Escenic Widget Framework

User Guide

3.4.1.173597

A CCI COMPANY

Table of Contents

1 Introduction

The Escenic Widget Framework is an add-on product for the Escenic Content Engine that greatly simplifies the process of designing publications. Without the Widget Framework, publication design requires considerable HTML and JSP programming skills. With the Widget Framework, publications can be designed using a drag-and-drop interface in Escenic Content Studio (the Escenic content editor).

This manual is a user guide for:

- Publication designers who want to use the Widget Framework to design Escenic publications
- Editors and/or journalists who want to be able to "tweak" the layout of Widget Framework publications

The prerequisites for using this manual are:

- You have the general design skills needed to work with publication layouts
- You are familiar with the general structure of Escenic publications
- You already know how to use Content Studio for editorial purposes
- You already have some experience of using Web Studio for section management purposes

1.1 What is The Widget Framework?

The Escenic Content Engine is a **template-based** publishing system, in which content production is completely separated from layout design. This allows writers and editors to concentrate on the production of content without needing to think about layout, and allows designers to ensure that a publication has a consistent, well-designed appearance. Web pages are generated by combining content items written and edited using Content Studio with templates written in HTML/JSP.

This approach works well, but it has some disadvantages:

- It requires designers to have HTML and JSP programming skills in addition to design skills
- It makes publication design a relatively slow and error-prone process, with the result that:
  - Publications cannot easily be redesigned for special occasions
  - The production of ad-hoc extra publications is difficult and in general, too costly

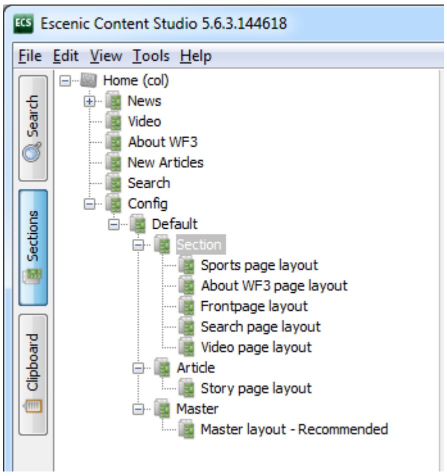
The Widget Framework solves this problem by enabling publication designers to assemble templates from a library of predefined template fragments called **widgets**. In this way it is possible to build a complete set of templates for a publication in a fraction of the time it would take to write, test and debug templates by hand.

1.1.1 About Templates

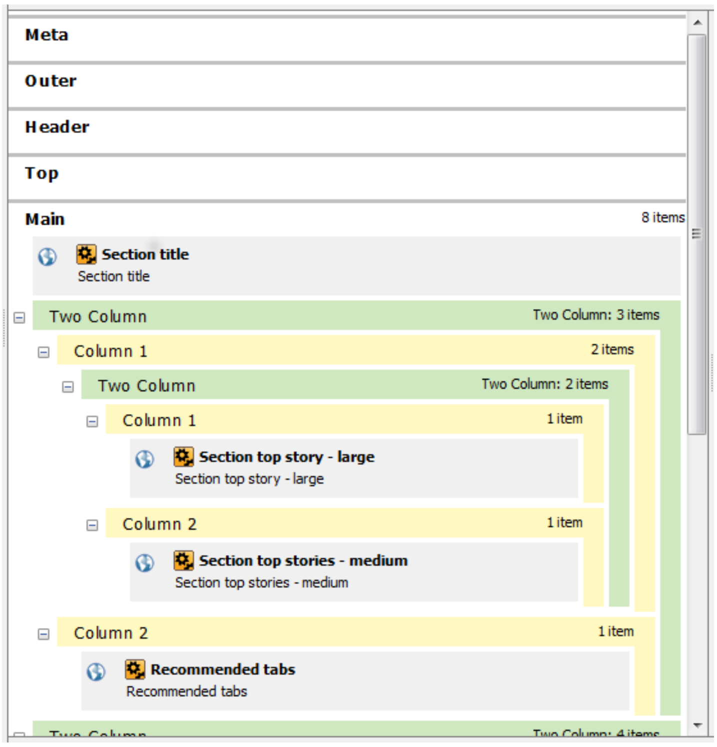
A Widget Framework template is represented in Content Studio by a special kind of section called a **template section**. It appears in the Content Studio section tree along with all the other sections. The factors that differentiate a template section from an ordinary section are:

- Its name: a template always has a name starting with **`config`**.
- Its section page content: instead of actual content items (teasers and so on), a template's section page contains widgets.
- The section page's Page options property, which is set to **`config`**.

The following illustration shows a publication section hierarchy. Note that the template sections all have names starting with **`config`**, and that they are organized in their own tree:



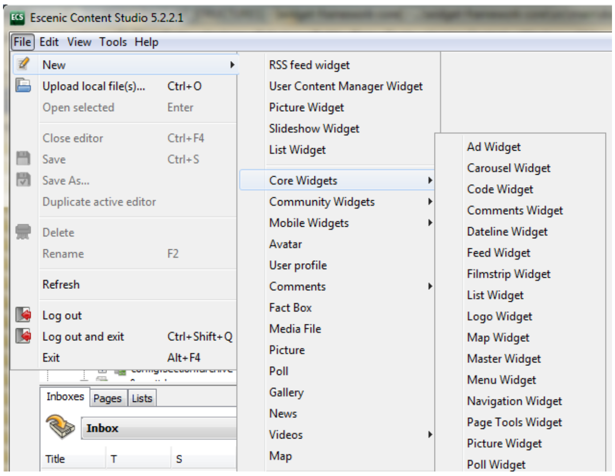
Here is a template's section page as it appears in a Content Studio section page editor:



All the "content items" on this section page are in fact widgets.

1.1.2 About Widgets

A widget is represented in Content Studio by a special kind of content item. If the Widget Framework is installed on your site, and you click on **File** > **New** in Content Studio, then the displayed sub-menu should contain a **Widgets** option (or possibly several widget options such as **Core widgets** and **Community widgets**) . Selecting one of these options displays a further sub-menu containing a list of available widget types:



The Widget Framework includes a set of standard widgets for creating page components such as:

- breadcrumbs
- menus
- search tools
- teasers
- stories

making it possible to meet most requirements "out of the box". If you have special needs that are not satisfied by the widgets in the standard, it is possible to write custom widgets of your own (if you have the necessary HTML and JSP programming skills). This manual does not, however, cover the design and creation of custom widgets.

The standard widgets do, however, really satisfy a very wide range of requirements, since they are all configurable: that is, they have **properties** which you can use to control their appearance and behavior. This ensures that using the Widget Framework does not impose a generic, pre-packaged look and feel on publications - there is still plenty of scope for your creativity.

1.2 How Do I Start?

Now that you have a basic idea of what the Widget Framework is intended to do and how it works, it's probably best to get started. If the Widget Framework has been installed at your site, then all you need to do is start up Content Studio and log in to a publication. The next two chapters will help you to get started editing templates and widgets.

2 Templates

In the widget framework, a template is a special kind of section. A template differs from an ordinary section in the following ways:

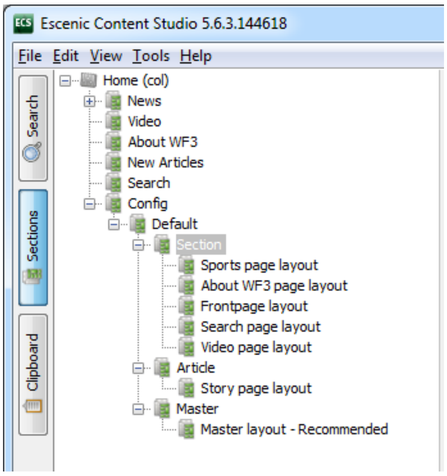
- It is named according to a special convention that identifies it as a template rather than an ordinary section.
- Instead of containing actual content (teasers and so on), it contains widgets. The widgets define how content is to be presented on section pages and on content pages.

This chapter will tell you how to create and edit templates. It also contains explanations of how templates work, and how they are related to ordinary sections and content pages. It is probably a good idea to read through the whole section and then to try following the instructions in section 2.1.

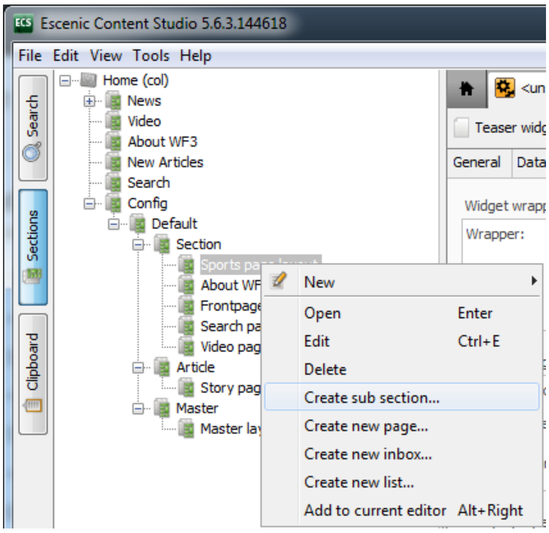
2.1 Creating a Template

A template is basically created in the same way as an ordinary section, but it must be named correctly and should be created in the `config` subtree. To create a template:

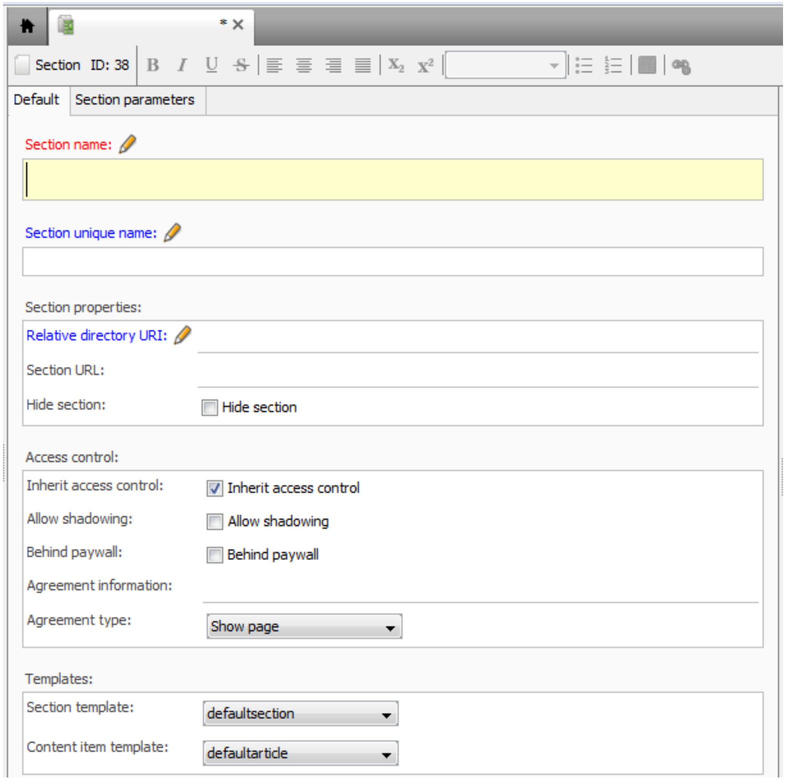
1. Open the Content Studio **Sections** panel and expand the section tree to display the existing templates in the `config` tree:



2. Select the template you want to be the parent of your new template and right click on it to display its context menu:



3. Select **Create sub section...** from the displayed menu. The new section that will be your template is opened in an editor:



Section name:

Section unique name:

Section properties:

Relative directory URI:

Section URL:

Hide section: ☐ Hide section

Access control:

Inherit access control: ☑ Inherit access control

Allow shadowing: ☐ Allow shadowing

Behind paywall: ☐ Behind paywall

Agreement information:

Agreement type: Show page

Templates:

Section template: defaultsection

Content item template: defaultarticle

4. Enter values for the following properties:

Section name
: You can set the section name to anything you like, but you are recommended to set it to the same value as **Unique section name**.

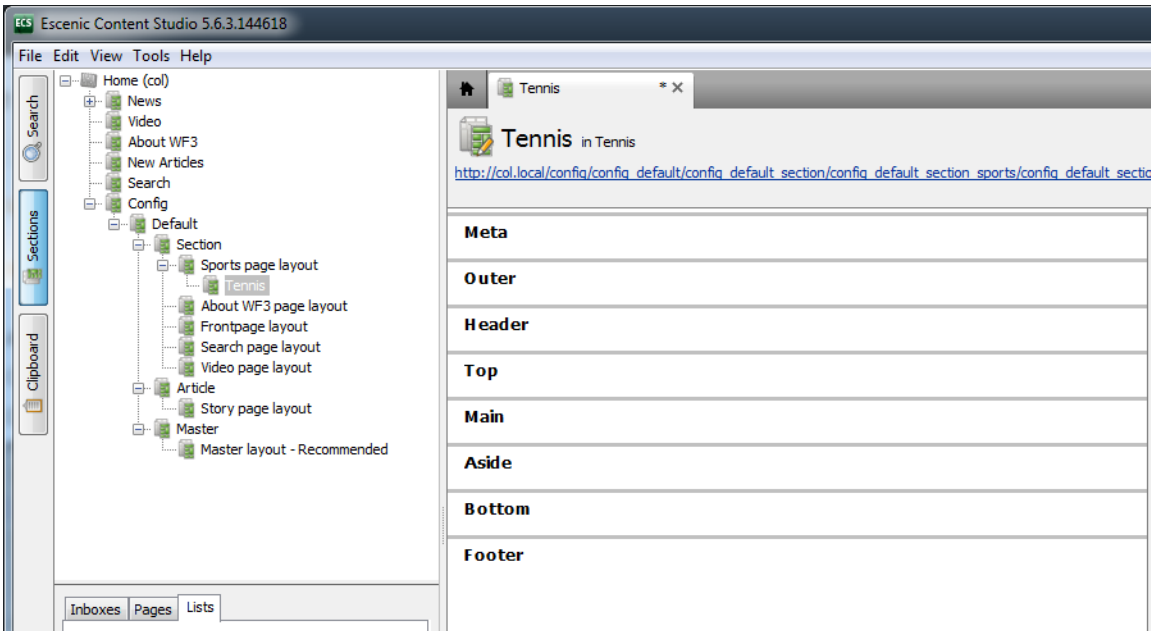
Unique section name
: The section's unique name must be set according a special naming convention. For details, see section 2.2.

Relative directory URI

It doesn't really matter what you enter here, but by convention you repeat the **Unique section name**, replacing all `"."` characters with `"_"`.

All the other properties can be left empty or with their default values.

5. Click on **Save & close**.
6. Double-click on your new template in the section to open its section page for editing:



7. Add the groups and widgets you require to the template.
8. Click on the **Save** button at the bottom of the section editor (or the **Publish** button if it is ready to use).

2.2 Naming Templates

How you name templates is very important, since it determines how they are used. The following names must be used:

`config`

`config` is the container for all templates. It is itself a template, however, and should not contain any layout objects (groups and widgets). If you do put any layout widgets in it, they will not be used.

`config.default`

The `default` template is used for all pages: section pages, article pages and tag pages. It is, as its name suggest, the root of the default template tree. You can create other template trees for special purposes (see section 2.2.1).

`config.default.section`

The `config.default.section` template is is used for all section pages. It is a child of the `config.default` template.

`config.default.article`
The **`config.default.article`** template is used for all content pages. It is a child of the **`config.default`** template.

`config.default.tag`
The **`config.default.tag`** template is used for all tag pages. It is a child of the **`config.default`** template.

`config.default.section.`*`section-name`*
There can be many such templates, where *section-name* is the unique name of a particular section in the publication. The template will then be used for this particular section. It is a descendant (child, grandchild etc.) of the **`config.default.section`** template.

`config.default.article.type.`*`type-name`*
There can be many such templates, where *type-name* is the name of one of the publication's content types. The template will then be used for content items of this particular type. It is a child of the **`config.default.article`** template.

`config.default.article.`*`section-name`*
There can be many such templates, where *section-name* is the unique name of a particular section in the publication. The template will then be used for content items belonging to this particular section. It is a child of the **`config.default.article`** template.

`config.default.article.`*`section-name`*`.type.`*`type-name`*
There can be many such templates, where *section-name* is the unique name of a particular section in the publication and *type-name* is the name of one of the publication's content types. The template will then be used for *type-name* content items belonging to the section called *section-name*. It is a child of the **`config.default.article`** template.

`config.default.tag.scheme.`*`tag-structure-name`*
There can be many such templates, where *tag-structure-name* is the name of a particular tag structure used in the publication - for example, **`config.default.tag.scheme.places`**. The template will then be used for tag pages belonging to the specified tag structure. It is a child of the **`config.default.tag`** template.

`config.default.tag.`*`tag-structure-name`*`.`*`tag-term`*
There can be many such templates, where *tag-structure-name* is the name of a particular tag structure used in the publication, and *tag-term* is the **term** of a tag in that structure - for example, **`config.default.tag.places.bangladesh`**. The template will then be used for that specific tag's tag page. It is a child of the **`config.default.tag`** template.

`config.default.master.`*`master-template-name`*
There can be many such templates, each defining a **master template**. A master template contains a template fragment that can be re-used in other templates. It is a child of the **`config.default`** template. You can give a master template any name you choose. For more detailed information about master templates, see section 2.6.

This means that:

- The layout of a section called **`news`** will be based on the layouts in the templates **`config`**, **`config.default.section`** and **`config.default.section.news`** (if it exists).
- The layout of a content item belonging to the content type **`story`** will be based on the layouts in the templates **`config.default`**, **`config.default.article`** and **`config.default.article.type.story`** (if it exists).

- If there is a template called **`config.default.article.news.type.story`**, then any **`story`** content items belonging to the **`news`** section will use this template instead of **`config.default.article.type.story`**.
- The layout of a tag page for the **`countries`** tag **`bangladesh`** will be based on the layouts in the templates **`config.default`**, **`config.default.tag`**, **`config.default.tag.scheme.countries`** (if it exists) and **`config.default.tag.countries.bangladesh`** (if it exists).

2.2.1 Multiple Template Roots (Content Profiles)

The Widget Framework is delivered with a single template root or **content profile** called **`default`**. It is, however, possible to create additional template roots, each with their own hierarchy of templates. In this way you can define different layouts for different purposes. You might, for example, add a **`newsletter`** template root, with a hierarchy of templates designed to provide a newsletter layout, or an **`inline`** root containing templates for in-line loading (see section 2.2.1.1).

For information about how to configure the Widget Framework to support and make use of multiple template roots, see Content Profiles in the **Escenic Widget Framework Developer Guide**.

2.2.1.1 The Inline Template Root

The template root name **`inline`** is reserved for a special purpose. It can be used to provide a special set of **`article`** templates that are used for in-line loading of content. In-line loading means that when the user clicks on a teaser link on a section page, the content item is loaded directly into the current page instead of being displayed on a new page.

This on-line loading behavior can be configured in the **Link settings** of **`Teaser`** and **`Teaser view`** widgets. When a content item is loaded in this way, its layout is determined by the content of the appropriate **`config.inline.article`** template rather than a **`config.default.article`** template.

Inline templates are only ever selected for content items, so there is no point including **`section`** templates under the **`inline`** template root.

Inline templates are selected in exactly the same way as default templates. If, for example you want a special **`inline`** template to be used for your **`Story`** content items, then you can achieve this by defining the following **`inline`** template hierarchy:

```
config
  config.inline
    config.inline.article
      config.inline.article.type.story
```

2.3 Organizing Templates

Template sections are organized in a hierarchy under the **`config`** section. The **`config`** section itself is not a template. The **`config.default`** template is a child of the **`config`** section and is the root of the default template inheritance tree. Beneath the **`config.default`** template are the standard templates **`config.default.section`**, **`config.default.article`**, **`config.default.tag`** and **`config.default.master`**:

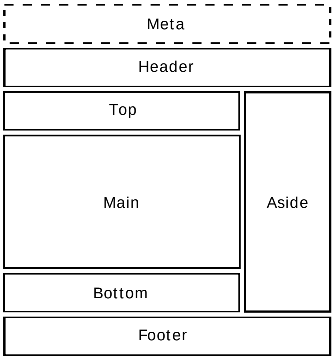
```
config
  config.default
    config.default.section
    config.default.article
    config.default.tag
    config.default.master
```

This part of the template hierarchy is fixed. Below this point, the templates and their hierarchical structure are user-defined. The hierarchical structure is very important, since it governs how the layouts in templates are combined by inheritance. For more about the details of how template inheritance works, see section 2.5.

2.4 Templates, Section Pages and Content Pages

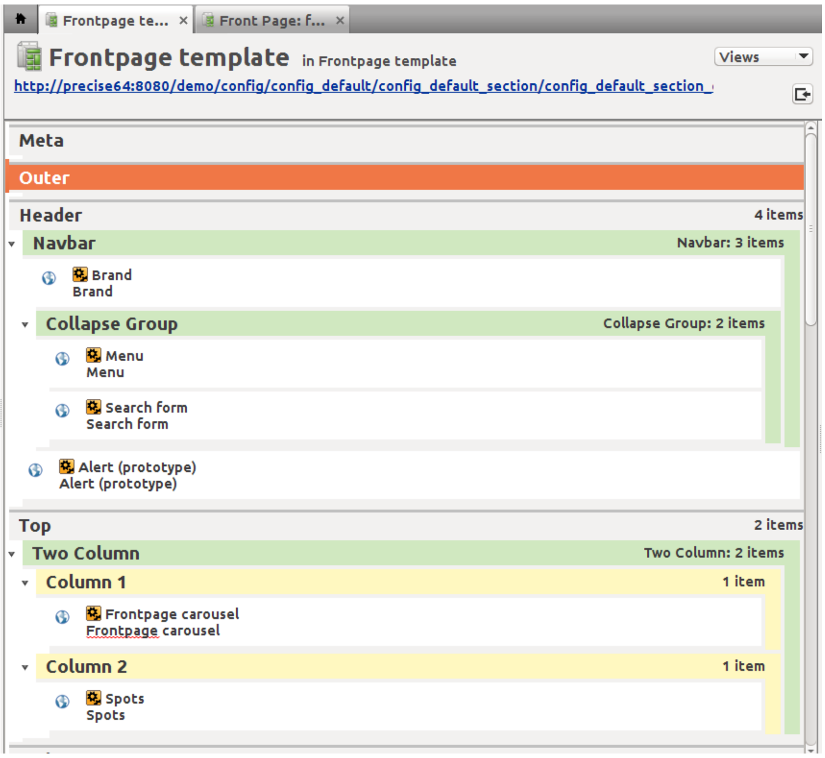
A template defines the graphical layout of a set of publication pages. That layout is stored in the template section's active page. Its top-level structure consists of a set of areas representing the publication's overall graphical structure. The default set of areas used in the demo publication delivered with the Widget Framework consists of the following areas: `Outer`, `Header`, `Top`, `Main`, `Aside`, `Bottom` and `Footer`. It also has a special area called `Meta`, which is used to hold "invisible" widgets."Invisible" widgets affect how pages work in some way, but do not occupy any physical space on the page.

The exact physical position, shape and size of these areas is configurable - they are entirely determined by CSS and may be device-dependent - but they will usually look something like this:



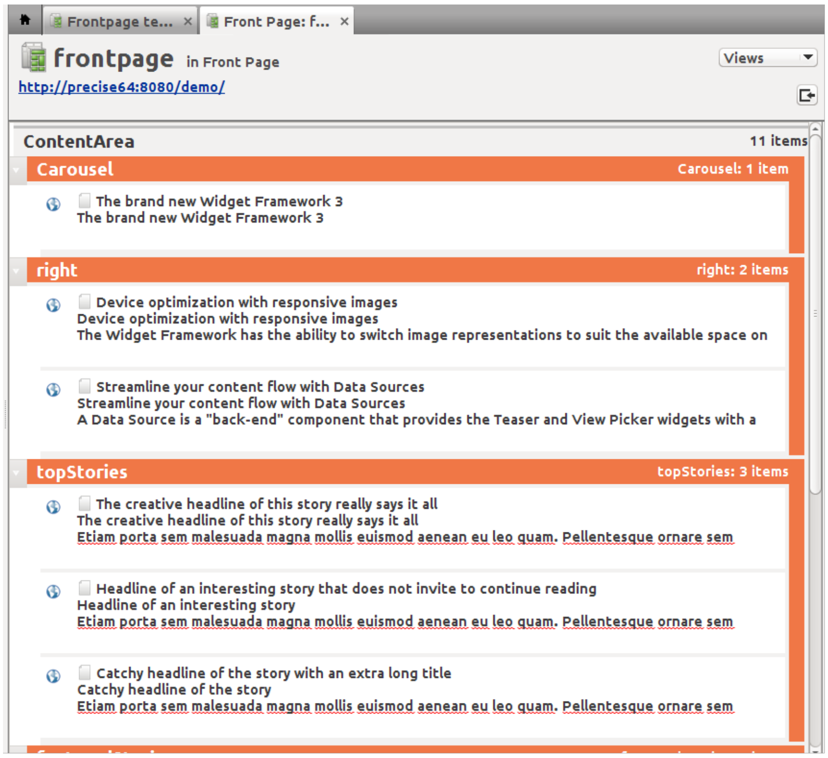
Each of these areas may have an internal structure of groups and areas. The `Main` area of a section, for example, might have a `Top Stories` group and a `Featured Stories` group, each holding content items whose teasers are to be placed in the main area of the page.

The physical shape, position and size of groups and areas are not shown when editing templates in Content Studio:



The names of the groups and areas created within each top-level area, do however, usually describe some of their physical characteristics (`Two column` and `Three column` groups, for example).

The section pages on which journalists and editors desk their stories in Content Studio have a very different structure from the template pages on which they are based. A typical section page has only one top-level area called `ContentArea`, which is subdivided into **logical** groups that have no direct relationship with physical page layout. These groups have names like **`Top stories`**, **`Featured stories`**, **`Main`**, **`Carousel`** and so on:



Carousel — Carousel: 1 item

☐ The brand new Widget Framework 3
The brand new Widget Framework 3

right — right: 2 items

☐ Device optimization with responsive images
Device optimization with responsive images
The Widget Framework has the ability to switch image representations to suit the available space on

☐ Streamline your content flow with Data Sources
Streamline your content flow with Data Sources
A Data Source is a "back-end" component that provides the Teaser and View Picker widgets with a

topStories — topStories: 3 items

☐ The creative headline of this story really says it all
The creative headline of this story really says it all
Etiam porta sem malesuada magna mollis euismod aenean eu leo quam. Pellentesque ornare sem

☐ Headline of an interesting story that does not invite to continue reading
Headline of an interesting story
Etiam porta sem malesuada magna mollis euismod aenean eu leo quam. Pellentesque ornare sem

☐ Catchy headline of the story with an extra long title
Catchy headline of the story
Etiam porta sem malesuada magna mollis euismod aenean eu leo quam. Pellentesque ornare sem

There is, in other words, no direct correspondence between the groups on a section page and the groups in the templates that govern its layout. In order for a content item that has been dragged into a group on a section page to actually appear on the published page, it must be explicitly **selected** by one of the widgets placed on the template.

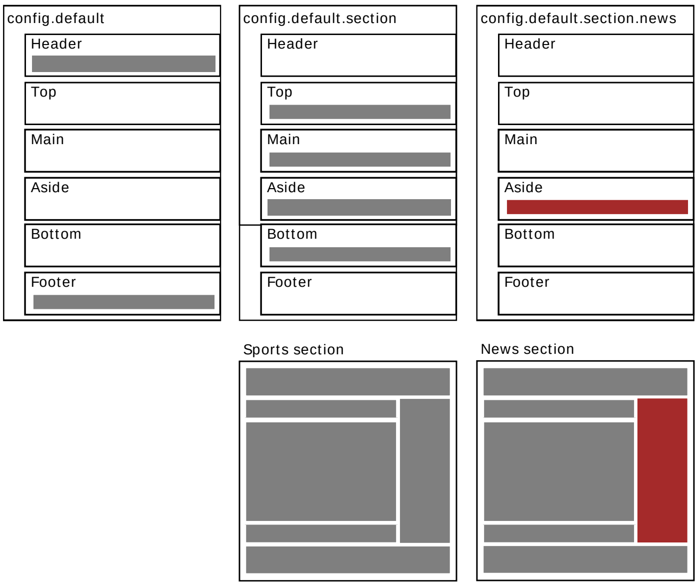
For more information about this, and about widgets in general, see chapter 3.

2.5 Inheritance in the Widget Framework

As described above, all of a publication's templates have the same top-level structure: **`Meta`**, **`Outer`**, **`Header`**, **`Top`**, **`Main`**, **`Aside`**, **`Bottom`** and **`Footer`**, for example. If a template (**`config.default.section.news`**, for example) has widgets in all of these sections, then the news section of the publication will take all of its layout from this template. If, however, only the **`Aside`** area of the template contains widgets, then the Widget Framework will look in the **`config.default.section`** template to get the layout for the **`Meta`**, **`Outer`**, **`Header`**, **`Top`**, **`Main`**, **`Bottom`** and **`Footer`** areas. If **`config.default.section`** is not complete either (if, for example, it's **`Outer`**, **`Header`** and **`Footer`** areas are empty) then the Widget Framework will look in the **`config.default`** template for the missing information.

This inheritance mechanism makes it very easy to create a standardized layout for a whole publication, and only create specialized layouts where you actually need them. A typical approach is to define a standard layout for the **`Header`** and **`Footer`** areas in the **`config.default`** template, and then define standard layouts for the remaining areas in the **`config.default.section`** and **`config.default.article`** templates. If your publication has a very standardized layout, then this may be sufficient. If, however, some of your section pages have significantly different layouts, then you can achieve that by creating **`config.default.section.`***name* templates for those particular sections, and overriding just those areas that are different from the standard layout. Similarly you can, if necessary, create **`config.default.article.`***type* templates for any content types that require specialized layouts.

The Sports section in the following illustration takes its **`Header`** and **`Footer`** layouts from the **`config.default`** template, and its **`Top`**, **`Main`**, **`Aside`** and **`Bottom`** layouts from the **`config.default.section`** layout. The News section, however takes its **`Aside`** layout from the **`config.default.section.news`** template, which overrides the **`Aside`** layout in **`config.default.section`**:



The following sections describe the specific inheritance mechanisms used for different types of page.

2.5.1 Section Template Inheritance

There are two different section template inheritance mechanisms, one governing the selection of templates, and one governing the merging of the layout definitions in the sections. This section describes how they are used to display a section called **`premierleague`** in a section hierarchy like this:

```
ece_frontpage
  sports
    football
      premierleague
```

with a corresponding template hierarchy that looks like this:

```
config
  config.default.section
    config.default.section.sports
      config.default.section.football
        config.default.section.premierleague
```

Template selection

The Widget Framework first has to determine which template to select. In this case it is simple, since the **`premierleague`** section has its own template (**`config.default.section.premierleague`**). If there was no such template, however, then the Widget Framework would look for the template of its parent section (**`football`**), and so on upwards. The Widget Framework bases its search on the section hierarchy, looking for templates in the following order:

1. **`config.default.section.premierleague`**
2. **`config.default.section.football`**
3. **`config.default.section.sports`**
4. **`config.default.section`**
5. **`config.default`**

> Note that even though **`ece_frontpage`** is the parent of **`premierleague`**, its template is not included in the lookup sequence. **`ece_frontpage`** is a special case, and its template is never used for any other section.

Area layout selection

Once it has found a template (in this case **`config.default.section.premierleague`**) the Widget Framework then tries to assemble a complete template, containing a layout for each top-level area. In this case it bases its search on the template hierarchy rather than the section hierarchy. If, for example, **`config.default.section.premierleague`** only contains a layout for the **`Aside`** area, then it will continue to search for the **`Meta`**, **`Outer`**, **`Header`**, **`Top`**, **`Main`**, **`Bottom`** and **`Footer`** areas in the **`config.default.section.football`** template. It will search the templates in the following order, stopping once it has a complete set of layouts.

1. **`config.default.section.premierleague`**
2. **`config.default.section.football`**
3. **`config.default.section.sports`**
4. **`config.default.section`**
5. **`config`**

With a flatter template hierarchy

The two search sequences described above are very similar because the template hierarchy closely matches the section hierarchy. This need not be the case, however. If the template hierarchy looked more like this:

```
config
  config.default.section
    config.default.section.sports
    config.default.section.football
    config.default.section.premierleague
```

then the template search sequence would be the same, but the layout search sequence would be:

1. **`config.default.section.premierleague`**
2. **`config.default.section`**

3. `config`

With a missing template

If there was no `premierleague` template at all:

```
config
  config.default.section
    config.default.section.sports
    config.default.section.football
```

then the Widget Framework would start by using the template search sequence to select the `football` section's template, `config.default.section.football`, and would then search for layouts in the following sequence:

1. `config.default.section.football`
2. `config.default.section`
3. `config`

2.5.2 Content Item (Article) Template Inheritance

Content item templates (or article templates) are all located under the `config.default.article` template. You can create:

- Specialized templates for specific content item types (for example `config.default.article.type.story` and `config.default.article.type.picture`)
- Specialized templates for content items belonging to specific sections (for example `config.default.article.premierleague` or `config.default.article.football`)
- Specialized templates for specific content item types belonging to specific sections (for example `config.default.article.premierleague.type.story` or `config.default.article.premierleague.type.picture`)

Below is a description of how the template and area layouts are selected for a `story` content item in the `premierleague` section, assuming a section hierarchy like this:

```
ece_frontpage
  sports
    football
      premierleague
```

Template selection

To find the correct template, the Widget Framework first looks for an article template of the correct type for the correct section. If there is no such template, then it looks for an article template for the correct section. If it still doesn't find a template then it repeats these two steps for the parent section, (`football`), and so on up the section hierarchy:

1. `config.default.article.premierleague.type.story`
2. `config.default.article.premierleague`
3. `config.default.article.football.type.story`
4. `config.default.article.football`

5. `config.default.article.sports.type.story`
6. `config.default.article.sports`
7. `config.default.article.ece_frontpage.type.story`
8. `config.default.article.ece_frontpage`
9. `config.default.article.type.story`
10. `config.default.article`
11. `config.default`

Area layout selection

> This section describes the Widget Framework's **default** area layout selection mechanism, which is based on the template hierarchy. In earlier versions of the Widget Framework, however, area layout selection was based on the template naming convention. You can force the current version of the Widget Framework to use this old area layout selection mechanism by setting the property `wf.conventionalInheritance` to `true` in your publication `feature` resource.

The Widget Framework assembles content item layouts in the same way as section page layouts. If, for example, `config.default.article.premierleague.type.story` only contains a layout for the **Aside** area, then it will continue to search for the **Meta, Outer, Header, Top, Main, Bottom** and **Footer** areas in the `config.default.article.premierleague` template.

If template hierarchy is like this:

```
config.default
  config.default.article
    config.default.article.type.story
    config.default.article.type.picture
    config.default.article.premierleague
      config.default.article.premierleague.type.story
```

It will search the templates in the following order:

1. `config.default.article.premierleague.type.story`
2. `config.default.article.premierleague`
3. `config.default.article`
4. `config.default`

Section template hierarchy variations

Section template hierarchy variations such as a flatter structure or missing templates are handled in exactly the same way as for section templates (see section 2.5.1).

2.5.3 Tag Template Inheritance

Tag pages are automatically generated pages created to display links and content related to tags. When a user clicks on a tag keyword link, the corresponding tag page is displayed. The tag page usually contains a brief article about or definition of the subject of the tag, followed by teasers and links leading to tagged content items. The layout of these tag pages can be defined using tag page templates.

Tag page templates are all located under the `config.default.tag` template. You can create:

- Specialized templates for specific tag structures (for example `config.default.tag.scheme.places` and `config.default.tag.scheme.sports`)
- Specialized templates for specific tags (for example `config.default.tag.places.bangladesh` or `config.default.tag.sports.football`)

Below is a description of how the template and area layouts are selected for a `dhaka` tag page, assuming a tag structure like this:

```
places (Tag structure)
  asia
    bangladesh
      dhaka
```

Template selection

To find the correct template, the Widget Framework first looks for a tag template for the specific tag (`dhaka`). If there is no such template, then it looks to see if the parent tag (`bangladesh`) has a template, and so on up the tag hierarchy. It will then check to see if there is a template for the tag structure (`places`), and if not it will continue up the template hierarchy to the root:

1. `config.default.tag.places.dhaka`
2. `config.default.tag.places.bangladesh`
3. `config.default.tag.places.asia`
4. `config.default.tag.scheme.places`
5. `config.default.tag`
6. `config.default`

Area layout selection

> Just like in case of section and article, area layout selection for tag templates follows the **default** area layout selection mechanism, which is based on the template hierarchy.

The Widget Framework assembles tag page layouts in the same way as section page and content item layouts. If, for example, `config.default.tag.places.dhaka` only contains a layout for the **Aside** area, then it will continue to search for the **Meta, Outer, Header, Top, Main, Bottom** and **Footer** areas in the `config.default.tag.places.bangladesh` template.

For example, if template hierarchy is like this:

```
config.default
  config.default.tag
    config.default.tag.scheme.people
    config.default.tag.scheme.foods
    config.default.tag.scheme.places
      config.default.tag.places.asia
        config.default.tag.places.bangladesh
          config.default.tag.places.dhaka
```

then for the `dhaka` tag, it will search the templates for area layouts in the following order:

1. `config.default.tag.places.dhaka`

2. `config.default.tag.places.bangladesh`
3. `config.default.tag.places.asia`
4. `config.default.tag.scheme.places`
5. `config.default.tag`
6. `config.default`

2.5.4 Custom Templates

Editors and journalists can override the template selection process for either a section page or a Story content item by specifying a **custom template**. To do this the Content Studio user simply enter the name of the required custom template in the section page's or Story's **Custom template** option.

In the case of a section page the **Custom template** option can be displayed by clicking on the section page's **Page Options** button. For a Story content item the **Custom template** option is displayed on the **Layout** tab. In both cases, the effect of specifying a custom template is that the custom template is added to the start of the normal template search sequence. So if, for example, the name of a template that does not exist is specified in the **Custom template** field, then the normal template will be used instead.

Story **Layout** tabs actually contain two custom template fields. As well as the **Custom template** field there is a **Template variant** field. This field offers a less flexible customization mechanism. It can be set to one of 3 predefined values (**Style 1**, **Style 2** and **Style 3**) that correspond to three content-type specific templates called `style1`, `style2` and `style3`. In other words, if the user selected **Style 1**, then the first template looked for in the template search sequence described in section 2.5.2 would be `config.default.article.premierleague.type.story.style1`.

Should the user specify a value in both the **Template variant** and **Custom template** fields, then the value specified in **Custom template** takes precedence.

2.6 Master Templates

The inheritance mechanism described in section 2.5 provides a useful means of re-using groups of widgets. It only works, however, for **whole** areas. You can, for example, re-use the whole of a template's `Main` area by creating a child template with an empty `Main` area. But what if you only want to re-use one particular group of widgets, not the whole area? What if you want to re-use a group of widgets from a template other than the parent of the template you are working on? Master templates provide a more flexible means of re-using groups of widgets that solves these problems.

A master template has a name of the form:

`config.default.master.`*master-template-name*

and contains a set of templates that you want to be able to re-use. You might, for example, create a menu that you want to re-use in many of your templates.

Once you have created a master template containing the widgets you want (let's call it `config.default.master.menu`), you can use it by creating a `master` widget and placing it in any section and/or article templates you like. A `master` widget has two required properties:

Title

The name of the widget instance you have created. You might set this to **`Menu`**.

Master section unique name

The name of the master template this widget is to reference. You would set this to **`config.default.master.menu`** in this case.

Set these properties, save the widget, and then drag it into the templates where you want to use the widget group defined in the master template.

For a description of how you can enable editorial staff to quickly switch between master templates, see section 6.5.

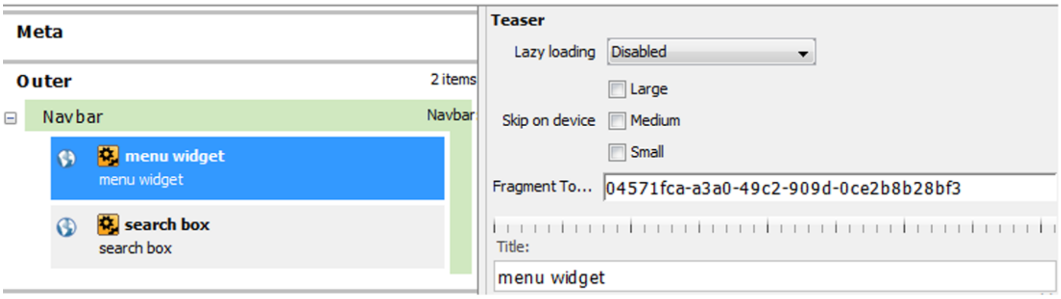
2.7 Lazy Loading

The Widget Framework uses Bootstrap-based CSS to enable **responsive** web sites that adjust page layout to fit the available screen space. A page displayed on a mobile phone (or in a small window on a PC) is formatted differently to the same page displayed full-screen on a PC.

This mechanism provides a good basis for making web sites that perform well on a wide range of devices, but it is not sufficient in all cases. With the Widget Framework's lazy loading feature you can also prevent parts of a web page from loading on certain devices.

You can individually configure the loading behavior of every part of a template (areas, groups and the widgets placed in the template). By default all components are loaded in the normal way but you can specify that certain components should only be loaded if the browser window is a certain size.

The loading behavior is controlled by options displayed in the Content Studio template editor:



Meta

Outer — 2 items

Navbar — Navbar

menu widget — menu widget

search box — search box

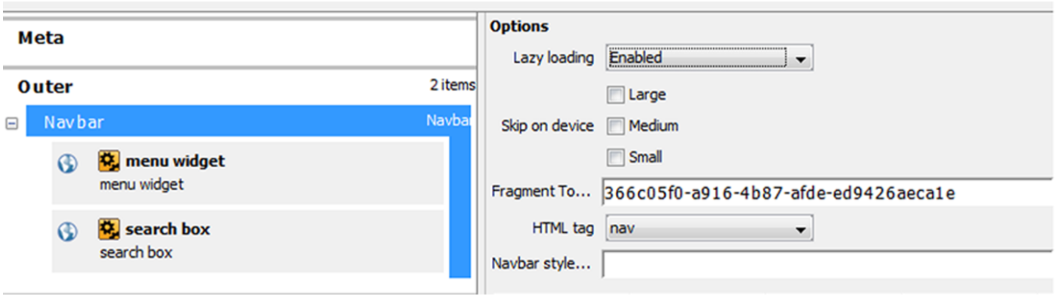
Teaser

Lazy loading: Disabled

Skip on device: ☐ Large ☐ Medium ☐ Small

Fragment To...: 04571fca-a3a0-49c2-909d-0ce2b8b28bf3

Title: menu widget



Meta

Outer — 2 items

Navbar — Navbar

menu widget — menu widget

search box — search box

Options

Lazy loading: Enabled

Skip on device: ☐ Large ☐ Medium ☐ Small

Fragment To...: 366c05f0-a916-4b87-afde-ed9426aeca1e

HTML tag: nav

Navbar style...:

When you select an area, group or widget, the options displayed on the right include the following lazy loading options:

Lazy loading

Select the required value:

Enabled

This page component (area, group or widget) is not loaded if the browser window matches a selected "skip" size.

Disabled (default)

This page component (area, group or widget) is always loaded.

Skip on device

Check the window sizes for which loading is to be skipped.

Large

If checked, and **Lazy loading** is **Enabled**, then this component will not be loaded into large browser windows.

Medium

If checked, and **Lazy loading** is **Enabled**, then this component will not be loaded into medium browser windows.

Small

If checked, and **Lazy loading** is **Enabled**, then this component will not be loaded into small browser windows.

You can determine the meaning of Large, Medium and Small yourself by setting section parameters in the root section of your publication. The parameters (and their default settings) are:

- `wf.clienttype.large=(min-width: 992px)`
- `wf.clienttype.small=(max-width: 767px)`
- `wf.clienttype.medium=(min-width: 768px) and (max-width: 991px)`

Fragment token

An automatically generated ID is written to this field the first time a template is saved. You should not modify this value.

Widget Framework's lazy loading mechanism is SEO-friendly. The fragments that are lazily loaded are indexed by search engines. In order to make them indexable by search engines, Google's Ajax Crawling Scheme is used.

> Nested lazy loading is not supported. If you enable lazy loading for a group or area, then the loading rule you specify is applied to that component and everything it contains. You cannot switch off lazy loading for one of its child components, or apply a different loading rule. Any lazy loading changes you make to child components will be ignored.

3 Widgets

A widget is a **template fragment**: a bundle of HTML, JSP and CSS code that performs a particular function such as displaying a content item or dateline on a publication page. As a publication designer, however, you do not need to think about this internal structure: you will see widgets as special content items that you can work with in Content Studio in much the same way as ordinary content items. You can create, delete and modify them in much the same way as ordinary content items. The main difference is that widgets are placed in templates (the special section pages described in chapter 2) rather than in ordinary section pages.

3.1 About Widgets

Widgets fall into the following main categories:

- Content-based widgets
- Content-free widgets

3.1.1 Content-based Widgets

A content-based widget is a widget that renders all or part of one or more content items. There are two types of content-based widget:

- Widgets that render the current content item and are therefore used only in content templates and tag templates (in which case the tag's topic content item is the current content item). The Content Body widget is an example of this kind of widget.
- Widgets that can render information about many widgets and are therefore primarily used in section templates. Such widgets may also be used in content templates (to render information about a content item's related items, for example). The only widgets of this kind at present are the Teaser widget and the related Teaser View widget.

3.1.2 Content-free Widgets

A content-free widget does not require access to any particular content items, and can be used in both content templates and section templates. The Menu widget, Code widget and Breadcrumb widget are all examples of this kind of widget.

3.1.3 Data Sources

The Teaser widget performs two functions:

- It retrieves a set of content items from the Content Engine.
- It renders selected fields and information about each of the content items in the set.

The widget's data retrieval function is performed by a component called a Data Source. A Data Source is actually an independent, separate component and can be used in three ways:

- Embedded in a widget

- Defined independently and then used by one or more "client" widgets
- Desked in a group on the section page, where it behaves like an automated list

The Data Source component is very powerful and flexible. It allows you to define multiple queries that can select content items using a variety of different methods. You can, for example:

- Select the content items desked in a specified group on the current section page
- Select the current content item's related content items
- Select content items that are tagged with the same tags as the current content item
- Select content items based on data returned by the Escenic Analysis Engine (the most read content items, for example)
- Select content items using a free text search

The results returned by the queries you define are merged into a single collection that you can filter and sort in various ways to produce the final list of content items that will be displayed by a widget or widgets (or directly on a section page in the case of a desked data source).

3.1.3.1 Desked Data Sources

Although data sources are primarily intended to be used as the "back end" of widgets, they can also be used independently by editorial staff as a means of creating automated lists. If a data source is desked directly on an ordinary section page, then it behaves something like a list: on the published page, it is replaced by the content items that it retrieves.

Data source nesting

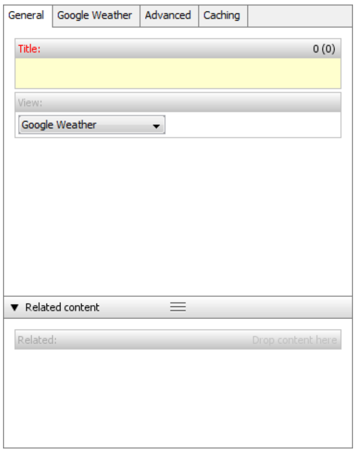
A data source with a "by section group" query selects content items from a section page group, and that section page group may itself contain a data source. This kind of data source nesting is allowed, but limited in order to prevent complex chains of nested data sources and/or loops. The limit can be set by configuring the Widget Framework's `/com/escenic/framework/datasource/fetcher/DeskedItemFetcher` component. This component has a `maxDatasourceNestingLevel` property that limits nesting as follows:

`maxDatasourceNestingLevel`	**Meaning**
0	Desked data sources are ignored
1	Desked data sources are used, but any further nesting is ignored
2	Desked data sources are used. If a desked data source selects from a section page group containing another data source, then that data will be used, but any further nesting is ignored
n...	And so on ...

By default, `maxDatasourceNestingLevel` is set to `1`.

3.2 Widgets and Widget Types

So far in this manual, we have often used the word **widget** to mean **widget type** (it's difficult to avoid doing so), but it is important to clearly understand the difference between the concepts **widget** and **widget type**. If you click on **File** > **New** > **(Core/Community) Widgets** in Content Studio, then what you see in the displayed menu is a list of **widget types**. If you click on one of the listed types - let's say you click on **Menu Widget** - then this results in the creation of an actual **widget**, an instance of the widget type Menu. One of the things you are required to do when you create any kind of widget, is to name the instance you have created by setting it's `Name` property. The following illustration shows the property list for a newly-created menu widget: the **Title** label is displayed in red, indicating that this property is required.



View: Google Weather

Naming the widgets you create is important, because you can create many widgets of the same type. You might, for example, call your Menu widget `Header Menu`, and place it in the Header area. But if you also want your publication to include a footer menu, then you can create another Menu widget called `Footer Menu` and set it up accordingly. A typical content template will usually contain several instances of the Content Field widget type, each set up to display a different content item field: a `Title` widget displaying the `title` field and a `Subtitle` widget displaying the `subtitle` field, for example.

3.3 Creating a Widget

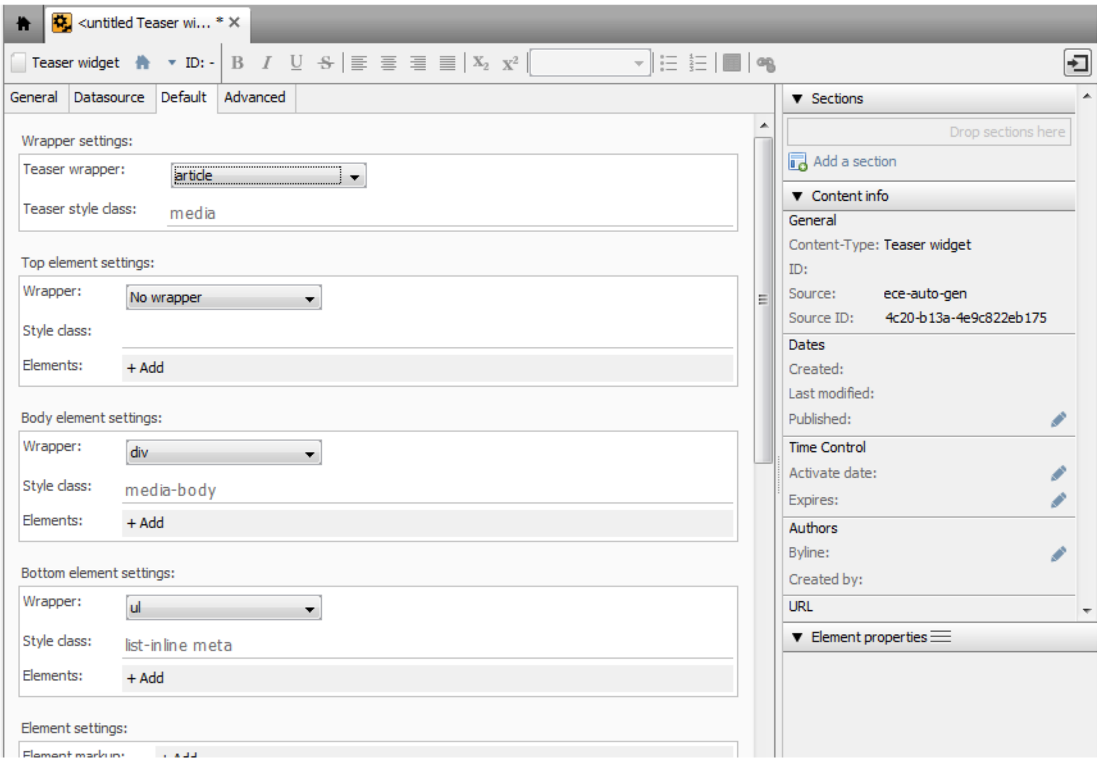
A widget is special kind of content item, so you can create one in the same way as any other content item:

1. Select **File** > **New** > **Widgets**.
2. the type of widget you want to create - **Teaser Widget** for example.
3. Configure the widget by entering values in its property fields.
4. Click on **Save** or **Publish**.

All widgets have at least one mandatory property field (displayed in red) that you must fill in: **Title**, the name that will be used to identify the widget. Some widget types may have other mandatory properties.

Widgets may have many other property fields, but usually most of them will have default values. The fields are divided into groups and displayed on separate tabs. Mandatory property fields always appear on the default **General** tab. For more information about property fields and how they are organized, see section 3.4.

In addition to the **General** tab, most widgets have an **Default** tab and an **Advanced** tab. Some widgets have no additional tabs, others have many.



Wrapper settings:

Teaser wrapper: article

Teaser style class: media

Top element settings:

Wrapper: No wrapper

Style class:

Elements: + Add

Body element settings:

Wrapper: div

Style class: media-body

Elements: + Add

Bottom element settings:

Wrapper: ul

Style class: list-inline meta

Elements: + Add

Element settings:

Element markup: + Add

Content-Type: Teaser widget

ID:

Source: ece-auto-gen

Source ID: 4c20-b13a-4e9c822eb175

3.4 Configuring a Widget

Configuring a widget is a simple matter of opening it in Content Studio and setting its properties. There are very few properties that **must** be set in order for a widget to work, and most properties have sensible defaults, so configuring widgets is in general not particularly difficult. Moreover, many properties are common to most or all widget types, so once you have learned what these common properties control, learning to use new widget types does not require much additional effort.

Widget properties are divided into groups and displayed on separate tabs. The following three tabs are common to almost all widget types:

- **General**
- **Default**
- **Advanced**

The properties most commonly displayed on these tabs are described in the following sections. Some widget types have only these three tabs. Other widget types have additional tabs containing widget-specific properties that are for the most part not discussed in this manual. For information about widget-specific properties, see the widget descriptions in the Escenic Widget Framework Core Widgets Reference.

3.4.1 General Properties

A **General** properties tab contains some or all of the following property fields:

Title
This property is always mandatory. Use it to set the name of the widget.

View
Many widgets have a **View** property. It is intended to allow switching between different widget views. In the current version, however, it has only one value: **Default**.

3.4.2 Default Properties

Most widgets have a **Default** tab, and it usually contains the widget's most commonly-used settings.

3.4.3 Advanced Properties

An **Advanced** properties tab usually contains the following property fields:

Widget wrapper
The HTML elements that make up a widget are always wrapped in a single HTML element. This property lets you choose what element to use for this purpose.

Style Id
This property lets you set the widget wrapper element's `id` attribute, so it can easily be selected in CSS files. This attribute is mainly of interest to theme designers.

Style Class
This property lets you set the widget wrapper element's `class` attribute, so it can easily be selected in CSS files. This attribute is mainly of interest to theme designers.

Enable Edge Side Includes (ESI)
Check this property to enable **Edge Side Includes (ESI)** processing of this widget. ESI is a widely-used standard for caching of web pages (see http://www.edge-delivery.org/). If you check this option, then the HTML code representing the widget is wrapped in ESI mark-up specifying that it may be cached by ESI servers. You can specify that some widgets are cached, while others are not, and you can specify how long different widgets should be cached for. You may, for example, decide that a widget whose content changes infrequently can be cached for a long time. Other widgets with more volatile content can be set up to expire more frequently, or not cached at all.

> Do not enable this option unless:
>
> - You know what ESI is and understand how to use it
> - You know that the generated output will be processed by an ESI server

Maximum age (in seconds)
If you have checked **Enable Edge Side Includes (ESI)**, then you must use this property to specify how long the widget may be cached by the ESI server. The ESI server will only use the cached version of the widget until this limit is reached.

Set cache control directive

If you have checked **Enable Edge Side Includes (ESI)**, then this property controls the value of a cache control directive included in the header of the response generated when a caching server requests a new version of this widget. It can be set to one of the following values:

`public` (the default)

The widget has public content that is valid for all users. The cached copy can therefore be used for all page requests.

`private`

The widget has user-specific content that is not valid for all users. It should therefore be saved in a private user-specific cache and only used for responding to requests from that user.

`no-cache`

The widget should not be cached at all.

The **Advanced** properties tab of some widget types may also contain other widget-specific properties.

3.4.4 Configuring Teaser Widget Data Sources

A Teaser widget renders selected fields from a set of selected content items. The content items are selected by a component called a Data Source. A Teaser widget therefore has a **Data Source** tab that you can use to configure the Data Source. There are two different types of Data Source:

External

An external Data Source is independently-defined: you create it in the same way as a widget. You can then use it in a Teaser widget by dropping it on the widget's **External Data Source** field as a related content item.

Embedded

An embedded Data Source is embedded inside the Teaser widget. You configure it by editing the **Embedded Data Source definition** form displayed on the **Data Source** tab.

A widget can only have one data source, so before you configure a Teaser widget, you first need to decide what kind of Data Source to use. You should use an external Data Source if you want several widgets to be able to share the same Data Source. Otherwise it is most convenient to have the Data Source definition embedded in the widget that uses it. To select the Data Source type you must set the **Use** field (which is the first field on the **Data Source** tab) to **External Data Source** or **Embedded Data Source**.

If you select **Embedded Data Source** then you can configure the embedded Data Source using the form displayed in the **Embedded Data Source definition** field. If you select **External Data Source** then you need to create a Data Source, save it and then associate it with your Teaser widget by dropping it on the widget's **External Data Source** field as a related content item.

To create an external Data Source, select **File** > **New** > **Data Source** and then configure it in the same way as you would configure a widget. The Data Source component has a **Definition** field that contains exactly the same Data Source configuration form as the Teaser Widget's **Embedded Data Source definition** field. For a complete description of this form and how to use it, see Configuring a Data Source.

3.4.5 Configuring View Pickers and Teaser Views

Using an external Data Source component allows you to decouple the Teaser widget's back-end data retrieval function from its front-end presentation functionality. The Teaser View widget performs a similar function with regard to the Teaser widget's front end. A Teaser View widget has all the same presentation functionality as a Teaser widget, but no back-end functionality at all. It does not have an embedded Data Source, nor any means of attaching an external Data Source. A Teaser View widget can only be used in combination with a View Picker widget.

A View Picker widget can be regarded as a kind of switch that connects one Data Source to several Teaser View widgets. Like a Teaser widget, a View Picker widget can be configured to use either an embedded Data Source or an external Data Source. Unlike a Teaser widget, however, the View Picker widget has no front-end presentation functionality. Instead, it has a **Related view** field on which you can drop Teaser View widgets. You can then configure the Teaser View widgets to select content items from the View Picker's Data Source. The Teaser View widgets can select content items based on two criteria:

- Content type
- Position in the Data Source collection

The procedure for configuring these widgets is as follows:

1. Create and configure the Teaser View widgets you require. A Teaser View widget is identical to a Teaser widget, except that it has no **Data Source** tab.
2. **Save**/**Publish** the Teaser Views. Note, however, that these widgets cannot be placed on a template in the normal way.
3. Create and configure a View Picker widget. You can configure it with either an embedded Data Source or an external Data Source in the same way as a Teaser widget.
4. Drop the Teaser View widgets you created on the View Picker's **Related view** field.
5. Select each dropped Teaser View widget in turn and edit its content item selection properties. These properties are displayed in the **Element properties** panel on the right. Note that you can only edit these properties here - you cannot edit them when editing the Teaser View widgets themselves.
6. **Save**/**Publish** the View Picker widget.
7. Place the Teaser View widget in the required location in a template.

3.5 Adding a Widget to a Template

Since widgets are represented by content items in Content Studio, you can add a widget to a template in exactly the same way as you would add a content item to an ordinary section:

1. Click on **Sections** on the left side of the window to display the Sections tab.
2. Locate the template you want to add a widget to in the section tree (`config.default.section`, for example: this is the default template for all section pages). Double-click on the section to open it.
3. Click on **Search** on the left side of the window to display the Search tab.
4. Clear the **Date** field and click on the **Widgets** link. All available widgets will then be listed in the search results area. If you know the name of the widget you want, you can also enter a search

string in the **Find** field before clicking on the **Widgets** link in order to list only the widget you are interested in.

5. Locate the widget you are interested in and drag it from the search results list to the required location in the template you opened.

4 Themes

The graphical appearance of Widget Framework publications can be controlled by selecting **themes**. A theme is a package of CSS files, graphics files (button icons, and so on) and javascript files that determine the appearance and behavior of the templates. A theme can also have **variants**. A variant contains modifications to some aspects of a theme.

You can use themes and variants in whatever way you choose, but the general idea is that you can use different themes for different publications that you want to have completely different appearances, and use variants on those themes for minor variations within publications at the section level (so that the Sports section looks different from the News section, for example).

The Widget Framework is supplied with one theme called **`default`**, which has an empty variant called **`variant`**. It is possible to make themes and variants of your own (if you have the necessary CSS and Javascript programming skills). This manual does not, however, cover the design and creation of custom themes and variants.

Themes and variants can be set on a per-section basis, allowing you to give a different look-and-feel to different parts of a publication. For a description of how to do this, see section 4.1.

4.1 Changing Themes and Variants

To change the theme or variant of a section you must have section editing rights for the sections you want to change. You need to open the section for editing in Content Studio and set some section parameters:

1. Open the Content Studio **Sections** panel and expand the section tree to display the section you are interested in.
2. Select the section and right click on it to display its context menu.
3. Select **Edit** from the context menu.
4. Click on the **Section parameters** tab in the opened section editor.

You can now set the following section parameters:

```
theme.name=your-theme
```

to set the theme, or

```
theme.variant=your-variant
```

to set the variant

> Section parameters are inherited, so when you change a section's theme or variant in this way, the change also applies to all the section's sub-sections. The appearance of an entire publication can be changed by setting the root section's **`theme.name`** parameter.

5 Access Control

With the introduction of the Widget Framework, Content Studio becomes a tool not only for writers and editors, but also for publication designers and layout experts. In some cases, the same people may be responsible for both kinds of work, but in most organisations (especially in larger ones) these jobs will be carried out by specialists.

Since the Widget Framework's templates are in fact sections, you can use the standard Content Engine access control mechanisms to enforce this division of labour. All you need to do is create separate user groups for editorial and design staff, and then assign them different section-level access rights. You might, for example, assign the following access rights:

- Design staff: **`editor`** or **`journalist`** access to templates (the **`config`** section and its subsections), **`reader`** access to content sections.
- Editorial staff: **`editor`** or **`journalist`** access to content sections, no access to templates.

You can, of course, specify access rights in more detail, creating sub-groups of designers with greater or lesser access to different templates, and give some individuals specialised access rights.

For a detailed description of Content Engine access control and how to use it, see Editing Users and Persons in the **Escenic Content Engine Publication Administrator Guide**.

6 How To...

This section contains some step-by-step descriptions of how to carry out specific tasks using the Widget Framework. The first few examples are based on the following scenario:

- Your publication has an existing section called **`Sports`**, based on a template called **`config.default.section.sports`**.
- You want to add a new section under the **`Sports`** section called **`Tennis`**.
- You want the **`Tennis`** section to have a slightly different layout to the other **`Sports`** sections, so you need to create a new template for it.

In order to achieve these objectives you need to:

1. Create a new **`Tennis`** section (see section 6.1).
2. Create a template defining the layout for your new section (see section 6.2).
3. Edit the new template (see section 6.3).

When you have carried out these tasks, add some content to your **`Tennis`** section and view the results in a browser. You should see that the **`Tennis`** section's layout is different from that of other **`Sports`** sections.

6.1 Create a Section

To create a new **Tennis** section:

1. Start Content Studio and log in as a user with **`editor`** access to ordinary sections.
2. Create your new **`Tennis`** section as a subsection of the **`Sports`** section. For general information on how to use Content Studio, see the Escenic Content Engine User Guide. Set the new section's **Unique section name** to **`tennis`**.

6.2 Create a Template

To create a new template for your **`Tennis`** section:

1. Start Content Studio and log in as a user with **`editor`** access to templates.
2. Follow the general instructions in section 2.1 (steps 1 to 5). Create your new template as a subsection of **`config.default.section.sports`**, and set its properties as follows:

   Section name
      `config.default.section.tennis`

   Unique section name
      `config.default.section.tennis`

   Relative directory URI
      `config_default_section_tennis`

> The last section of the template's **Unique section name** must **exactly match** the **Unique section name** of the section it defines the layout for. If this is not the case, then the template will not be found by the widget framework and will not be used.

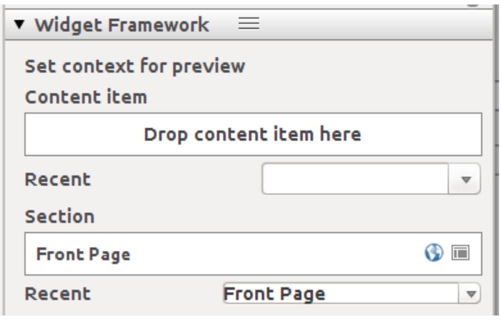
6.3 Edit a Template

To edit your `tennis` template:

1. Start Content Studio and log in as a user with `editor` or `journalist` access to templates.
2. Find the new template (`config.default.section.tennis`) and open it.
3. You will see that the new template is empty: this means that it inherits all of its layout from its parent, `config.default.section.sports`. Open `config.default.section.sports`, copy the contents of its `Main` area, and paste them into the `Main` area of `config.default.section.tennis`.
4. Now make a change to the widgets you have pasted into `config.default.section.tennis`: simply delete one of the widgets, for example, or replace it with a different type of widget.
5. Click on `Save` or `Publish` to save your changes.

6.4 Preview a Template

You can preview a template you are working on in Content Studio in just the same way as editors can preview content items and section pages. The main difference is that in order to preview a template, you need to supply some content as well. Therefore, when you display a templates **Preview** tab, the options panel on the right contains an additional **Widget Framework** section where you can specify the content you want to use for preview purposes:



Widget Framework

Set context for preview

Content item

Drop content item here

Recent

Section

Front Page

Recent: Front Page

You can simply drop a content item or section page in the appropriate field, or select one you have used earlier from the corresponding **Recent** field. You can change the selected content item or section page to see what the template looks like with different content.

6.5 Enable Master Template Switching

Master template switching is a technique you can use to provide editorial staff with a very quick way to switch between different layouts. A typical use case is "instant" reformatting of the front page when a big story breaks. Under normal news conditions a "default" layout gives two or three top stories broadly similar prominence. When a big story breaks, however, editorial staff want to switch to a "breaking news" layout in which much greater prominence is given to the number one story, and less to the other top stories.

Master template switching allows editorial staff to make this kind of change by simply changing a section page option in Content Studio.

To set up master template switching:

1. Create the master templates you want to switch between (say **`config.default.master.frontpage.default`** and **`config.default.master.frontpage.breaking`**). These master templates should contain sets of correctly configured widgets that you have tested in the location where you intend to use them.
2. Add a template switching option to one of the groups on the required section page. To do this you have to edit your publication's **`layout-group`** resource and add a structure something like this:

   ```
   <group name="topStories">
     ...
    <ct:options>
       <ct:field name="masterconfig" type="enumeration">
         <ui:label>Select page layout</ui:label>
         <ui:description>Master configuration option for the group</ui:description>
         <ct:enumeration value="frontpage.default">
           <ui:label>Default</ui:label>
         </ct:enumeration>
         <ct:enumeration value="frontpage.breaking">
           <ui:label>Breaking news</ui:label>
         </ct:enumeration>
         <ui:value-if-unset>frontpage.default</ui:value-if-unset>
       </ct:field>
     </ct:options>
     ...
   </group>
   ```

   The important points about this option definition are:

   - It **must** be called **`masterconfig`**.
   - It should be an **`enumeration`** field with one **`enumeration`** element for each master template that you want to switch between.
   - The **`value`** attributes of the **`enumeration`** elements must be the names of the master templates you have created, but without the **`config.default.master`** prefix.

   See here in the **Escenic Content Engine Template Developer Guide** for information about the **`layout-group`** resource and how to edit it.
3. Add a Master widget to the template used by the section page.
4. Set the following fields in the Master widget:

   Title

   Set to anything you like.

   Master section unique name

   Set this to the name of any one of your master templates (for example **`config.default.master.frontpage.default`**). This is the template that will get used if the Master widget is used on a section page that has no **`masterconfig`** option.

Master group name

Set this to the name of the section page group to which you added the `masterconfig` option (`topStories` in the example shown above) Make sure that you use the **name** of the group (as specified in the `layout-group` resource) and not its **label** as displayed in Content Studio. These may be the same, but are not necessarily so.

Make sure that all the changes you have made are published. You should now be able to switch between the layouts you have defined by simply changing the value of the `masterconfig` option you added to the section page and pressing **Publish** in Content Studio.

6.6 Add Search Functionality

To add search functionality to a typical publication you need to:

- Design and create a search results page
- Add a search box to every page

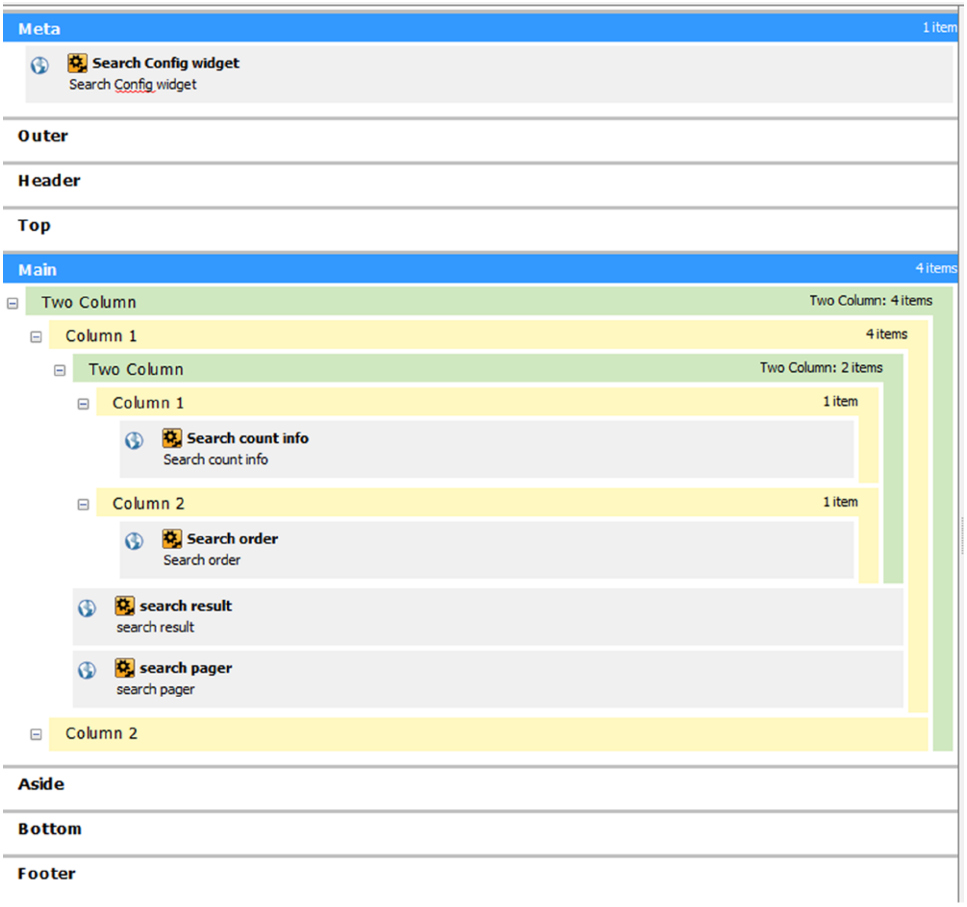
These tasks are described below.

6.6.1 Creating a Results Page

To create a results page:

1. Create a search results section as a child of the publication's home section. Set the section's **Section unique name** property to `search`.
2. Create a template for the search results page as a child of the publication's `config.default.section` template. The new template's **Section unique name** must be set to `config.default.section.search`.
3. Create a Search Config widget and configure it according to your needs, then publish it.
4. Drag the Search Config widget into your search template's **Meta** area.
5. Create a Search Component widget, set **Title** to a name of your choice (`Search Results`, for example).
6. On the **General** tab, set **Component** to **Results List**.
7. Publish the widget.
8. Drag the Search Results widget into your search template's **Main** area (probably inside a group somewhere).
9. If you want other search components on your search page, create other kinds of Search Component widgets (a result counter and a pager control, for example) and place them in your search template.
10. Publish the template.

The following screenshot shows a typical search results page template:



Meta 1 item

Search Config widget
Search Config widget

Outer

Header

Top

Main 4 items

Two Column — Two Column: 4 items

Column 1 — 4 items

Two Column — Two Column: 2 items

Column 1 — 1 item

Search count info
Search count info

Column 2 — 1 item

Search order
Search order

search result
search result

search pager
search pager

Column 2

Aside

Bottom

Footer

6.6.2 Adding a Search box to Pages

To add a search box to the pages of your publication.

1. Create a Search Component widget.
2. On the **General** tab, set **Title** to a name of your choice (`Search Box`, for example).
3. On the **General** tab, set **Component** to **Search Box**.
4. On the **Search Box** tab, set **Target section** to point to the search results page you have created (see section 6.6.1). If required, set other fields on this tab as well.
5. Publish the widget.
6. Open your `config.default` template.
7. Drag your new search box template into the template (usually into the **Header** or **Outer** area).
8. Publish the template.

Adding your search box to the `config.default` means that it will appear on every page of your publication. Should you not want this, you would need to add it to a different template or templates.

6.7 Load Content In-line

The Widget Framework supports in-line loading of content items. What this means is that when a user clicks on a link (for example, a teaser link on a section page), then instead of a new page being loaded, the content item referenced by the teaser is loaded into an area on the current page.

In order to achieve this behavior you need to do two things:

- Configure a **Teaser** or **Teaser view** widget to perform in-line loading instead of normal linking. You do this using the widget's **Link settings**, which you will find on its **Default** tab. See The Default Tab.
- Create one or more `config.inline.article` templates for displaying the in-line content. See section 2.2.1.1 for further information.

6.8 Enable/Disable Profiling

The Escenic Content Engine has a JSP profiling facility that can be useful for tuning purposes. Profiling is not enabled by default Widget Framework publications because it carries a small performance penalty (less than 1%).

You can turn on profiling by setting the following section parameter (usually in the root section of your publication):

```
jsp.statistics=on
```

You can turn it off again for part of a publication by setting it to `off` in a particular section:

```
jsp.statistics=off
```

6.9 Display "Mega-Dropdown" Using Menu Pane Group

Displaying a menu directly in a Menu Pane group enables more complex menu layouts than using a Menu widget. You can easily create a menu containing "mega-dropdown" panes with complex multi-column layouts.

To display a menu in this way, add a Menu Pane group to your template. In the Menu Pane's group options set the following fields:

Reference
Set this option to **Sections in menu**.

Menu name
Enter the name of a menu you have created for this publication using the Content Engine's Menu Editor plug-in.

You can now add groups, areas and widgets to the Menu Pane group to define the layout of the drop-down pane you want to be shown for the sections in menu. You can use all the same components that you would use when creating a section page layout: Data Sources, Teaser widgets and so on. When the user selects a section from the menu, then the selected section is supplied to these components as the context section, and a pane layout is created.

The drop down pane is only displayed for the **sections** in the menu: other elements such as articles and links are simply rendered as links in the usual way.

> Since sections in the displayed menu are rendered using the content of the Menu Pane, you cannot use this technique to display multi-level menus: any subsections in the menu will be ignored.

6.10 Use a Context Section Group

You can use a Context Section group to:

- Display content from a specific section on the current page
- Display content from multiple sections on the same page
- Display content from a content item's owning section on a content page

When used to display content from multiple sections, the Context Section group works in a similar way to the Menu Pane group, in that it relies on a menu created using the Menu Editor plug-in. The Context Section group, however, does not use the menu as a menu, but just as a list of sections.

To use a Context Section group, you place it in the required location on the page and then set the following options:

Use

Set this to either **Specific section**, **Sections in menu** (to display content from multiple sections) or **Home section of context article** (for use on content pages).

Section

If you set **Use** to **Specific section** then select the section you want to use as the context section

Menu name

If you set **Use** to **Sections in menu** then enter the name of a menu you have created for this publication using the Content Engine's Menu Editor plug-in.

You can now add groups, areas and widgets to the context section group to define the layout. If you set **Use** to **Sections in menu**, then the layout you have defined will be repeated for every section in the menu you specified. If you set **Use** to either **Specific section** or **Home section of context article** then the layout will be used only once, to display the content from either the specified section or the current content item's home section.

> Lazy loading can be applied to Context Section groups, but only to the group as a whole, not to the individual items (groups, areas and widgets) placed inside it.

6.11 Enable Editorial Teaser Layout Control

The Widget Framework is in general designed to enforce a separation of concerns between editorial staff and publication designers, so that editorial staff have very little control over the layout and appearance of a publication. You can, however, provide editorial staff with some limited control over the layout of section page teasers. You can do this for teasers that are displayed by Teaser Views under the control of either View Picker or Teaser Grid widgets.

Enabling editorial control allows editorial staff to override the following aspects of teaser display:

- The Teaser view used
- The font size used for the teaser's title and/or lead text
- Whether or not an image is displayed with the teaser
- The image aspect ratio
- The CSS style used for the teaser

Editorial staff are able to change these display parameters by setting options on the section page groups in which they desk the teaser content items (see chapter 7 for a description of these options).

6.11.1 Editorial Control of View Picker Teasers

To enable editorial control over View Picker teasers you must:

1. Create a View Picker widget and configure it to use a data source containing a "by section page group" query that selects content items from a `Curated view` section page group.
2. Ensure that the View Picker widget's **Rendering mode** option is set to **Iterate with related view**.
3. Ensure that the View Picker widget's **Disable view adjustments by editor** option is **not** checked.
4. Place the View Picker widget in the required location on the required section page template.
5. If you want to allow editors to select Teaser views with the **Use view** option (see section 7.1.1), then you need to ensure that **Allow editors to select this view** is set in the required Teaser views.

6.11.2 Editorial Control of Teaser Grid Teasers

The Teaser Grid widget is explicitly designed to give editors control over teasers. To set up a Teaser Grid widget for use:

1. Create a Teaser Grid widget and configure it to use a data source containing a "by section page group" query that selects content items from a `Grid` section page group.
2. Place the Teaser Grid widget in the required location on the required section page template.
3. Make sure that the `Grid` group referenced by the Teaser Grid widget's data source actually exists (that is, either create it yourself or inform the editorial staff who control the relevant section page that they need to create it).
4. If you want to allow editors to select Teaser views with the **Use view** option (see section 7.2.1), then you need to ensure that **Allow editors to select this view** is set in the required Teaser views.

6.11.3 Defining Override CSS Styles

The **Teaser Style** option with which editorial staff can set CSS styles (see section 7.2.1) displays a list of predefined options. The options displayed in this list can be modified and extended by adding a configuration file to one of your installation's Content Engine configuration layers. Each style option is defined by two entries in the file, one to define the label displayed in the list of options and one to specify the CSS classes that will be applied if it is selected:

```
options.curatedStyle.exclusive.value=well well-xs
options.curatedStyle.exclusive.label=Exclusive
```

Editing and deploying configuration layer properties files is more of a developer task than a publication designer task, and the `/com/escenic/framework/ui/Enumerations.properties` file that needs to be added is also used in the configuration of custom data sources, so this process is described more fully in the **Widget Framework Developer Guide**, see http://docs.escenic.com/widget-dev-guide/3.4/query_definition_enumeration_field.html.

6.12 Configure Video Advertising Services

The Widget Framework supports the inclusion of advertisements in video streams from on-line advertising services that can supply ads in the following industry-standard formats:

- **VAST**
- **Google IMA**

Currently, you can only include ads in your videos if you use JW Player as your video player, and the only kind of ads currently supported are **pre-roll** ads. Pre-roll ads are advertisements that play before a requested video is played.

You can configure the inclusion of pre-roll advertisements in videos at three different levels:

- At publication/section level, by setting the `wf.media.ad.preroll.enable` section parameter on a section to `true`.
- At widget level, by setting the **Enable pre-roll** option on the **Advertisement** tab of a Media widget, Teaser widget or Teaser view
- At content item level, setting the **Enable pre-roll** option on the **Advertisement** tab of a Video content item

In all three cases you also have to supply a **Service** content item. A Service content item is designed to hold the URL of an external service such as an advertising service. It has a field for holding a base URL, plus an array of fields for holding parameter names and values, from which a complete URL including a query string can be constructed. For a full description, see section 6.12.1. How you supply the service content item varies as follows:

- At publication/section level you specify the ID of the required Service content item by setting the `wf.media.ad.preroll.service.id` section parameter. The ID of a content item is displayed in the **Content Info** panel if you open it in Content Studio.
- At widget and content item level you simply drop the required Service content item into the **Pre-roll ad** relation on the widget's or content item's **Advertisement** tab

You can configure pre-roll inclusion at several of the above level. If settings conflict, then the most specific level wins: content item settings override widget settings and widget settings override section settings.

6.12.1 Defining a Service

A Service content item contains the following fields:

Title

The name of the service.

Service type

The type of service. Currently you can specify the following values:

VAST

Select this for ad services that supply ads in the VAST (Video Ad Serving Template) format.

Google IMA

Select this for ad services that supply ads using (Google Interactive Media Ads) APIs.

Base URL

The base URL of the service (that is the URL without any parameters)

Parameters

An array of parameter settings for the service. Add the parameters required by the service you are configuring. They are all assembled into a query string and appended to the **Base URL** to create a complete service request.

Name

The name of a parameter required by the service you are configuring.

Value

The value to be assigned to this parameter. You can specify this value using a mixture of literal text and JSTL expressions that will be replaced with dynamic values. Note that in addition to the usual `${section}` and `${article}` objects, you also have access to the following special context objects:

`${sectionParams}`

This object offers a shorthand means of accessing section parameters. You can access a section parameter called `x.y`, for example, with the expression `${sectionParams.x.y}`. This is equivalent to `${section.parameters['x.y']}`.

`${mediaContent}`

This object represents the video content item to be displayed.

Fallback

A fallback value to be assigned to this parameter if **Value** evaluates to an empty string.

Encode

Check this option if you want the value you have specified to be URL-encoded.

It also contains a **Base Service** relation on which you can drop another Service content item: the current Service will then inherit the base Service's settings. Local settings override inherited settings. This also applies to the contents of the **Parameters** array: a local parameter setting overrides an inherited parameter with the same name.

This inheritance mechanism makes it easy to create a set of similar Services that all inherit from the same base Service but have slightly different parameter settings.

7 Editorial Layout Control

The Widget Framework is in general designed to enforce a separation of concerns between editorial staff and publication designers: editorial staff are responsible for content and organisation: designers are responsible for layout and functionality. For some organizations, however, this separation can be too strict, and there is a requirement to provide editorial staff some control over layout. This chapter describes the layout controls the Widget Framework can make available to editorial staff.

The functionality described here is available to editors when editing section pages. It mostly needs to be enabled by designers, however, and is only useful if designers have configured other components to make use of it.

7.1 Using Curated View Groups

Curated View is a top-level group type that you can add to section pages by right-clicking on the **Content Area** root and selecting **Insert** > **Insert new Curated view**. You can then drag content items into the group in the usual way. What is special about the Curated view group is that you can then control the appearance of the teasers generated for these content items by setting various display options (see section 7.1.1).

> There no point adding a Curated view group to a section page unless you know that your designers have configured a View Picker widget to display the content items that you add to it. Your usage of Curated view groups must be co-ordinated with the designers.

7.1.1 Curated View Options

If you select a **Curated View** group in a section page editor, then the following options are displayed on the right:

Use view
Use this option to select the Teaser view that will be used for all teasers in the group. Start typing and then select the required teaser view from the displayed options. The view you select is used to override the default Teaser view(s) configured by the publication designer.

Image
Use this option to select the image version to be used for any images in the teasers in the group. You can also suppress the display of images by selecting the **Hide** option.

Teaser style
Select a style to be applied to all teasers in the group.

Title size
Select a font size to be applied to the titles of all teasers in the group.

Lead text size
Select a font size to be applied to the lead text of all teasers in the group.

7.2 Using Grid Groups

Grid is a top-level group type that you can add to section pages by right-clicking on the **Content Area** root and selecting **Insert** > **Insert new Grid**. Most of the groups that you can add to section pages in a standard Widget Framework publication are simple groups that cannot be subdivided. Grid groups, however, have a standard set of subgroups representing various column arrangements. You can use them to control how teasers are to be laid out on a page or part of a page. First you subdivide the Grid group by inserting one or more subgroups, and then you drag content items into the subgroups, not the the root Grid.

> There no point adding a Grid group to a section page unless you know that your designers have configured a Teaser Grid widget to display the content items that you add to it. Your usage of Grid groups must be co-ordinated with the designers.

You can insert the following subgroups into a Grid group by right-clicking on the group, then selecting **Insert** > **Insert new [group]**:

Row

A container for Column groups. You can add as many Rows as you like to a Grid. Do not add content items directly to a Row. A number of options are displayed on the right, which you can use to override default teaser formats (see section 7.2.1).

Column

Insert into a Row group. You can add as many Columns as you like to a Row. You can then either fill the columns with content items as required or subdivide it further. A Column group has a **Width** option that you can use to control column width. The total width of all the Columns in a Row should not exceed 12.

Two column

A predefined row containing two columns. Any content items you add to a Two column group are divided between the two columns: the first half are put in column 1, and the second half in column 2. If you add an odd number of content items, then column 1 will get the extra one. A number of options are displayed on the right, which you can use to override default teaser formats and control column widths (see section 7.2.1).

Three column

A predefined row containing three columns. Any content items you add to a Three column group are divided between the three columns: the first third are put in column 1, the second third in column 2 and the remainder in column 3. A number of options are displayed on the right, which you can use to override default teaser formats and control column widths (see section 7.2.1).

Two column (1+N)

A predefined row containing two columns. It works the same way as the Two column group except that only the first content item is allocated to the first column, and all the remaining content items are allocated to the second column. A number of options are displayed on the right, which you can use to override default teaser formats and control column widths (see section 7.2.1).

Two column (N+1)

A predefined row containing two columns. It works the same way as the Two column group except that only the last content item is allocated to the first column, and all the remaining content items are allocated to the second column. A number of options are displayed on the right, which you can use to override default teaser formats and control column widths (see section 7.2.1).

7.2.1 Grid Options

If you select a **Row**, **Two column**, **Three column**, **Two column (1+N)** or **Two column (N+1)** group in a section page editor, then the following options are displayed on the right:

Use view
Use this option to select the Teaser view that will be used for all teasers in the group. Start typing and then select the required teaser view from the displayed options. The view you select is used to override the default Teaser view(s) configured by the publication designer.

Image
Use this option to select the image version to be used for any images in the teasers in the group. You can also suppress the display of images by selecting the **Hide** option.

Teaser style
Select a style to be applied to all teasers in the group.

Title size
Select a font size to be applied to the titles of all teasers in the group.

Lead text size
Select a font size to be applied to the lead text of all teasers in the group.

1st column size
Select the width to be used for the first column in a two or three column group. Remember that the total width available for all columns across the total page width is 12.

2nd column size
Select the width to be used for the second column in a three column group. Remember that the total width available for all columns across the total page width is 12.