ASET
Arizona Strategic Enterprise Technology

Digital Government: Agency Platform

Drupal 7 – Views – Creating a Gallery

Version 1.0

Executive Summary

Summary of Changes

This section records the history of significant changes to this document. Only the most significant changes are described here.

Version	Date	Author	Description of change
1.0	08/05/2016	Danielle Wilcox	Created Content

Table of Contents

EXECUTIVE SUMMARY **1**
SUMMARY OF CHANGES 1
GENERAL INFORMATION **3**
REQUIRED MODULES **3**
SKILL LEVEL: BEGINNER/INTERMEDIATE **3**
THE EXAMPLE SCENARIO **4**
PREREQUISITES TO COMPLETE **4**
CREATE A STANDARD GALLERY [BEGINNER] **5**
UPDATE THE STANDARD GALLERY [BEGINNER/INTERMEDIATE] **8**
ADDING MASONRY EFFECTS TO A GALLERY [INTERMEDIATE] **12**

Drupal 7 – Views – Creating a Gallery

General Information

Galleries are a collection of items that is often sought out for with sites handling photos, graphics, ecommerce, or simple showcasing in general. Out of the box, the Drupal 7 environment does not have a way to handle making and displaying galleries. In the newest Drupal environment (8), galleries are a bit more possible due to the inclusion of the **Views** module to the core system.

With some minor additions, creating a gallery in the Drupal 7 environment is very easy to accomplish. The hardest part is to make it responsive.

Required Modules

The following modules are required to create a basic gallery:

- **Views** - This module is a quick way to develop a list or information bundles based on certain filters or fields defined and updates automatically, as well as provide an option to search through the list. For example, this module can create a basic site map of all pages under a specific content type.

- **CSS Injector** – This module allows site administrators to add bits of custom CSS code to the Drupal website without making it required to edit the theme's standard CSS files on the server.

For a more complex system in creating a Masonry based gallery the following must also be installed:

- **Masonry API** – An API module developed for integrating the jQuery Masonry plugin. Masonry is a JavaScript grid layout library. It works by placing elements in optimal position based on available vertical space, sort of like a mason fitting stones in a wall.

- **Masonry Views** – This module provides the ability to use the Masonry API with the Views module.

- **Libraries** – (v 2.x) This module is an API based module for providing a common repository for all external libraries.

Skill Level: Beginner/Intermediate

- At a **Beginner** level, this guide requires some experience in the Drupal environment with spinning up Views.
- At an **Intermediate** level, this guide requires more experience with applying CSS and installing libraries.

The Example Scenario

Throughout the documentation, various images may contain additional captions as an example in order to provide a simple experience to the reader. This allows the reader to replicate what is shown as part of their learning process, if it is necessary.

For this scenario, the following will be referenced:

> *The local zoo has requested to create a gallery to display a main image they have from their upcoming events that will link to that event page. They've yet to determine if additional images will be used on the event page, but they're anxious to get a working gallery up that will link the visitor to the event information. Thankfully, they've gone ahead and created the content type already to funnel the events to with the following fields in the content: Event name (Title), Image field (Image), and a taxonomy reference (Tags).*

Prerequisites to Complete

Prior to beginning this guide, it will be assumed that the below list has been created, or may be in the process of creation as you reach each section. Each item listed will be provided in generic form, but also with alignment to the example scenario in parenthesis. This guide can be completed within one content type, but it is possible to add and use additional ones for further complexity. Remember, all fields can be reused in different content types and can be found under **Add existing field** when using the *Manage Fields* section.

1) Generate the content type (Gallery Item)
    a. For the example scenario fields: Title, Image (image field), Body, Tags (term reference/taxonomy).

2) Create an empty taxonomy list.
    a. As users enter content, this list will auto generate when entering their own taxonomy terms.

3) [OPTIONAL] URL Alias patterns have been created; ideal in such that when adding custom CSS, blocks, or Panels, it makes a wild card selection very useful. (event/[node:title])

4) [OPTIONAL] Prepopulate with the appropriate content type to generate some items/examples to work with for visualization.

5) [**INTERMEDIATE**] If planning to use the Masonry type gallery, the below must also be completed prior to setting anything up. In case the steps become out of date or to gather necessary/updated plugin files, check the Drupal Masonry API installation notes.

    a. Ensure that **Libraries** module is enabled/installed (v 2.x).

b. Ensure that jQuery meets the recommendation version (v1.6 or greater). This can be checked by visiting the Status Report of the Drupal environment (*admin/reports/status*).

c. Download the appropriate Masonry plugin files at the Drupal Masonry API page. (v3)

d. Save – Rename – Relocate the file to the site server:
…/code/sites/all/libraries/**masonry.pkgd.min.js**

e. Download the appropriate library directory for imagesloaded at the Drupal Masonry API page.

f. Save – Rename – Relocate the file to the site server:
…/code/sites/all/libraries/imagesloaded/**imagesloaded.pkgd.min.js**

g. Download and enable the Masonry API and Masonry Views modules – verify at the Status Report that the API plugin is properly detected.

Create a Standard Gallery [Beginner]

Required Modules: Views

The following steps will assist with creating a very basic gallery in Views to use right away, or as a starting point for additional sections later. There is no special customization other than building a standard Views to display each item for this section of the guide, and this, by default, does not adhere to responsive framework such as Bootstrap. Meaning – as the screen gets smaller, none of the images will stack.

Note: If not planning to develop further by using Masonry with filling in "white space" of a gallery, please consider to have a standard size of image widths/heights already determined. This will allow a uniform look across the images so that they do not appear off centered because one image is taller than the other.

1) After the content type has been created (and example content has been created), create a new View by going to **Structure** > **Views** > **Add new view**.

2) Provide a **View name**.

3) [OPTIONAL] Provide a **Description** (first check the **Description** box).

4) Ensure that **Content** is selected to show, from the appropriate content type selected. Leave the rest of this field at its default.

5) Check the **Create a page** option.
   Alternatively, if planning to insert this view elsewhere (such as Panels) this can be turned off, or a creation of a block can be done instead.

6) Provide a **Page Title**.

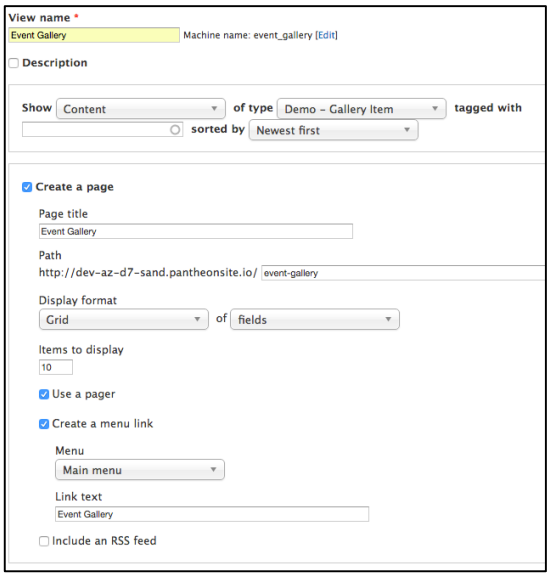
7) Under **Path**, it should auto generate the URL. If necessary, change the data provided.

8) Select **Grid** from the first drop down and **Fields** from the second of **Display Format**.

9) Leave the remaining options at its default value.

10) [OPTIONAL] If necessary, selecting **Create a menu link** will provide the settings to add a menu link automatically.

11) Select **Continue & Edit**.



*For this scenario, the following has been generated: "Event Gallery" is the view name; it is showing "content" of type "Gallery Item", with no additional settings; we will be creating a page with a title of "Event Gallery" that displays **Grid of fields**; uses a pager; and created a menu link.*

REMEMBER: AT ANY TIME, THESE SETTINGS CAN BE CHANGED AFTER THE VIEW HAS BEEN GENERATED IN THE VIEW CONFIGURATION ITSELF.

12) As appropriate, edit the **Display Name** to easily locate this view later.

13) Under **Fields**, select **Add**.

14) Locate and select the **Image field** appropriate to the content type.

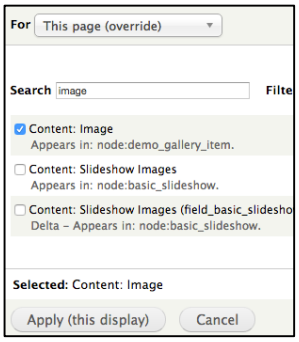
15) Under **For**…, select **This page (override)**.

16) Select **Apply (this display)**.

17) In the second window, unselect "**Create a label**".

18) Under **Image Style**, select **None (original image)**.

19) Set the **Link image to** as **Content**.



When adding a new field, we've selected "Content: Image", as it displays in our Gallery Item content type.

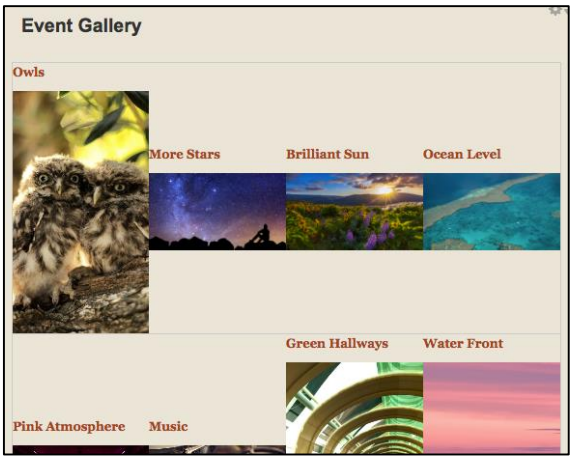
20) Leave the remaining settings as default.

21) Select **Apply (this display)**.

22) Rearrange the fields as necessary.

23) Click **Save** at the top of the Views configuration window to save and apply the view.

24) Verify the view works as intended, and adjust additional Views settings as needed (such as pager, menu, etc).



Update the Standard Gallery [Beginner/Intermediate]

Required Modules: Views, CSS Injector (unless creating/editing a CSS file)

The following steps will assist with making the basic gallery from the previous section more responsive and appealing to the web site visitor through the use of CSS. This guide assumes that knowledge and access to a CSS editing tool is available (such as CSS Injector module, or the direct CSS file), as well as that the responsive theming is developed on **Bootstrap**. It is recommended to use CSS Injector, as it is possible to define CSS changes to specific pages.

Note: If not planning to develop further by using Masonry with filling in "white space" of a gallery, please consider to have a standard size of image widths/heights already determined. This will allow a uniform look across the images so that they do not appear off centered because one image is taller than the other.

1) Create the basic view, and adjust standard settings as necessary.
   If this has not been created, please see the "Create a Standard Gallery" section.

2) Under the view **Format**, change the format to **Unformatted List**.

3) For *Unformatted List* settings, remove all checked boxes.

4) For **Row class**, enter the ideal *Bootstrap* column CSS for the different column behaviours. Example, *.col-xs-6* for mobile phone based layouts with 2 images across, or *.col-lg-4* to go three images on desktop. This field can support multiple classes at once.

   Note: In this field, CSS classes can be added with the leading period symbol or without. It will render the same.

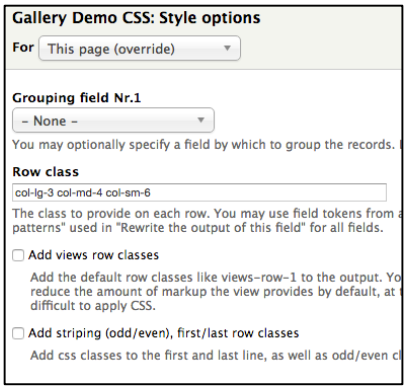
5) Select **Apple (this display)**.

6) Under **Fields**, only have an image based field and a title based field available and the image field should be above the title field.

7) Click on the image field to enter its settings window.

   a. Uncheck **Create a label**.

   b. Verify **None (original image)** is set as the *Image Style*.



In this scenario, we are using the following bootstrap CSS: col-lg-3 (large display, 4 per row), col-md-4 (medium displays, 3 per row), and col-sm-6 (small displays, 2 per row). By default, even though the extra small display is not defined, it will still apply one per row when the browser is smaller than 768px.

c. Verify that the *Link image to* is set to **Content**.

d. Click **Style Settings**.

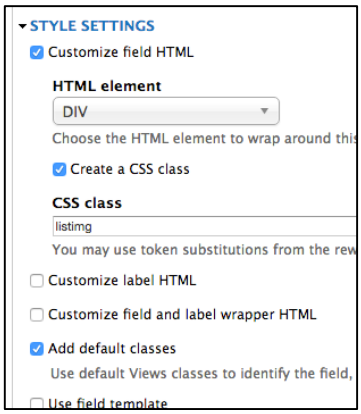
i. Check **Customize field HTML**.

ii. Under **HTML element**, select **DIV** from the list.

iii. Check **Create a CSS class**.

iv. Enter an appropriate CSS class name.
If necessary, this allows for additional CSS customization outside of the Views generated classes to help not accidently override other areas.

v. Leave the default settings.



*For this example, the CSS class to use will be **listimg***

e. Click **Apply (this display)**.

8) Click on the title field to enter its settings window.

a. Uncheck **Create a label**.

b. Verify that **Link this field to the original piece of content** is checked.

c. Click **Style Settings**.

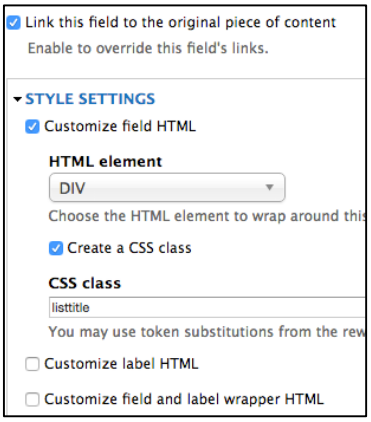
i. Check **Customize field HTML**.

ii. Under **HTML element**, select **DIV** from the list.

iii. Check **Create a CSS class**.

iv. Enter an appropriate CSS class name.
If necessary, this allows for additional CSS customization outside of the Views generated classes to help not accidently override other areas.

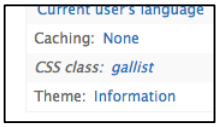
v. Leave the default settings.



*For this example, the CSS class to use will be **listtitle***

d. Click **Apply (this display)**.

9) Click the **Advanced** section on the most right of the Views window to expand additional settings.



*For this scenario, we will use **gallist***

10) Click **None** next to *CSS List*.

    a. Enter an appropriate CSS class name.
    If necessary, this allows for additional CSS customization outside of the Views generated classes to help not accidently override other areas.

    b. Leave the default settings.

11) Click **Save** at the top right corner of the Views settings window.

12) Go to the location necessary to edit the CSS. For the purpose of this guide, the **CSS Injector** module will be used and it is found under *Configuration* > *Development* > *CSS Injector*.

13) In **CSS Injector** – edit the following fields:

    a. **Friendly Name** – provide a unique ID name for the CSS rule. Ideally, attempt to create the name to help easily identify what it does.
    For the scenario – We will use "Gallery CSS".

    b. **Description** – Provide a simple description to describe what the code will accomplish.
    For the scenario – We will use "Gallery CSS for unformatted list".

    c. **Advanced Options** – If planning to have this apply to all themes, this area can be left at its default. Otherwise, adjust settings as appropriate.
    For the scenario – We will leave this area as its default settings.

    d. **Pages** – Assign specific pages if necessary that will use this CSS. Ideally, this is recommended if it is not planned to have the gallery on every page.
    *For the scenario – We will assign **The listed pages only** with the relative URL path the gallery is currently assigned to.*

14) Under **CSS code**, enter the necessary code to accomplish the desired outcome.

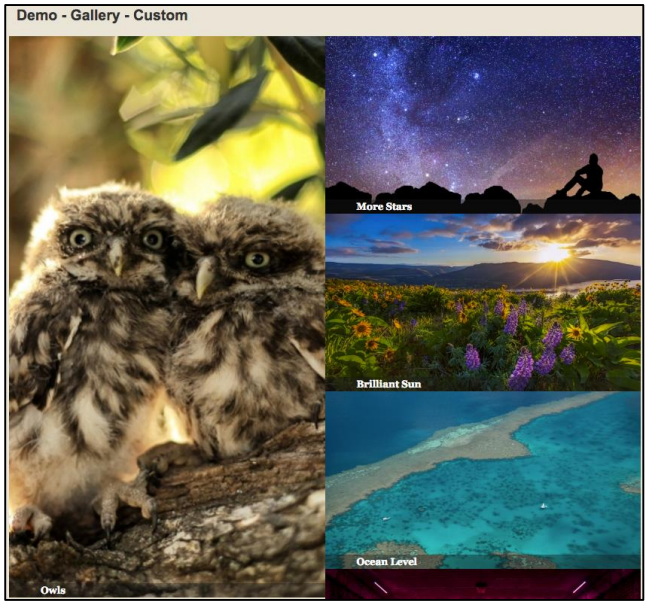
15) Click **Save** at the bottom to add the CSS.

FOR THE CURRENT SCENARIO, THE FOLLOWING CSS CODE WILL BE USED, AS WELL AS AN EXPLANATION TO WHAT IT IS ACCOMPLISHING. IF AT ANY TIME IT MAY BE NECESSARY TO INSPECT AND LOCATE ADDITIONAL VIEWS CLASSES, MOST WEB BROWSERS CAN INSPECT THE WEBPAGE AND PROVIDE YOU WITH THE CLASSES USED.

.gallist .views-field { margin-bottom: 0px !important; }	We are calling out the individual div groups that contain the image and iitle fields. By default, a bottom margin is applied between list items and we would like to remove this margin between the rows when we look at the gallery. We would like for this to apply only to this specific view in case we add future views later. To do this, we first call the CSS class that we applied to the entire view (.gallist) and then we add the standard/default views class that targets each list item (.views-field). From there, we are forcing the bottom margin to not have a size.
.gallist .listimg img { height: auto !important; width: 100% !important; }	The <img> tag, when used within the image field (.listimg) of the view that is being called (.gallist) will now force all images to fill the entire space it belongs to width wise. The height will automatically adjust itself as necessary. This allows images that are much taller to not show gaps while adjusting to the responsive layout if wider images are also surrounding the taller image. If all images are of the same height and width, this piece can be omitted.
.gallist .listtitle { position: absolute; bottom: 0; background: rgba(25, 25, 25, .4); width: 100%; }	When the title field (.listtitle) is showing in the specific view (.gallist), we want to force the DIV to align directly to the bottom of the picture that it relates to – this is completed by adding *bottom* to the CSS. The *position* property is telling the DIV to overlay the image field. The *background* and *width* property are adjusted to have the title field expand with the image and show a transparent background.
.gallist .listtitle a { color: #ffffff; margin-left: 10%; margin-bottom: 5%; }	In the title field, when a link is provided, we want the links to be white. As well, we are telling the links to be slightly offset so that they are not directly against the left or right edges. Instead of assigning pixels to the *margin* properties, we use percentages so that it can be responsive to the width of the area.

16) Preview and verify that the CSS has been applied to the appropriate View.

17) Adjust any settings if necessary.



Adding Masonry Effects to a Gallery [Intermediate]

Required Modules: Views, CSS Injector (unless creating/editing a CSS file), Masonry API, Masonry Views

The following steps will assist with making the basic gallery from the first section more responsive and appealing to the web site visitor through the use of CSS and Masonry API. This guide assumes that knowledge and access to a CSS editing tool is available (such as CSS Injector module, or the direct CSS file), as well as that the responsive theming is developed on **Bootstrap**. It is recommended to use CSS Injector, as it is possible to define CSS changes to specific pages.

Note: This method is typically ideal if planning to have multiple images of various sizes. The Masonry API is a JavaScript that will automatically figure out the image sizes and fit them in a puzzle-like format when the page loads.

1) Create the basic view, and adjust standard settings as necessary.
   If this has not been created, please see the "Create a Standard Gallery" section. Ideally, this guide will be picking up from the previous section, "Update the Standard Gallery", and the future changes will reflect from this setup.

2) Verify that the Masonry API and Masonry Views modules have been installed correctly.

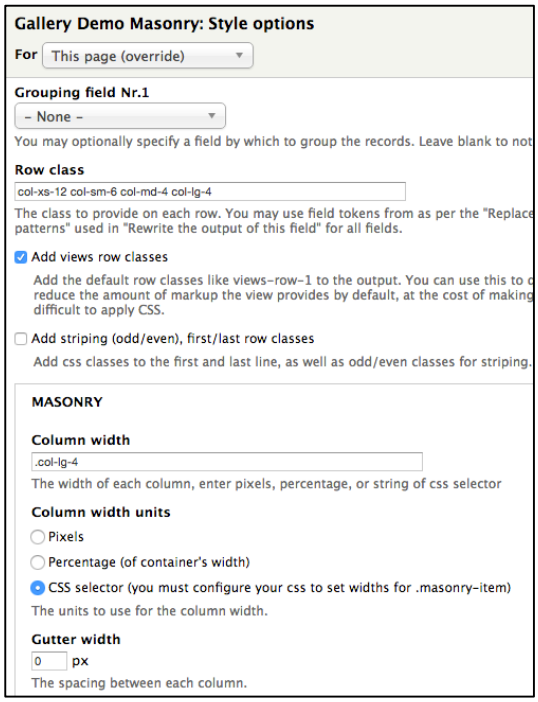
3) Under **Format**, change the format to **Masonry**. The settings should be as follows:

   a. For **Row class**, follow the same method as the previous section - enter the ideal *Bootstrap* column CSS for the different column behaviours. Example, *.col-xs-6* for mobile phone based layouts with 2 images across, or *.col-lg-4* to go three images on desktop. This field can support multiple classes at once.

   Note: In this field, CSS classes can be added with the leading period symbol or without. It will render the same

   b. Under **Masonry** section, set the **Column width** to either the medium or large *Bootstrap* column class used in the previous step. This provides a default of the column (each masonry item) to be one view-row wide of the view. The previous step will define the responsive setup.

   c. Ensure **CSS Selector** is selected.



*For this scenario, the Row classes will consist of **col-xs-12 col-sm-6 col-md-4 col-lg-4**. The Column width will be set to **col-lg-4**.*

d. **Gutter width** should be set to **0**.

e. Leave the remaining sections as their default values.

f. Select **Apply (this display)**.

4) Click **Save** at the top right corner of the Views settings window.

5) Go to the location necessary to edit the CSS. For the purpose of this guide, the **CSS Injector** module will be used and it is found under *Configuration* > *Development* > *CSS Injector*.

6) In **CSS Injector** – edit the following fields:

a. **Friendly Name** – provide a unique ID name for the CSS rule. Ideally, attempt to create the name to help easily identify what it does.
For the scenario – We will use "Gallery CSS Masonry".

b. **Description** – Provide a simple description to describe what the code will accomplish.
For the scenario – We will use "Gallery CSS for Masonry".

c. **Advanced Options** – If planning to have this apply to all themes, this area can be left at its default. Otherwise, adjust settings as appropriate.
For the scenario – We will leave this area as its default settings.

d. **Pages** – Assign specific pages if necessary that will use this CSS. Ideally, this is recommended if it is not planned to have the gallery on every page.
*For the scenario – We will assign **The listed pages only** with the relative URL path the gallery is currently assigned to.*

7) Under **CSS code**, enter the necessary code to accomplish the desired outcome.

8) Click **Save** at the bottom to add the CSS.

FOR THE CURRENT SCENARIO, THE FOLLOWING CSS CODE WILL BE USED, AS WELL AS AN EXPLANATION TO WHAT IT IS ACCOMPLISHING. IF AT ANY TIME IT MAY BE NECESSARY TO INSPECT AND LOCATE ADDITIONAL VIEWS CLASSES, MOST WEB BROWSERS CAN INSPECT THE WEBPAGE AND PROVIDE YOU WITH THE CLASSES USED. ALL PREVIOUS CSS CODE WILL BE USED FROM THE PREVIOUS SECTION OF THIS GUIDE. THE BELOW IS ADDITIONAL CODE USED.

.gallist .masonry-item { **padding: 0px !important;** **margin: 0px !important;** **border: 0px !important;** **}**	We are requesting that any Masonry item (.masonry-item) within the specific View/Gallery (.gallist) needs to have all borders and empty space between each item removed. By default, Masonry adds these properties. The idea for this gallery is to have each image attached to the other like a puzzle without space in between. We added !important so that it will override the Masonry settings.

9) Preview and verify that the CSS has been applied to the appropriate View.

10) Adjust any settings if necessary.