](#_Toc445330830)

Introduction

Welcome to Mastering Orchard Layouts, a book that will take us on a journey through the wonderful and exciting world of the Orchard Layouts module, which was first released as part of Orchard 1.9.

The book is divided in three parts.

**Part 1** introduces the Layouts module and looks at it from a user's perspective.

**Part 2** goes a step further and looks at the various shapes and templates from a theme developer's perspective.

**Part 3** takes a deep dive and looks at the module from a developer's point of view. Here we'll learn about extensibility and APIs and how to create custom elements and element harvesters.

I hope you will enjoy reading this book as much as I had writing it, and that the knowledge you will find here useful in your Orchard Projects.

# Introduction to Layouts

With the release of Orchard 1.9 came a new module called Orchard.Layouts. Before we try it out to see what we can do with it, let's first give it some context to get a better understanding of why it was created in the first place, what problems it solves, and, equally important, what problems it won't solve.

## Defining Layouts

Layouts are everywhere. You find them not only in newspapers, magazines, and the book you're currently reading, but you find them just about anywhere, like in the office and your living room. Cities, planets and the universe, they all have a layout.

Elements, objects and shapes that are placed in a particular position relative to each other, are said to be part of a layout. In other words, a layout is an arrangement of elements. That's a great definition for sure, but how does this relate to Orchard you ask? Let's find out.

## Orchard and Layouts

Orchard, unsurprisingly enough, is about building and managing web sites that consist of web pages. Within the context of a webpage, a layout is the arrangement of visual elements on a page. Those visual elements can include things such the site's navigation, side bar and the site's content. The site's content itself, too, can have a layout. For example, two blocks of text that appear next to one another are said to be laid out horizontally.

And that is where the Layouts module comes in: it enables content editors to create layouts of contents.

Now, technically speaking, a theme developer can choose to setup their theme in such a way that the entire layout is controlled by the Layouts module. But, practically speaking, this is probably not the best use of the module in its current form. One reason is the fact that layouts are provided through a content part (the Layout Part), which means layouts created by the Layouts module can only be applied to content items. So although you could very well add a Menu element to a page content item, not all pages in Orchard are provided by content items.

For example, the Login screen is provided by a controller. If the site's main navigation is implemented as a layout element, the main navigation would disappear as soon as a page is displayed by something other than a content item.

As the Layouts module evolves over time, new site editing paradigms may using the Layouts module emerge, but until then, we will focus on how to use the Layouts module from a content editor's perspective. Which, as you'll see, is quite impressive.

## Layout and Elements

When you enable the Layouts feature provided by Orchard.Layouts, a new content part called Layout Part is added to the list of available content parts, and is attached to the Page content type by default. It is this Layout Part that enables content editors to visually arrange elements on a canvas, effectively enabling them to create layouts of contents.

These elements are a new type of entity in Orchard and represent the objects you can place on a canvas. An example of these elements is the Html element, which enables the user to add content. Another example is the Grid element, which enables the user to create a layout by adding Row and Column elements to it. The Column element is a container element into which you can add other elements, such as Html and Image elements. You can imagine that using these elements, it is easy to create a layout of contents. It is this capability that gave Orchard.Layouts its name.

## When to use Orchard.Layouts?

From what you have read so far, the answer to the question of when to use the Layouts module may seem obvious: whenever you need to create a layout of content, use the Layouts module. But you may be wondering that surely, this was possible before we had this module? Well, yes, but that was a very hard thing to do. Let me explain.

Let's say we have a web page with content that consists of two paragraphs as seen in figure 1.1.

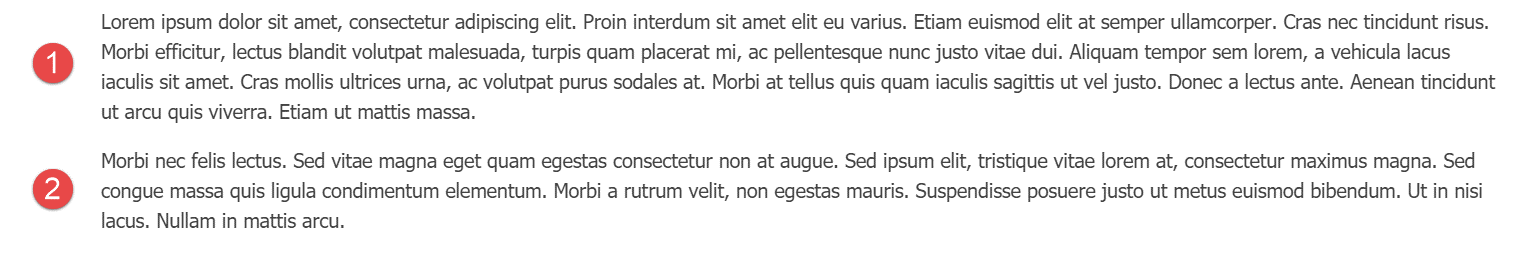


Figure 1‑1 – Two paragraphs, vertically stacked.

Now, let's say that we want to display those two paragraphs laid out horizontally instead, as seen in figure 1.2.

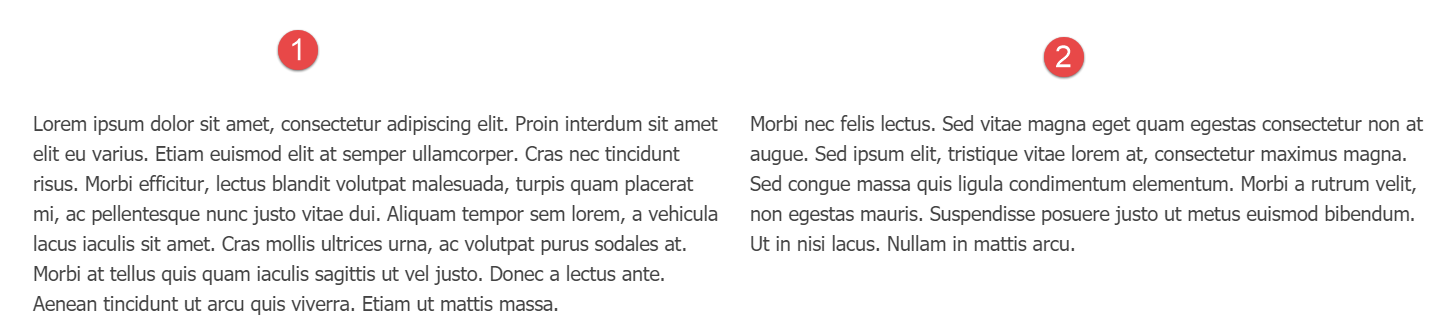


Figure 1‑2 – The same two paragraphs, horizontally laid out.

Before we see how to achieve that with the layouts module, let's explore our options before Orchard.Layouts.

### Option 1 - Direct HTML manipulation

One option is to edit the HTML source of the Body Part content and leverage your HTML skills by adding an HTML table element, or maybe even using Bootstrap's Grid CSS classes and apply them on <div> elements. Although that would certainly work, it is far from ideal, because it would require the content editor to know about HTML tables and how to work with them. For a simple two-column layout for a body of text this may not be that big of a deal, but it becomes icky real fast when working with more complex layouts.

### Option 2 - Widgets and Zones

Another option is to provide two zones, let's say *AsideFirst* and *AsideSecond*. The theme’s Layout.cshtml view renders these zones horizontally. You would then simply add an Html Widget to both zones, and the two Html widgets would appear next to each other. Although this approach works, a major disadvantage is that now the textual content becomes unrelated to the content item itself, since you are using widgets. To manage the content on this page, you have to go to the Widgets screen, create a page specific layer, and add two widgets. Now imagine you have to do that for 20 pages. That means 20 widget layers, 2 HTML widgets per layer, and 20 Page content items with no contents. And this is just two columns. Imagine you have other types of layouts, for example one row with two columns, another row with 4 columns, and perhaps rows with one column taking up 2/3 of the row and a second column 1/3 of the row. Crazy. Allowing this level of freedom to the content editor user would easily end up in a maintenance nightmare.

There is a way to associate widgets with content items directly by taking advantage of a free gallery module called IDeliverable.Widgets, which allows you to associate widgets with your content items directly. Although this is better than having to create a layer per page, it is still not ideal.

### Option 3 - Content Fields and Placement.info

 Yet another option is to create various content types, where a content type would have multiple content fields.

For example, we could create a new content type called TwoColumnPage with two TextField fields. The theme would use Placement.info to place each field into two horizontally laid out zones.

Although this option is (arguably) better than the previous option using widgets, there is still the limitation of freedom when you want to introduce additional layouts. Not to mention the fact that we're now basing the content type name on what it looks like, rather than its semantic meaning. It is not pretty.

## Enter Orchard.Layouts

With the inclusion of the Layouts module, a fourth option appeared. And a much better one too!

With Orchard.Layouts, creating a two-column layout could not be simpler. Simply add a Grid element with a single Row and two Column elements to the canvas, add some content elements, and you're done. No need for HTML editing, no additional zones, no widgets and layers, and no additional content types.

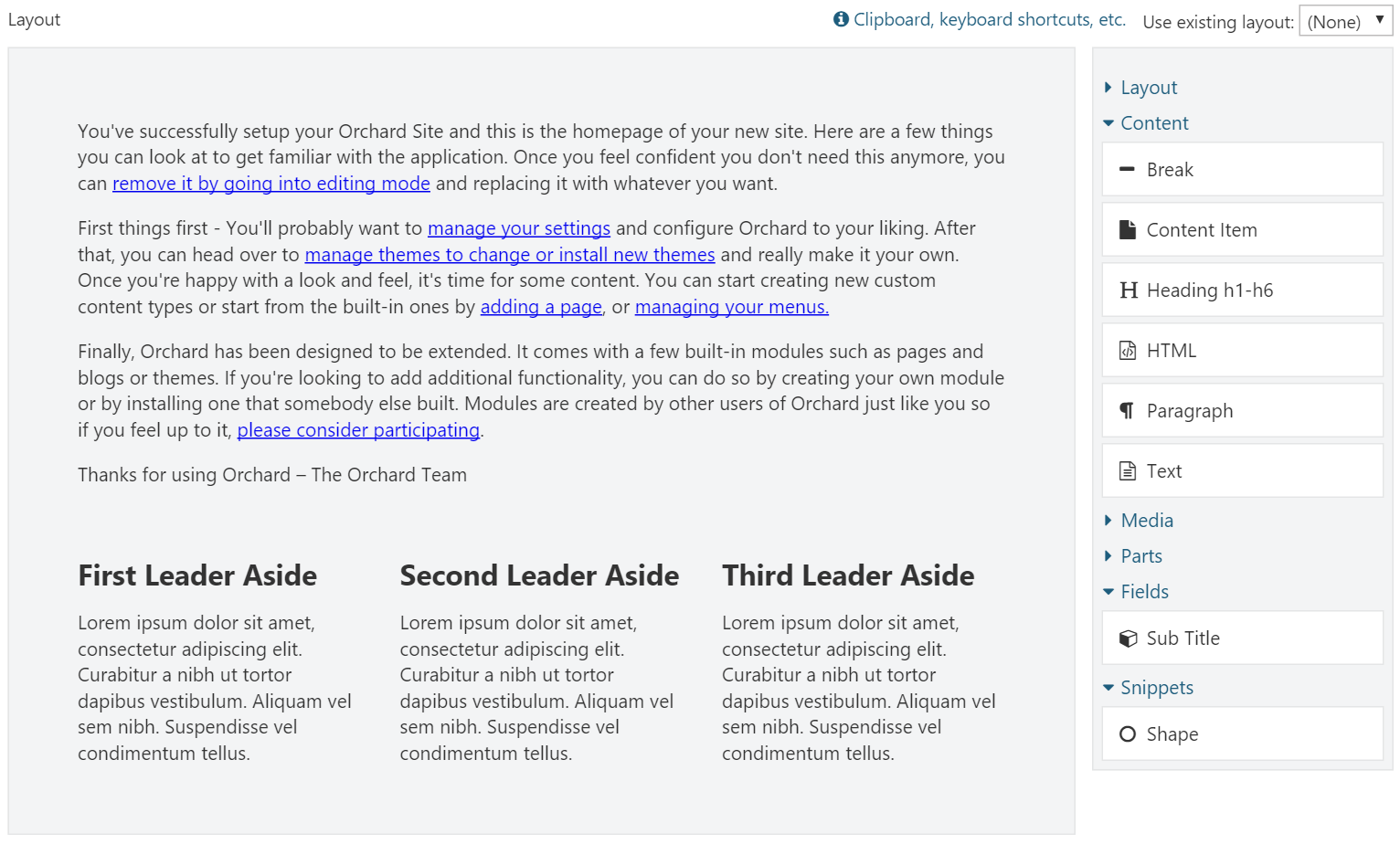


Figure 1‑3- The Layout Editor.

The layout editor consists of a design surface called the *canvas* and a toolbox containing elements that the user can drag and drop onto the canvas.

To sum it up, thanks to the Layouts module, we:

* No longer necessary to create page-specific layers and widgets to achieve complex layouts of contents.
* No longer necessary to create specific content types just for supporting multiple layouts.
* Have an easy way to create various layouts of content.

## Where did the Body Part go?

When you install Orchard 1.9 or later for the first time and have a look at the Page content type, you will notice that it doesn’t have the Body Part anymore. Instead, you will see the new Layout Part attached. However, the Body Part is still a happy citizen within the Orchard, and will remain as such. The Layout Part simply serves a different purpose, namely to enable the user to layout pieces of contents. The Body Part is great when all you need is an editable body of text. Blog Posts are a great example where I would rather use the body Part instead of the Layout Part, because all I want to do there is simply start writing content without having to first add an Html element to the canvas. In the end, it's all about choice and being able to pick the right tool for the job.

## What happens to my existing site and its contents when upgrading to Orchard 1.9?

If you're worried about your existing content, fear not. When you upgrade your site to 1.9 or beyond, the Layouts feature will not be automatically enabled. And even when you enable the feature yourself, it will not change your content type definitions. If you do want to use Layouts on existing Orchard installations that have been upgraded to the latest codebase, you will simply have to enable the Layouts feature and attach the Layout Part manually.

## The Nature of Elements

With the Layouts module came a new type of entity called Element. Unlike Widgets, Elements are not content items, but are, quite simply, instances of the Element class. Elements can contain other elements, and this is how layouts emerge.

The hierarchy of elements are stored using the *infoset* storage part of the content item, implemented via the Layout Part. This means that whenever a content item is loaded with the Layout Part attached, the elements are loaded all at once, unlike Widgets, where each widget is loaded individually.

Similar to content items, or more accurately, content parts, Elements have their own drivers, which decouples element data from element behavior. This pattern is borrowed from the content part and content field system that also leverage drivers.

## Does Orchard.Layouts work with grid systems such as Bootstrap?

Many websites today use CSS grid frameworks such as Bootstrap. These grid systems enable web designers to layout visual components onto a grid that is made up of rows and columns. So, you may be wondering whether the Layouts module plays nice with such grid systems. As it turns out, this scenario is well-supported. The Grid, Row and Column elements map nicely to Bootstrap's **container**, **row** and **col-md-\*** CSS classes. You will have to override the shape templates for these elements in your theme so you can modify the CSS classes to use. We'll look into this in detail in chapter 10.

## Summary

In this chapter, I introduced you to the new Layouts module, what it is for and why we need it.

Orchard.Layouts enables users to arrange elements of various types onto a canvas.

We explored what problem the Layouts module solves and when to use it. Where we had to resort to rather cumbersome solutions before, Layouts makes it a breeze to create all sorts of content layouts.

In the next chapter, we'll have a closer look at Orchard.Layouts from a user's perspective, and see how to actually use it.

# First Look

In this chapter, we'll take a tour through the Layouts module and see how it works from a user's perspective.

## The Main Players

First off, let’s go over some of the main concepts that are provided by the Layouts module and what their role is.

### The Layouts Feature

When you setup a new Orchard 1.9 or later installation with the Default recipe, the Layouts feature will be enabled by default. Enabling this feature will cause a new part called Layout Part to be made available.

As mentioned before, one noticeable difference between Orchard 1.9 and previous versions is that the Page content type will have the Layout Part attached instead of the Body Part.

### The Layout Part

It is this Layout Part that we are interested in. It provides a layout editor consisting of a canvas and a toolbar with available elements that the user can add to the canvas.

### Elements

Elements are a new concept in Orchard. They are visual components that contain data and provide behavior. Elements can contain other elements, which is how you can create layouts, as we'll see shortly.

Out of the box, there are currently seven categories of elements:

* Layout
* Content
* Media
* Parts
* Fields
* Snippets
* UI

It's all lovely stuff, and we'll get to know all of the available elements in the next chapter.

### The Layout Editor

The Layout Editor is the component that enables the user to add elements to a canvas, using the Grid, Row and Column elements to create layouts.

The editor consists of two main sections: the canvas (1) and the toolbox (2).

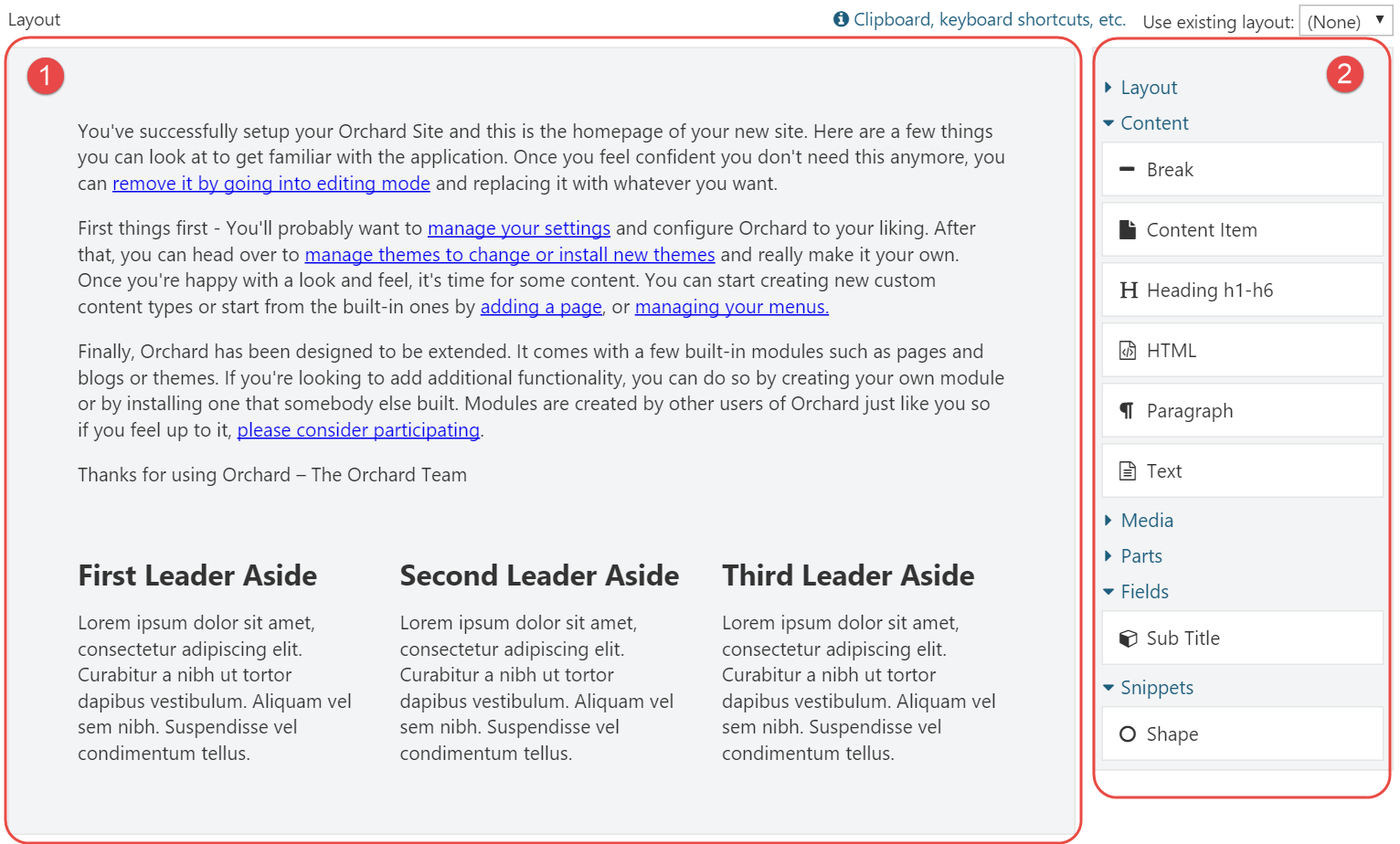


Figure 2‑1 – The Layout Editor consists of the canvas (1) and the toolbox (2).

The canvas is the area onto which you place elements that are available from the toolbox.

The canvas itself is an element of type Canvas, and is the root of the tree of elements.

The toolbox is a repository of all available elements in the system, grouped per category. Elements are bound to Orchard features, which means that other modules can provide additional element types.

The user places elements from the toolbox onto the surface by using drag & drop. If the selected element has an editor associated with it, a dialog window presenting the element's properties will appear immediately when it's dropped onto the canvas.

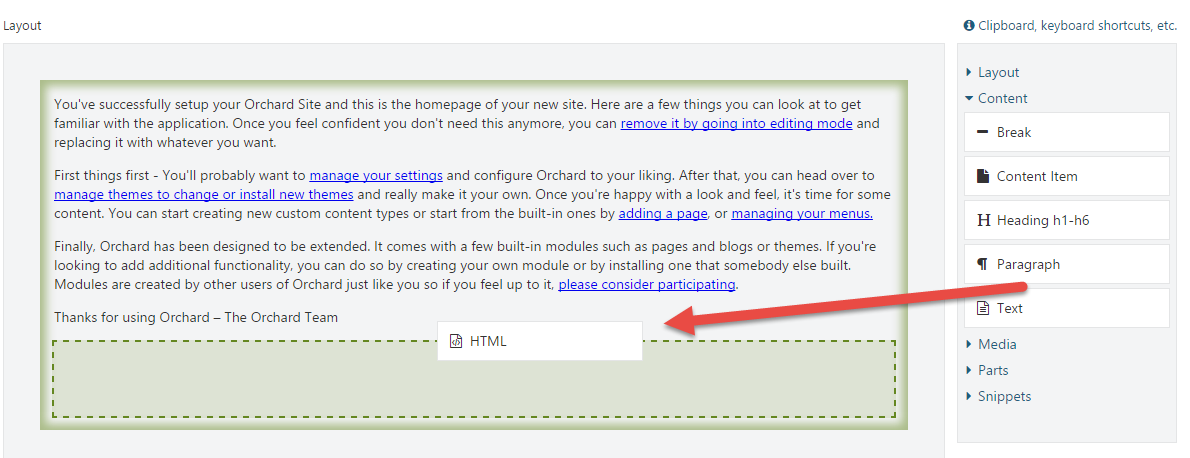


Figure 2‑2 – The user drags and drops elements from the toolbox to the canvas.

## Working with the Layout Editor

Let’s have a look at the various ways we can interact with the layout editor and the elements.

### Element Editor Controls

Depending on the element being selected, the user can perform certain operations on that element. These operations are represented as little icons as part of a mini toolbar that becomes visible when an element is selected. Common operations are Edit, Edit Properties, Delete, Move up and Move down. More specific operations are Distribute columns evenly and Split column, which apply to Row and Column elements, respectively.

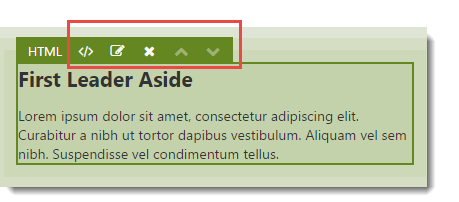


Figure 2‑3 - Each element has a toolbar to control properties of and perform operations to the element.

The button with the  icon is probably the most-commonly used one, as it launches a dialog window that enables the user to configure the element. The icon right next to it () provides a dropdown menu with a list of configurable properties common to all elements. These properties are:

* HTML ID
* CSS Classes
* CSS Styles
* Visibility Rule

The first three properties are rendered onto the HTML tags when an element is rendered. The Visibility Rule determines whether or not the element should be displayed at all. I will have more to say about Visibility Rules in chapter 8.

The following table lists the complete set of keyboard shortcuts.

|  |  |  |
| --- | --- | --- |
| **Icon** | **Shortcut** | **Description** |
|  | Enter | Launches the element specific editor dialog. |
|  | Space | Displays an inline popup window with properties common to all elements. |
|  | Del | Deletes the selected element. |
|  | Ctrl + Up | Moves the element up. Alternatively, use drag & drop to change the position within the current container. |
|  | Ctrl + Down | Moves the element down. Alternatively, use drag & drop to change the position within the current container. |
|  |  | Distributes the columns of the selected row evenly. |
|  |  | Splits the selected column into two. |
|  | Alt + Left | Decreases the column offset by one. |
|  | Alt + Right | Increases the column offset by one. |

### Keyboard Support

In addition to the keyboard shortcuts listed in the table above, there is also keyboard support for doing things like copy, cut, paste, and navigating around the hierarchy of elements on the canvas.

Although the layout editor provides a link to a small pop-up window listing all of the available keyboard shortcuts, I included a complete reference here:

|  |  |
| --- | --- |
| **Clipboard** | |
| Ctrl + X / ⌘ + X | Cuts the selected element. |
| Ctrl + C / ⌘ + C | Copies the selected element. |
| Ctrl + V / ⌘ + V | Pastes the copied element into the selected container element. |
| **Resizing Columns** | |
| Alt + Left | Moves the left edge of the focused column left. |
| Alt + Right | Moves the left edge of the focused column right. |
| Shift + left | Moves the right edge of the focused column left. |
| Shift + Right | Moves the right edge of the focused column right. |
| The Alt and Shift keys can also be combined to move both edges simultaneously. | |
| **Focus** | |
| Up | Moves focus to the previous element (above) |
| Down | Moves focus to the next element (below). |
| Left | Moves focus to the previous column (left). |
| Right | Moves focus to the next column (right). |
| Alt + Up | Moves focus to the parent element. |
| Alt + Down | Moves focus to the first child element |
| **Editing** | |
| Enter | Opens the content editor of the selected element. |
| Space | Opens the properties popup of the selected element. |
| Esc | Closes the properties popup of the selected element. |
| Del | Deletes the selected element. |
| **Moving** | |
| Ctrl + Up / ⌘ + Up | Moves the selected element up. |
| Ctrl + Down / ⌘ + Down | Moves the selected element down. |
| Ctrl + Left / ⌘ + Left | Moves the selected element left. |
| Ctrl + Right / ⌘ + Right | Moves the selected element right. |

### Moving Elements within its Container

Once an element is placed on the canvas, its position can be changed within its container using drag & drop or using the Ctrl + arrow keys.

### Moving Elements across Containers

At the time of this writing, it is not possible to move an element to another container using drag & drop. Instead, you will have to use the Cut/Paste keyboard shortcuts (Ctrl+X and Ctrl+V) to move an element from its current container to another one.

### Re-sizing Columns

Column elements can be re-sized by dragging their left and right edges. When you re-size a column, its adjacent column will be re-sized as well. If you want to re-size a column and introduce an offset (basically "detaching" the column from its neighbor), press the Alt key while dragging the edges. It works pretty slick, try it out.

## Layouts on the Front-end

Enabling the user to create and manage layouts from the back-end is only one half of the story of course. The other half is getting that layout out on the screen on the front-end. To accomplish that, the Layout Part driver simply invokes the driver of each element to build a shape. The resulting shape is a hierarchy of element shapes, ready for display on the front-end.

Each element is responsible for providing its own shape template. Container elements’ shape templates render each of their child elements.

We’ll learn how to take over the default rendering of elements in chapter 10.

## Summary

In this chapter, I provided a high level overview of what the Layouts module is all about. Which, at its core, is about the user being able to add elements to a canvas.

Although that may sounds pretty mundane, it is actually a very powerful feature that unlocks a host of new possibilities to the user which we will explore in the rest of this book.

# Meet the Elements

In this chapter, we'll go over all of the available elements in the default Orchard distribution. Most elements should be self-explanatory to use, but I think that some of them could use a little bit of a background to get a decent understanding on how to use them.

Elements are grouped by their category, so let’s go over them first.

## Element Categories

The list of elements as well as categories are completely extensible of course, but by default, Orchard comes with the following list of categories:

* Layout
* Content
* Media
* Parts
* Fields
* Snippets
* UI

Custom modules can provide additional categories or associate custom elements with existing categories.

What follows next is a complete list of available elements when all features (except the features from Orchard.DynamicForms) are enabled:

* Layout
  + Grid
  + Row
  + Column
  + Canvas
* Content
  + Break
  + Content Item
  + Heading
  + HTML
  + Markdown (requires the Markdown Element feature to be enabled)
  + Paragraph
  + Projection (requires the Projection Element feature to be enabled)
  + Text
* Media
  + Image
  + Media Item
  + Vector Image
* Parts
  + BodyPart
  + CommonPart
  + TagsPart
  + TitlePart
* Fields
  + Any content field attached to the content type will be made available as an element
* Snippets
  + Shape
  + Any Razor file ending in Snippet.cshtml in the current theme or any module will be made available as an element
* UI (New as of Orchard 1.10)
  + Breadcrumbs (requires the UI Elements feature to be enabled)
  + Menu (requires the UI Elements feature to be enabled)
  + Notifications (requires the UI Elements feature to be enabled)
* Widgets (New as of Orchard 1.10)
  + All widgets are available as elements

DynamicForms is another module introduced with Orchard 1.9 and provides its own set of elements.

Let’s go over each category and their elements.

## Layout

Elements in this category are typically container elements that layout their child elements in a particular way. Some container elements only support a specific set of child element types. For example, the Grid element can only contain Row elements, which in turn can only contain Column elements.

### Grid

The Grid element is a container element that can hold only Row elements.

Use the Grid element whenever you want to create a layout of elements. As mentioned, Grids contain Rows, and Rows in turn contain Columns. As you can imagine, these three types of elements are fundamental to create layouts.

When you add a new Grid to the canvas, it will not contain any Row elements initially. You will need to add Row elements to the Grid yourself. Easy enough.

### Row

The Row element, like the Grid element, is a container element. However, this element can only contain Column elements.

The Row element is represented in the toolbox as 7 pre-configured Row elements. The first Row toolbox item will add a single Row element with 1 Column element, the second toolbox item will add a single row with 2 columns, and so forth.

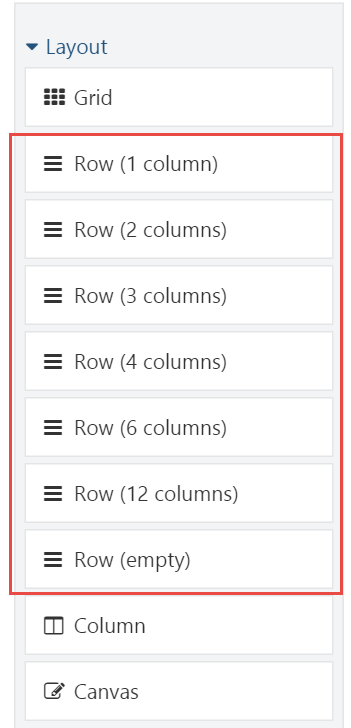


Figure 3‑1 - The Row element is represented in the toolbox as a set of pre-configured rows.

The Row element has a specific toolbar command called Distribute columns evenly. As its name implies, this command will evenly distribute the width of the columns of the row based on a maximum size of 12 columns. For example, if a Row element has 4 columns with varying sizes, and the maximum size is 12, each column will be re-sized to be exactly 3 units in size.

### Column

Unlike the Grid and Row elements, the Column element can hold any type of child elements, except for Row and Column elements. Columns themselves can only be contained by Row elements.

A column has two specialized properties: Width and Offset. Together, these values make up the total size of the column.

#### Adding Columns

There are various ways to add columns to a row. If the row doesn't contain any column, you need to add a Column element to it. In addition, if the Row element contains at least one Column, you can split that column into two. You can repeat this until you have the desired amount of columns.

#### Offsetting

Another property of the Column element is its Offset. By default, the offset for any column is 0. Increasing the offset increases the overall size of the column. You typically use an offset if you want the contents of a column to appear more to the right.

Combined with Row and Grid, the Column element is one of the key ingredients for creating layouts.

### Canvas

The Canvas element, like the Column element, can contain any type of element, except for Row and Column elements. Unlike the Column element, the Canvas element can be added to any other container element, except for Grid and Row, since they have an exclusive white-list of allowed children.

Whenever you create a new layout, that layout always start with a root element of type Canvas.

The Canvas element is great if you ever need a generic type of container element. Although in most cases you probably won't need it since the Column element already is a container, there are occasions where a Canvas is very useful. The prime example being when working with Layout Templates. A layout template enables the reuse of pre-created layouts. When a layout template is applied to a layout, the elements from the template are sealed, which means the user cannot make any modifications to these elements. If a sealed container element is empty however, it will accept elements as its children. But if a sealed container element contains at least one element, no elements are allowed to be added to that container. To work around that, you need to add an empty container element that acts as a placeholder. The Canvas is quite suitable for that purpose.

See chapter 4 for more information on Layout Templates.

## Content

As you have probably guessed, the Content category contains all elements concerning content.

### Break

The Break element is probably one of the simplest elements available. It has no specialized properties. All it doesis render the **<hr>** HTML element.

### Content Item

The Content Item element is very similar to the Content Picker Field and enables the user to select one or more content items to render inline on the canvas.

When adding or editing a Content Item element, the user is presented with a dialog window displaying the content properties of the element. The element has two specialized properties:

* + A list of content items to render
  + The DisplayType to use when rendering the selected content items

This element enables you for example to render the same content item at various locations.

### Heading

The Heading element maps directly to the **<h1>** to **<h6>** HTML elements, and has two specialized properties:

* Level
* Text

The Level indicates the size of the heading and ranges from 1 to 6. For example, if you specify level 3, the **<h3>** tag will be rendered. The following is an example of HTML output when specifying level 6 and the text "Hello Layouts!":

<h6>Hello Layouts!</h6>

### Html

The Html element is probably the most commonly used one when it comes to placing content onto the canvas. It has a single property called HTML, which stores the HTML markup. Use this element whenever you want to display textual content anywhere on the canvas.

The HTML editor used by default is TinyMCE, but you can change this by enabling other features that provide another editor for the html flavor. For example, if you enable the CKEditor feature, that's the editor you'll see when editing Html elements.

### Markdown

The Markdown element lets the user use Markdown syntax which gets transformed into HTML when being rendered on the front-end.

You'll need to enable the Markdown Element feature to enable this element.

### Paragraph

The Paragraph element maps directly to the **<p>** HTML element and enables the user to add individual paragraphs to the page.

You may be wondering why you would want to use this element over the Html element. Well, here's the idea: all elements have common properties such as HTML ID, HTML Class and Html Style. These property values are rendered as HTML attributes on the HTML tag output of the element. Now, when the Html element is rendered and has a value for at least one of the three common properties, a surrounding **<div>** element is rendered onto which the common properties are rendered as attributes. However, there may be occasions where you actually want these common property values to be rendered as attributes on a **<p>** tag directly instead of a surrounding **<div>** element. That's when you use the Paragraph element.

### Projection

The Projection element is the little brother of the Projection Part and allows the user to select a Query to project a list of content items.

In order to make this element available from the toolbox, enable the Projection Element feature.

### Text

The Text element provides a simple textarea input control for its content input and renders that input as raw HTML.

I’m not sure when you would ever want to use this element, but it is there if you prefer to hand-code the HTML directly, instead of using a WYSIWYG editor such as TinyMCE.

## Media

The Media category contains all elements that display some form of media such as images, documents and videos.

### Image

The Image element allows the user to pick a single image content item from the Media Library. When rendered on the front-end, the element renders the **<img>** HTML element. Use the Image element when:

* + You only need to display a single image per element.
  + You want the common properties to be rendered as part of the <img> HTML tag rather than the <div> element that is rendered when using the Media Item element.

### Media Item

The Media Item element allows the user to pick more than one media item. The user can control what display type to use when rendering the selected media items. Use the Media Item element when:

* + You want to display a list of various types of media items.
  + You want to control the display type being used to render each media item.

### Vector Image

The Vector Image element is similar to the Image element, but only supports vector graphics formats such as .svg. In addition to the selected media item, the Vector Image element has two additional properties: Width and Height, both expressed in number of pixels. These values will be rendered as width and height attributes on the <img> HTML tag.

## Parts

The Parts category contains elements for content parts that:

* + Have their Placeable property set to true.
  + Are attached to the current content item's type.

Part elements are interesting. They basically enable the user to place the content parts attached to the content type anywhere within the layout of the content item. You will need to configure Placement.info however to prevent the content parts from being displayed at their default locations. You can read ore on that in chapter 9.

### Placeable Parts

Content parts aren’t placeable by default. The Placeable property is a new part property that controls whether the content part is harvested as an element. By default, the Placeable property is set to true for the following content parts:

* + BodyPart
  + CommonPart
  + TagsPart
  + TitlePart

## Fields

The Fields category is similar to the way the Part elements work, but with a few differences:

* + Only content fields attached to the current content item's type are displayed as elements
  + There is no Placeable property for content fields, which means that all content fields are placeable when attached to a content type.

## Snippets

The Snippets category holds two types of elements by default: One is a generic Shape element, and the others are based on a convention where Razor files are named in a certain way.

### Shape

The Shape element is a very simple element that has just one content property called Shape Type. If you provide the shape type of an existing shape here, then that shape will be rendered wherever you place the Shape element.

You could even enable the Templates feature, create a template, and use that as the shape type for the Shape element.

### Snippet Elements

The second type of elements in the Snippets category are called snippets.

Snippets are quite similar to the Shape element, but the key difference is that instead of you providing the shape type name, the Snippet element harvester provides elements based on the existence of Razor files in the current theme whose file names end in Snippet.cshtml.

Snippets are provided by the Layout Snippets feature, so be sure to enable it when you want to try it out.

For example, a Razor file called LogoSnippet.cshtml in the Views folder of the current theme (or any module for that matter) would yield an element called Logo.

Snippets are a great tool for theme and module developers, as they provide a quick way of providing elements without having to write element classes and drivers.

What's more, Snippets can be made configurable. We will how this works in detail in chapter 11.

## UI

This feature is new as part of Orchard 1.10

The idea behind the UI elements is that they provide the user with elements that make up the UI of their site. Think menus, breadcrumbs and notifications.

The UI elements are provided by the \*UI Elements\* feature.

However, I’m afraid that these elements will not be very useful before Orchard has support for adding elements to zones and layers. Let me try and explain why.

Imagine you wanted to display the main menu using the Menu element. There are two major limitations with this:

1. There is no way to add elements to global zones. Although you could work around this by simply designing your theme in such a way that your entire website consists of a single Content zone and leveraging Layout Templates, this won't solve the next problem, which is:

2. The layout part is associated with content items. This means that their layouts only appear when you request the content item. This means that if you navigated to the Login screen for example, you would no longer see the main menu if you implemented that as a Menu element.

So, unless you want to display a UI element on specific pages, my advice is to not use them until Orchard unifies its Widgets and Elements story.

## Widgets

This category provides all widgets as elements, enabling the user to add widgets to layouts.

When you place a widget element onto a layout, the widget element harvester will create an actual widget content item for you and render that one on the front end. The created widget is not linked with any zone or layer, since it is linked with the content item containing the layout. If you delete the content item or the widget element, the widget itself is deleted as well, since it's managed by the element.

The ability to add widgets to layouts is new since Orchard 1.10.

## Summary

In this chapter, we got to meet all of the elements that ship with Orchard out of the box.

We went over each element in detail to get a better understanding of each element’s intended use.

The set of elements can be extended by custom element harvesters, which are responsible for providing elements to the system. Module developers can create custom harvesters, into which we will look in great detail in Part 3 - Extensibility.