# Tutorial: How to Build a Custom Panel on Banana

*Source: https://github.com/lucidworks/banana/wiki/Tutorial:-How-to-Build-a-Custom-Panel*

In this tutorial, we are going to show you how to build a new custom panel in Banana dashboard.

- Copy this template directory to your banana/src/app/panels/ and rename it to reflect the functionality of our panel. For example, in this tutorial, we will create a custom panel that draw a bar chart using D3.js, so let's name this panel, bar, and rename the directory accordingly to banana/src/app/panels/bar/.
- Inside the bar panel directory, we should see three files:

1. editor.html - this is the view code for the panel's settings page.

```
<div class="row-fluid">
   <div class="span3">
      <label class="small">Field for displaying values <tip>Specify a field to
show values on the bar chart</tip></label>
      <input type="text" class="input-small" bs-typeahead="fields.list" ng-
model="panel.field" ng-change="set_refresh(true)">
   </div>
   <div class="span2">
      <label class="small">Number of Rows <tip>Specify a number of rows to
retrieve the result set.</tip></label>
      <input class="input-small" type="number" ng-model="panel.max_rows" ng-
change="set_refresh(true)">
  </div>
</div>
```

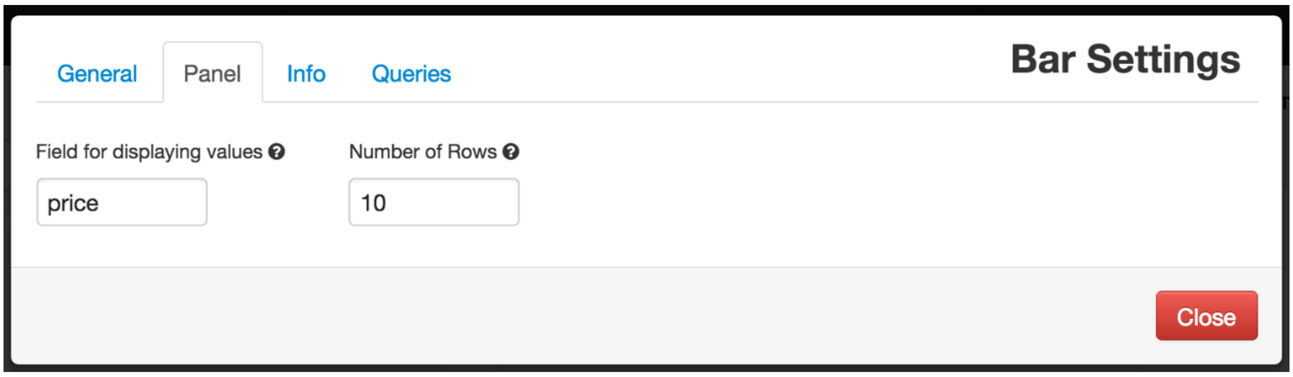
2. module.html - this is the view code (visual representation on the dashboard) of the panel.

```
<div ng-controller="bar" ng-init="init()">
   <style>
      bar-chart rect {
         fill: steelblue;
      }
      bar-chart text {
         fill: white;
         font: 10px sans-serif;
         text-anchor: end;
      }
   </style>
   <bar-chart></bar-chart>
</div>
```

3. module.js - this is where we define the panel's controller code.

```
/*
  ## Bar module
  * For tutorial on how to create a custom Banana module.
*/
define([
  'angular',
  'app',
  'underscore',
  'jquery',
  'd3'
],
function (angular, app, _, $, d3) {
  'use strict';

  var module = angular.module('kibana.panels.bar', []);
  app.useModule(module);

  module.controller('bar', function($scope, dashboard, querySrv, filterSrv) {
    $scope.panelMeta = {
      modals: [
        {
          description: 'Inspect',
          icon: 'icon-info-sign',
          partial: 'app/partials/inspector.html',
          show: $scope.panel.spyable
        }
      ],
      editorTabs: [
        {
          title: 'Queries',
          src: 'app/partials/querySelect.html'
        }
      ],
      status: 'Experimental',
      description: 'Bar module for tutorial'
    };

    // Define panel's default properties and values
    var _d = {
      queries: {
        mode: 'all',
        query: '*:*',
        custom: ''
      },
      field: '',
      max_rows: 10,
      spyable: true,
      show_queries: true
    };

    // Set panel's default values
    _.defaults($scope.panel, _d);

    $scope.init = function() {
      $scope.$on('refresh',function(){
        $scope.get_data();
      });
      $scope.get_data();
    };

    $scope.set_refresh = function(state) {
      $scope.refresh = state;
    };

    $scope.close_edit = function() {
      if ($scope.refresh) {
        $scope.get_data();
      }
      $scope.refresh = false;
      $scope.$emit('render');
    };

    $scope.render = function() {
      $scope.$emit('render');
    };

    $scope.get_data = function() {
      // Show the spinning wheel icon
      $scope.panelMeta.loading = true;

      // Set Solr server
      $scope.sjs.client.server(dashboard.current.solr.server +
dashboard.current.solr.core_name);
      var request = $scope.sjs.Request();

      // Construct Solr query
      var fq = '';
      if (filterSrv.getSolrFq()) {
          fq = '&' + filterSrv.getSolrFq();
      }
      var wt = '&wt=csv';
      var fl = '&fl=' + $scope.panel.field;
      var rows_limit = '&rows=' + $scope.panel.max_rows;

      $scope.panel.queries.query = querySrv.getQuery(0) + fq + fl + wt +
rows_limit;

      // Set the additional custom query
      if ($scope.panel.queries.custom != null) {
          request = request.setQuery($scope.panel.queries.query +
$scope.panel.queries.custom);
      } else {
          request = request.setQuery($scope.panel.queries.query);
      }

      // Execute the search and get results
      var results = request.doSearch();

      // Populate scope when we have results
      results.then(function(results) {
        $scope.data = {};

        var parsedResults = d3.csv.parse(results, function(d) {
          d[$scope.panel.field] = +d[$scope.panel.field]; // coerce to number
          return d;
        });

        $scope.data = _.pluck(parsedResults,$scope.panel.field);
        $scope.render();
      });

      // Hide the spinning wheel icon
      $scope.panelMeta.loading = false;
    };
  });
  module.directive('barChart', function() {
    return {
      restrict: 'E',
      link: function(scope, element) {
        scope.$on('render',function(){
          render_panel();
        });

        // Render the panel when resizing browser window
        angular.element(window).bind('resize', function() {
          render_panel();
        });

        // Function for rendering panel
        function render_panel() {
          // Clear the panel
          element.html('');

          var parent_width = element.parent().width(),
              height = parseInt(scope.row.height),
              width = parent_width - 20,
              barHeight = height / scope.data.length;

          var x = d3.scale.linear()
                    .domain([0, d3.max(scope.data)])
                    .range([0, width]);

          var chart = d3.select(element[0]).append('svg')
                        .attr('width', width)
                        .attr('height', height);

          var bar = chart.selectAll('g')
                      .data(scope.data)
                    .enter().append('g')
                      .attr('transform', function(d,i) {
                        return 'translate(0,' + i * barHeight + ")";
                      });

          bar.append('rect')
            .attr('width', x)
            .attr('height', barHeight - 1);

          bar.append('text')
            .attr('x', function(d) { return x(d) - 3; })
            .attr('y', barHeight / 2)
            .attr('dy', '.35em')
            .text(function(d) { return d; });
        }
      }
    };
  });
});
```

- Let's define the settings for the bar panel. To keep it simple, we will only define two settings. First setting is the field for displaying values on the bar chart. And second setting is the number of rows that we want to retrieve results from Solr.
- On the basis of the code of **editor.html** seen before, this is what the bar panel settings page looks like:

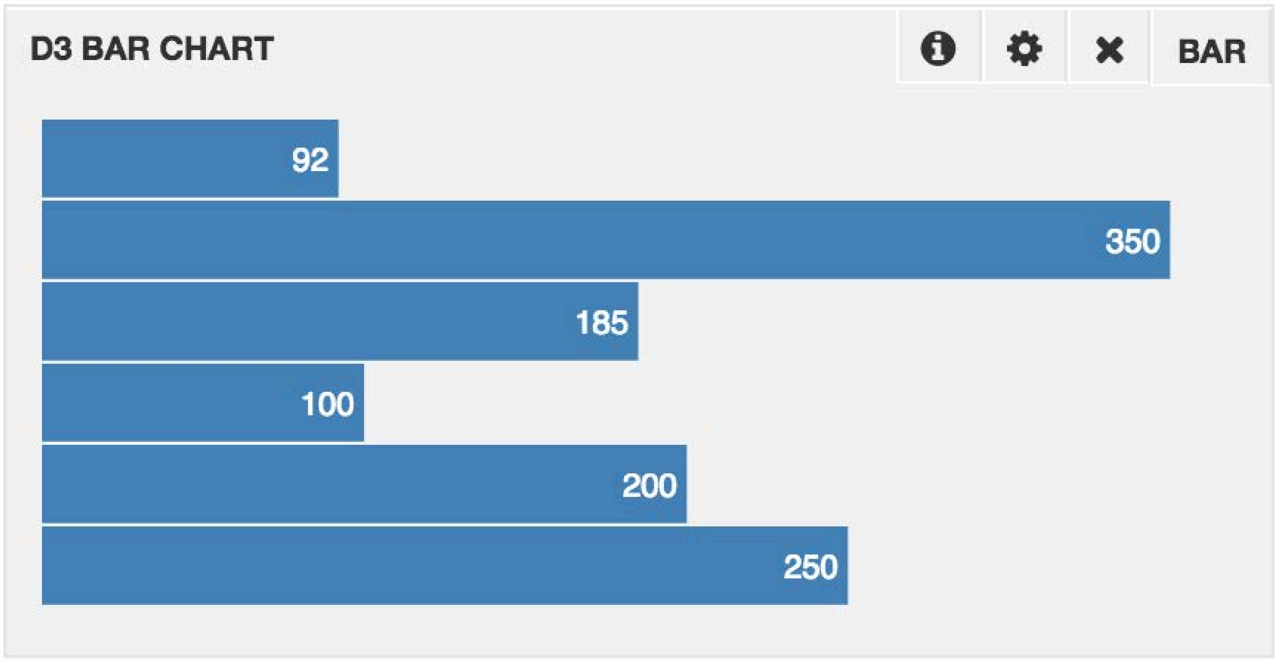


Bar Settings

Field for displaying values: price

Number of Rows: 10

- Next, let's edit ***module.html*** to display the bar chart on the dashboard. We can also include CSS code to customize the chart stylings. Here we will specify two stylings; one for 'rect' element to show the chart in steel blue color, and the other for 'text' element to show the text color on the chart in white.
- On the basis of the code of **module.html** seen before, this is what the bar panel settings page looks like on the dashboard:



- Note that `<bar-chart>` is an AngularJS directive that we will define in the next step. This directive is responsible for drawing our chart (the presentation layer).
- ***module.js*** is where our controller code lives. We will define the default settings of the panel, the business logics (e.g. sending requests to Solr and transforming responses from Solr), and the panel's directive for drawing the chart. Let's take a look at the complete code of module.js (copied before) and then we will go through each section to understand them.

- Since we will use D3.js to draw the chart, we need to inject it into our module at the beginning of ***module.js***:

```
define([
      'angular',
      'app',
      'underscore',
      'jquery',
      'd3'
  ],
  function (angular, app, _, $, d3) {
      ...
  });
```

- Then we name our module and controller `kibana.panels.bar` and `bar` respectively:

```
var module = angular.module('kibana.panels.bar', []);
  app.useModule(module);

  module.controller('bar', function($scope, dashboard, querySrv, filterSrv)
{...}
```

- Next, we define `$scope.panelMeta` which provides meta data about the panel. In general, we will only need to edit `status` and `description` of the panel. The list of possible statuses are: *Experimental, Beta, Stable, and Deprecated*.

```
$scope.panelMeta = {
      modals: [
          {
              description: 'Inspect',
              icon: 'icon-info-sign',
              partial: 'app/partials/inspector.html',
              show: $scope.panel.spyable
          }
      ],
      editorTabs: [
          {
              title: 'Queries',
              src: 'app/partials/querySelect.html'
          }
      ],
      status: 'Experimental',
      description: 'Bar module for tutorial'
  };
```

- Next, we define the panel's default properties and values. These settings can be accessed via `$scope.panel` object. In ***editor.html***, we expose two settings which are `$scope.panel.field` and `$scope.panel.max_rows`, so the user can configure them through the UI:

```
var _d = {
      queries: {
          mode: 'all',
          query: '*:*',
          custom: ''
      },
      field: '',
      max_rows: 10,
      spyable: true,
      show_queries: true
  };

  _.defaults($scope.panel, _d);
```

- `$scope.init()` function is for initializing the panel. Here, we also set the listener on `refresh` event. So whenever the dashboard got refreshed, `$scope.get_data()` will be called to get data from Solr and update the panel. Almost all Banana panels will have the same code for `$scope.init()`.

```
$scope.init = function() {
      $scope.$on('refresh',function(){
          $scope.get_data();
      });
      $scope.get_data();
  };
```

- `$scope.set_refresh(state)` function is used to trigger the refresh event by setting it to true like this `$scope.set_refresh(true)`:

```
  $scope.set_refresh = function(state) {
      $scope.refresh = state;
  };
```

- `$scope.close_edit()` function is called whenever we close the panel's settings page (***editor.html***):

```
  $scope.close_edit = function() {
      if ($scope.refresh) {
          $scope.get_data();
      }
      $scope.refresh = false;
      $scope.$emit('render');
  };
```

- `$scope.render()` function is used to send the render event so the panel will be redrawn:

```
  $scope.render = function() {
      $scope.$emit('render');
  };
```

- `$scope.get_data()` function is the main function where Solr requests will be sent and responses will be processed according to our logics and then rendered on the panel:

```
  $scope.get_data = function() {
      // Show the spinning wheel icon
      $scope.panelMeta.loading = true;

      // Set Solr server
      $scope.sjs.client.server(dashboard.current.solr.server +
dashboard.current.solr.core_name);
      var request = $scope.sjs.Request();

      // Construct Solr query
      var fq = '';
      if (filterSrv.getSolrFq() && filterSrv.getSolrFq() != '') {
          fq = '&' + filterSrv.getSolrFq();
      }
      var wt = '&wt=csv';
      var fl = '&fl=' + $scope.panel.field;
      var rows_limit = '&rows=' + $scope.panel.max_rows;

      $scope.panel.queries.query = querySrv.getQuery(0) + fq + fl + wt +
rows_limit;

      // Set the additional custom query
      if ($scope.panel.queries.custom != null) {
          request = request.setQuery($scope.panel.queries.query +
$scope.panel.queries.custom);
      } else {
          request = request.setQuery($scope.panel.queries.query);
      }

      // Execute the search and get results
      var results = request.doSearch();

      // Populate scope when we have results
      results.then(function(results) {
          $scope.data = {};

          var parsedResults = d3.csv.parse(results, function(d) {
              d[$scope.panel.field] = +d[$scope.panel.field]; // coerce to
number
              return d;
          });

          $scope.data = _.pluck(parsedResults,$scope.panel.field);
          $scope.render();
      });

      // Hide the spinning wheel icon
      $scope.panelMeta.loading = false;
  };
```

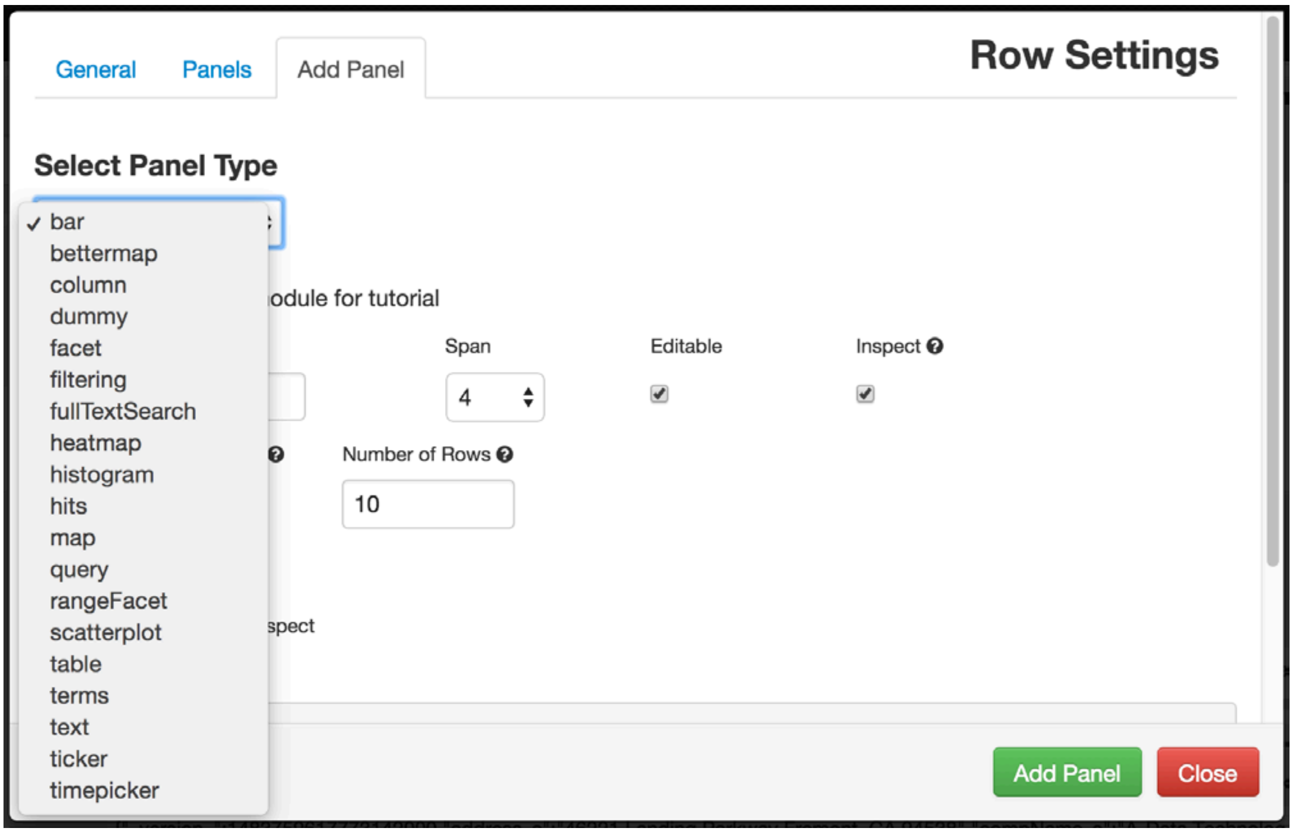
- Next, we will define our panel's directive for drawing the bar chart. Here we name our directive `barChart` so in ***module.html***, we have to call it `<bar-chart>` following the AngularJS rule. The main logic is in `render_panel()` function. In there, we setup our D3 bar chart's height and width. Then we use the parsed results from Solr stored in `scope.data` to draw the bar chart (inside directive, we do not need to prepend `$` to the scope variable when referencing it):

```
module.directive('barChart', function() {
      return {
          restrict: 'E',
          link: function(scope, element) {
              scope.$on('render',function(){
                  render_panel();
```

```
            });

            // Render the panel when resizing browser window
            angular.element(window).bind('resize', function() {
                render_panel();
            });

            // Function for rendering panel
            function render_panel() {
                // Clear the panel
                element.html('');

                var parent_width = element.parent().width(),
                    height = parseInt(scope.row.height),
                    width = parent_width - 20,
                    barHeight = height / scope.data.length;

                var x = d3.scale.linear()
                        .domain([0, d3.max(scope.data)])
                        .range([0, width]);

                var chart = d3.select(element[0]).append('svg')
                            .attr('width', width)
                            .attr('height', height)

                var bar = chart.selectAll('g')
                            .data(scope.data)
                          .enter().append('g')
                            .attr('transform', function(d,i) {
                                return 'translate(0,' + i * barHeight + ")";
                            });

                bar.append('rect')
                    .attr('width', x)
                    .attr('height', barHeight - 1);

                bar.append('text')
                    .attr('x', function(d) { return x(d) - 3; })
                    .attr('y', barHeight / 2)
                    .attr('dy', '.35em')
                    .text(function(d) { return d; });
            }
        }
    };
});
```

- Add the new panel name to panel_names array in banana/src/config.js, so that it will show up in the drop-down box in the "Add Panel" interface.

```
panel_names: [
    'bar',
    ...
]
```



- That's it! Now you should be able to add the bar panel to the dashboard after refreshing your browser. If the panel does not show up, you may need to restart your web server that hosting Banana or clearing your browser's cache.