Building Grid Layouts Based on a Framework

Talk by Heather Floyd @ uWestFest March 2016
www.HeatherFloyd.com - @HFloyd

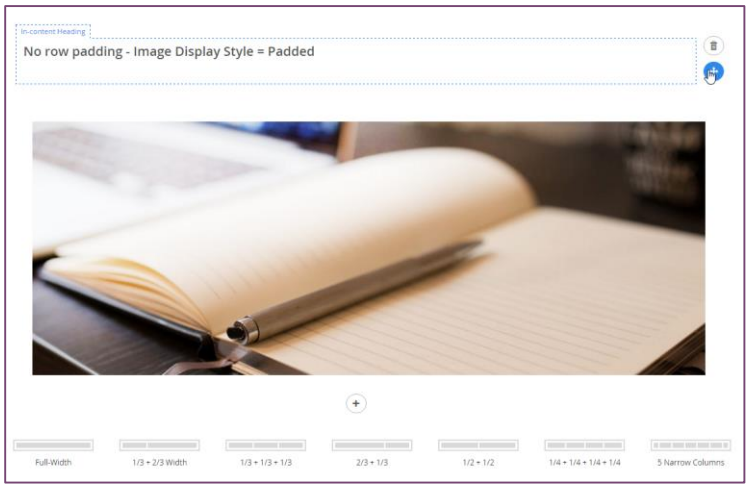
Outline, Notes, & Code Snippets

Welcome

What I mean by "Framework":

- **Strongly-typed models for Doctypes** (ModelsBuilder – now built into core (v. 7.4))
- **Scandza Framework** (proprietary customized models to handle Umbraco things like Media Images + collection of personal "helper" library functions)
- Also – just a **standardized system of development** (even if you aren't using either of the above)

The Umbraco Grid Control



Main Grid Benefits

- A balance between editor power and designer/developer control (proper rendering of HTML/CSS)
- Easier for editors to get fancy customized page designs without having to mess with manually adding HTML and CSS styles (like in an RTE)
- Designed for mobile responsiveness. Supports Bootstrap out-of-the-box, but several other renderers are available, plus the render is completely customizable by developer.

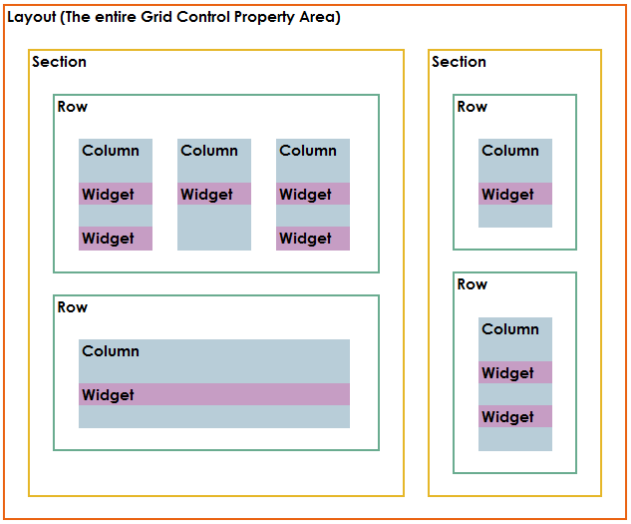
A Brief History of the Grid

- Originally created by Antoine from Barcelona Agency LECOATÍ
- Incorporated into the Core for Umbraco version 7.2 (Dec 2014)
- Updated for Umbraco version 7.4 (Feb 2016)

Grid Structure

"Grid Control" Datatype added to a Document Type

Layout > Sections > Rows > Columns > Widgets (Controls)



Developer configures which Sections and what Row Configurations (arrangements of columns) are available for editors to use. Also, which widgets can be used in each row/column.

On a grid control on a Content node, the entire grid is stored as a JSON blob. This is somewhat convenient, but also poses specific challenges (discussed later).

Design & Specification Challenges - "Think in a Grid"

- Designers need to define page design portions as rows/columns/widgets.
- Can end up with very complex row configurations and widgets if not managed. (Photoshop is easy – CSS and grid architecture is hard!)
- Clients need to change their thinking from "page layouts" to "modularity"

Grid Architecture – Planning Pays Off

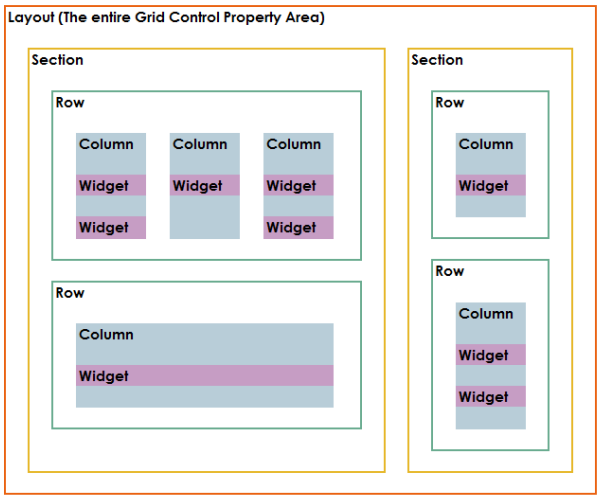
- Breaking up page designs into rows.
- Determining what column arrangements and row options are needed to handle the variety of page elements.
- Defining what widgets are needed, with what properties.

Architecture Tips

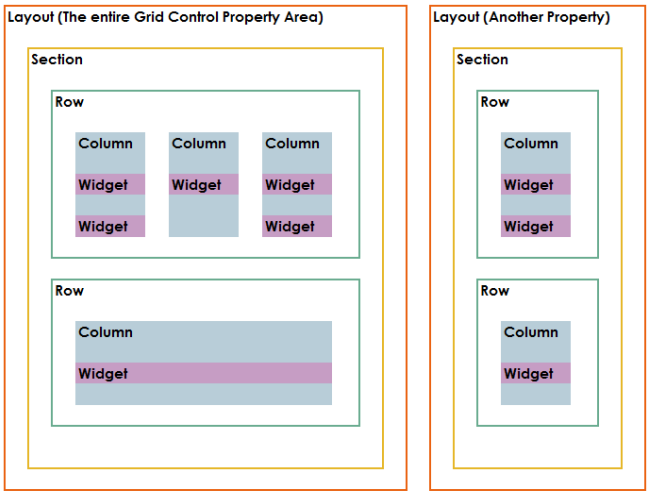
Avoid multiple Sections (aka a choice of Layout) (example: "Full-width Page" & "Page with Sidebar")

- Main reason is future-proofing… you can't change a grid control's Layout on a content node without deleting ALL rows and widgets first. (YIKES!)
- If the over-all template design of the site changes (for instance, to remove any sidebar page layouts), all the content pages will have to be manually reconfigured by editors. (See first point, above.)
- If you want to provide a sidebar… Add two grids to the DocType and have the template render the 2-columns next to one another. Remember – it's always easier to combine things programmatically than to split them!

Instead of this:



Do this:



How much of the Page is a Grid?

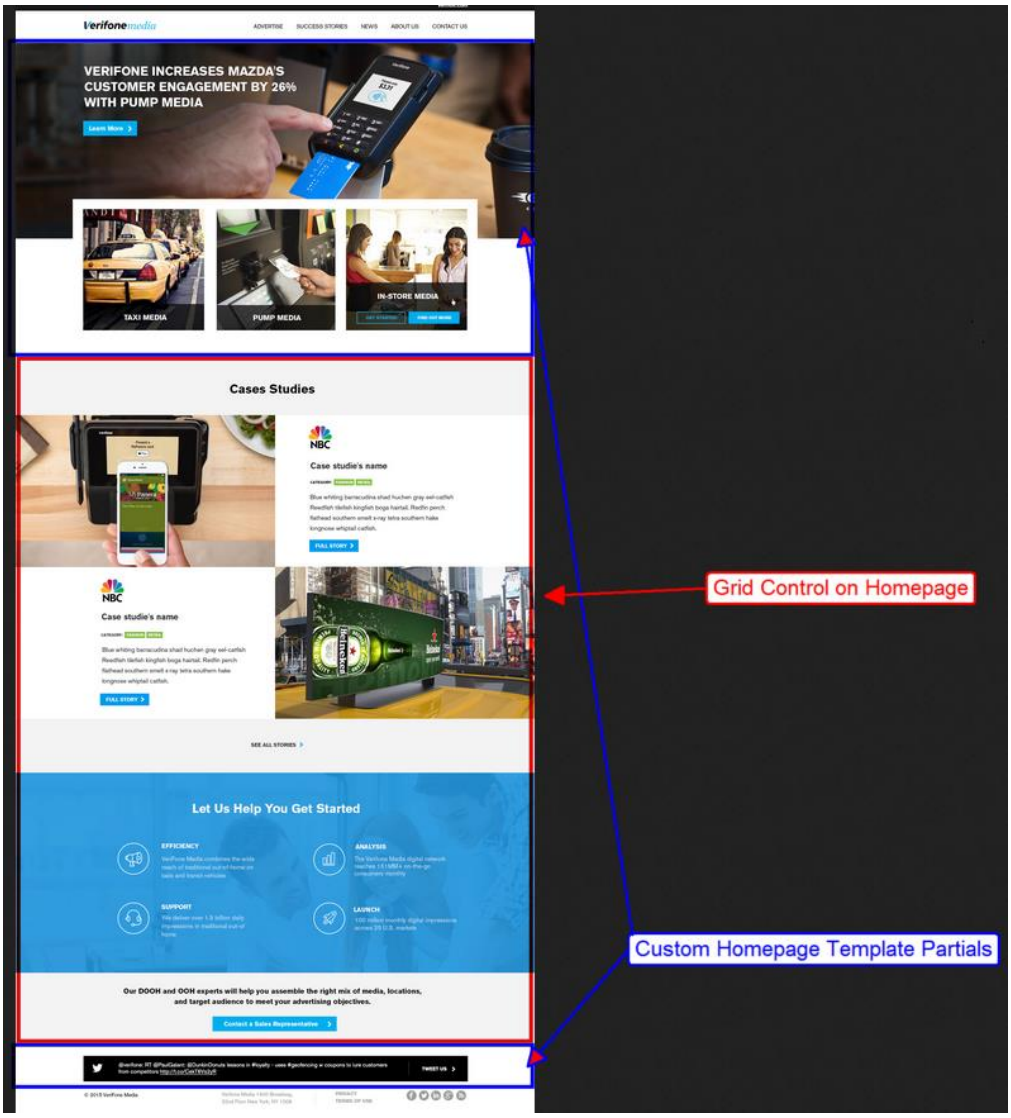
Consider whether your template will be completely a grid – or if only portions of the page will be a grid.

HINT: if a widget would only be used on a single doctype/template, consider **not** building it as a widget.

Examples:

- **Content banner/slider** – Nested Content used on the Doctype instead
- **Twitter bar** – used only on Homepage, requires no editor interaction

Issue: Does it need to be inserted in the middle of grid content? Consider a macro (which can be inserted as a grid widget) to do the rendering rather than a custom widget with properties.



Pick the appropriate number of row configuration options.
Balance between flexibility & complexity.

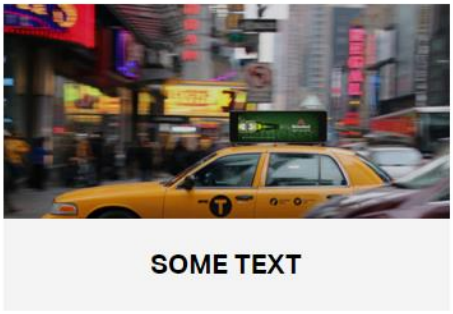
This will be based somewhat on the designs – so you can see how giving the designer some parameters could be helpful…

Consolidate Widgets

If different-looking page elements have the same editable components (for example: Header, Description, Image, Link), create one widget and include a "Layout Style" drop-down (Image on left, image on right, image on top, background image, etc.) and render appropriately.

Same widget with different "Layout Styles":

Layout: Image with Grey Boxed Text (no link)



Layout: Image with Grey Boxed Text (w. link)

Layout: Animated Black Overlay

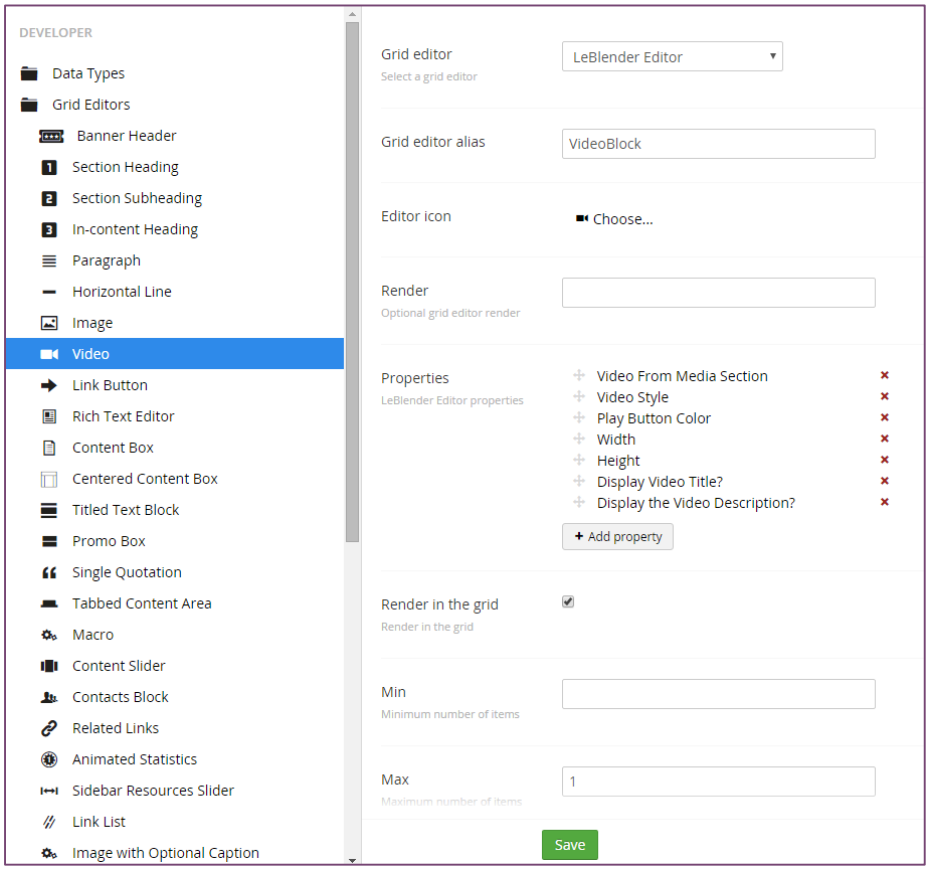
Creating Complex Widgets – without Angular knowledge

You can certainly build all your custom widgets in Angular, so you JavaScript fiends – have at it.

Check out Marc Stöcker's UK Festival 2015 talk for more about how to do that *[see Resources, below, for link]*

I'm really a back-end focused developer with slender Angular skills, AND I like to keep things as standard and simple as possible, so I use LeBlender for my customizing needs. *[see Resources for link]*

Even if you plan to build with Angular, install LeBlender just for the back-office area which makes it easy to see all the configured widgets, and do things like update their icon or name without having to muck about in a potentially large JSON config file.



LeBlender Custom Widgets

Easy to add any DataType as a property on the grid widget.

Process options via back-end cshtml files.

Built-in support for "1-to-many" objects (like slides) or use something like Nested Content in a custom Datatype (good if you have "widget-level" (like a header or display option) properties as well).

Using Strongly-typed models with Grid

Skybrud.Umbraco.GridData

Grid properties can be returned as a "Grid DataModel object *[see Resources for link]*

Customized Doctype model:

```
namespace VERImedia.Web.Models
{
    using Skybrud.Umbraco.GridData;
    using Umbraco.Web;
    using Zbu.ModelsBuilder;

    public partial class TextPage
    {
        /// <summary>Content</summary>
        [ImplementPropertyType("PageContent")]
        public GridDataModel PageContent
        {
            get { return this.GetPropertyValue<GridDataModel>("PageContent"); }
        }

    }
}
```

Used in a View:

```
@using VERImedia.Web.Models
@using Skybrud.Umbraco.GridData.Extensions

@inherits UmbracoViewPage<BaseModel<TextPage>>
@{
    Layout = "MasterInnerPage.cshtml";
}

@if (Model.Content.PageContent.IsValid)
{
    <!--START GRID-->
    @Html.GetTypedGridHtml(Model.Content.PageContent, "ScandzaBootstrap")
    <!--END GRID-->
}
```

(GetTypedGridHtml → Use the render file specified)

Custom Models in Widget Editors

Inside grid editor .cshtml files you can use your custom models – ex: MediaImage, Nested Content, Content Picker -> custom DocType model

```
@using Umbraco.Core.Logging
@using VERImedia.Web.Helpers
@using VERImedia.Web.Models

@inherits UmbracoViewPage<Lecoati.LeBlender.Extension.Models.LeBlenderModel>

@if (Model.Items.Any())
{
    var umbracoHelper = new UmbracoHelper(UmbracoContext.Current);

    //Only supports a single item
    var item = Model.Items.FirstOrDefault();

    if (item != null)
    {
        //Media Video Info
        MediaVideo mediaVideo = null;
        var previewImageUrl = "";
        var videoIFrameUrl = "";
        var title = "";
        var text = "";
        var videoName = "";

        try
        {
            var videoValId = item.GetValue("MediaVideo");
            var videoMedia = umbracoHelper.TypedMedia(videoValId);
            mediaVideo = PropertyConversion.ContentToMediaVideo(videoMedia);

            //Final Video Info
            videoIFrameUrl = mediaVideo.VideoUrl;
        }
        catch (Exception ex)
        {
            var msg = string.Format("{0} [Property '{1}']", "Error", "MediaVideo");
            LogHelper.Error<UmbracoViewPage>(msg, ex);
        }

        @*Hard-coding width and height for demo, would otherwise come from properties.*@
        var width = "720";
        var height = "405";

        <div class="embed-responsive">
            <iframe src="@videoIFrameUrl" width="@width" height="@height" frameborder="0" webkitallowfullscreen="" mozallowfullscreen="" allowfullscreen=""></iframe>
        </div>

    }
}
```

Converting media id to custom type "MediaVideo"

Using custom type property

Elegant Error-Handling for Grid widgets

Properties can change (esp. during active development) test each one and react appropriately.

If content exists, then you add a new property, and try to access it in the Editor rendering, that old content will show an error (because the value is missing from the JSON blob.)

You probably don't want to blow up the widget display if a non-needed property is missing, but you might want to log it, at least, so the content node could be updated later.

```
@if (Model.Items.Any())
{
    var umbracoHelper = new UmbracoHelper(UmbracoContext.Current);

    //Used for error logging
    var thisWidgetName = "TitledTextBlock";
    var nodeUrl = "UNKNOWN (Back-office)";
    if (!Mvc.IsRenderingInBackOffice(this.Request.Url))
    {
        nodeUrl = umbracoHelper.AssignedContentItem.Url;
    }
    var defaultErrorMsg = string.Format("ERROR in {0} Grid Editor on page '{1}': ", thisWidgetName, nodeUrl);

    //Only supports a single item
    var item = Model.Items.FirstOrDefault();

    //Header
    var header = "";
    try
    {
        header = item.GetValue("Header");
    }
    catch (Exception ex)
    {
        var msg = string.Format("{0} [Property '{1}']", defaultErrorMsg, "Header");
        LogHelper.Error<UmbracoViewPage>(msg, ex);
    }
```

"Grid on Steroids" - Row Options & Customizing the Grid Rendering File

Things you can test for while rendering the grid control in a view:

- Layout/Section names
- Row names/options
- Widgets used inside a row (which ones, quantity of widgets in a row)
- Properties on the widgets used

Things I have customized

Adding the row layout name as a css class to the row div

```
@helper StandardRowRendering(dynamic row, bool singleColumn)
{
    string rowName = row.name != null ? row.name.ToString() : "";
    var rowClass = rowName.MakeCodeSafe("-", true);
    var containerClass = GetContainerClass(row);

    <div @RenderElementAttributes(row)>
        @RenderOptionalOverlay(row)
        @Umbraco.If(singleColumn, string.Format("<div class=\"{0}\">", containerClass))
        <div class="row @rowClass">
            @foreach (var area in row.areas)
            {
                <div class="col-md-@area.grid column">
```

Added a row option to switch the container class for that row from standard to full-bleed

```
@helper StandardRowRendering(dynamic row, bool singleColumn)
{
    string rowName = row.name != null ? row.name.ToString() : "";
    var rowClass = rowName.MakeCodeSafe("-", true);
    var containerClass = GetContainerClass(row);

    <div @RenderElementAttributes(row)>
        @RenderOptionalOverlay(row)
        @Umbraco.If(singleColumn, string.Format("<div class=\"{0}\">", containerClass))
        <div class="row @rowClass">
            @foreach (var area in row.areas)
            {
                <div class="col-md-@area.grid column">
```

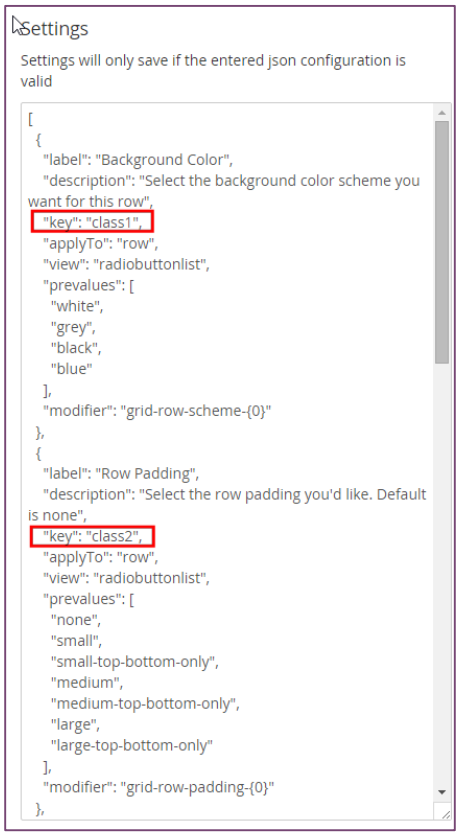
```
public static string GetContainerClass(dynamic contentItem)
{
    var className = "container";

    var allAttributes = (List<JProperty>)GetAttributesList(contentItem);

    var containerProp = allAttributes.Where(n => n.Name == "special-container").FirstOrDefault();

    if (containerProp != null)
    {
        if (containerProp.Value.ToString() == "Yes")
        {
            className = "container-fluid";
        }
    }

    return className;
}
```

Adding background color + background image options, which, if combined, created a colored overlay on the background image


```
@helper RenderOptionalOverlay(dynamic row)
{
    var HasBgImage = false;
    var HasBgColor = false;
    var colorClass = "";

    //Check for Image
    JObject style = row.styles;
    if (style != null)
    {
        foreach (JProperty property in style.Properties())
        {
            if (property.Name == "background-image")
            {
                HasBgImage = true;
            }
        }
    }

    if (HasBgImage)
    {
        //Check for Color
        JObject cfg = row.config;
        if (cfg != null)
        {
            foreach (JProperty property in cfg.Properties())
            {
                //Newtonsoft.Json.Linq.JValue token = property.Value<JValue>();
                string value = property.Value.ToString();

                if (value.Contains("grid-row-scheme-"))
                {
                    colorClass = value;
                    HasBgColor = true;
                }
            }
        }
    }

    if (HasBgImage & HasBgColor)
    {
        <div class="color-overlay @colorClass"></div>
    }
}
```

Added Support for Multiple 'Classes' on a row

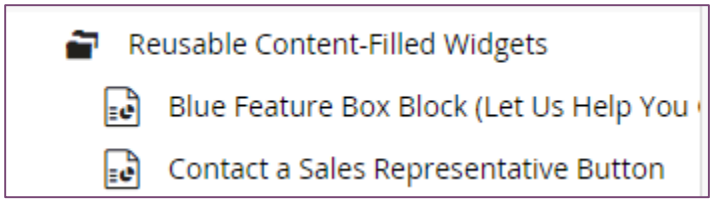
Changed the way the "classes" attribute is compiled so that multiple separate properties could append different classes (just by adding a number to "class" for the option)



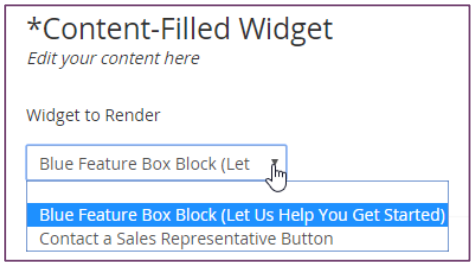

```
public static IEnumerable<string> CompileSettings(IEnumerable<JProperty> Properties)
{
    var attrs = new List<string>();

    var lastAttrib = "";
    var attribValueCompiled = "";

    foreach (JProperty property in Properties.OrderBy(p => p.Name))
    {
        //strip out "special" properties - which aren't HTML attributes, just markers we use in special row processing
        if (!property.Name.StartsWith("special-"))
        {
            var thisAttrib = StripNumbers(property.Name);

            if (thisAttrib == lastAttrib)
            {
                attribValueCompiled = string.Format("{0} {1}", attribValueCompiled, property.Value.ToString());
            }
            else
            {
                if (lastAttrib != "")
                {
                    attrs.Add(lastAttrib + "=\"" + attribValueCompiled + "\"");
                }

                attribValueCompiled = property.Value.ToString();
            }

            lastAttrib = thisAttrib;
        }
    }

    attrs.Add(lastAttrib + "=\"" + attribValueCompiled + "\"");

    return attrs;
}
```

Reusable content-filled widgets

Involves custom DocType for the widget (with a grid control on it) and a Content area for them to be stored…



And a DataType picker for the widget that is used on pages with grid controls:



Then the renderer looks for that control type and renders the external node's grid widget in its place:

```
@helper StandardRowRendering(dynamic row, bool singleColumn)
{
    string rowName = row.name != null ? row.name.ToString() : "";
    var rowClass = rowName.MakeCodeSafe("-", true);
    var containerClass = GetContainerClass(row);

    <div @RenderElementAttributes(row)>
        @RenderOptionalOverlay(row)
        @Umbraco.If(singleColumn, string.Format("<div class=\"{0}\">", containerClass))
        <div class="row @rowClass">
            @foreach (var area in row.areas)
            {
                <div class="col-md-@area.grid column">
                    <div @RenderElementAttributes(area)>
                        @foreach (var control in area.controls)
                        {
                            if (control != null && control.editor != null && control.editor.view != null)
                            {
                                if (!ControlIsReusableWidget(control))
                                {
                                    <text>@Html.Partial("grid/editors/base", (object)control)</text>
                                }
                                else
                                {
                                    //special handling
                                    var reusableControl = GetReusableControl(control);
                                    <text>@Html.Partial("grid/editors/base", (object)reusableControl)</text>
                                }
                            }
                        }
                    </div>
                </div>
            }
        </div>
        @Umbraco.If(singleColumn, "</div>")
    </div>
}
```

Most design challenges can be solved – with some creativity

"Gotchas" to be aware of

Courier 2.52.3 – The latest version handles things a lot better, but if you have highly customized widgets using custom datatypes (like Nested Content), you will want to do some testing, and might need to write some data resolvers to help those through the Couriering. All the "standard" grid controls are completely supported at this time.

Examine – Grid content is stored in a JSON blob, which includes the property information (so a search for a word which you use as a property name will turn up that page, even if that word isn't visible as content). Workarounds: Customize the data that goes into the Index (with some help from our friend the Skybrud. GridDataModel), or use a full-text indexing solution.

Check out Marc Stöcker's UK Festival 2015 talk for more details on these and other possible issues.

Also, Umbraco and Courier are improving rapidly, so do some testing of your own.

Conclusion

If your client is demanding page design flexibility, and you'd like to reduce the number of separate templates you are creating, the grid is an excellent option, and can fit right into your current development processes.

Resources

Recommended Code & Packages

LeBlender
https://our.umbraco.org/projects/backoffice-extensions/leblender/

Skybrud. GridDataModel
https://github.com/skybrud/Skybrud.Umbraco.GridData

Nested Content
https://our.umbraco.org/projects/backoffice-extensions/nested-content/

Full-Text Indexer for Examine
https://fulltextsearch.codeplex.com

Recommended Info

Our.Umbraco Documentation for "Grid Layout" Property Editor
https://our.umbraco.org/documentation/getting-started/backoffice/property-editors/built-in-property-editors/grid-layout

"The Grid: Structure, Settings, Editors and Use Cases" by Marc Stöcker @ UK Festival October 2015
https://www.youtube.com/watch?v=QvzwilqYOp4

Blog posts about Grid evolution & related Umbraco releases

Umbraco 7.2 grid updates
http://umbraco.com/follow-us/blog-archive/2014/6/26/umbraco-72-grid-updates/

Umbraco 7.2.0 beta out now
http://umbraco.com/follow-us/blog-archive/2014/10/28/umbraco-720-beta-out-now/

Umbraco 7.2 Beta 2
http://umbraco.com/follow-us/blog-archive/2014/11/10/umbraco-72-beta-2/

Umbraco 7.2 released
http://umbraco.com/follow-us/blog-archive/2014/12/4/umbraco-72-released/

The future of Grid Layouts
http://umbraco.com/follow-us/blog-archive/2015/11/17/the-future-of-grid-layouts/

7.4.0 beta: It's like a whole new backoffice
http://umbraco.com/follow-us/blog-archive/2015/12/17/740-beta-it-s-like-a-whole-new-backoffice/

Our Major Minor - introducing Umbraco 7.4
http://umbraco.com/follow-us/blog-archive/2016/2/11/our-major-minor-introducing-umbraco-74/