NetCoreControls Documentation

Release 0.1.0

Joao Pereira

Mar 25, 2019

Get started

A set of UI controls for ASP.NET Core.

GitHub repository

Constrols demonstration

CHAPTER 1

Features

- Independent from data source

You can use any data source you prefer. Just set up a method that returns the data you want to display.

- Dynamic models allowed

There is no need to create a model to render data to a control. Just return `dynamic` from your data method.

- AJAX Enabled

All controls use AJAX to communicate with the server and perform their actions.

- Controls are connected

You can easily associate a submit button or a filter with more than one control, even with different controls.

- Subscribe control events or create custom ones

All controls share the same base events. They also offer some other events related to the control itself. But hey!, if that isn't enough, you can create your one custom events!

1.1 Setup and Overview

You can use these controls with any web ASP.NET Core project. All controls were built natively for .NET Core and use Tag Helpers to perform all their logic.

Note: NetCoreControls only targets ASP.NET Core 1.1. If you have an ASP.NET Core 1.0 project then you can follow this guide for updating to ASP.NET Core 1.1.

1.1.1 Dependencies

You must use the **jQuery javascript** library starting from v2.x.

The controls also use some styles from Bootstrap, but it's not a mandatory requirement since you can link your own styles classes.

1.1.2 Basic setup

1. Install the NetCoreControls NuGet package

Add to `project.json` the following dependency:

```
"NetCoreControls" : "1.0.0-beta1"
```

Or you can use the Package Manager Console:

```
Install-Package NetCoreControls -Pre
```

2. Register NetCoreControls

In your `Startup.cs` class, inside the `ConfigureServices` method, add the following line after Mvc registration:

```
services.AddMvc();
(...)
services.AddNetCoreControls(Configuration);
```

3. Reference the assembly to enable usage as TagHelpers

In your `_ViewImports.cshtml` file inside your `Views` folder, add the following line:

```
@addTagHelper "*, NetCoreControls"
```

4. Add references to CSS and Script files

Inside your `<head></head>` tag, insert the following:

```
<link href="@Url.Action("GetNccCssFile", "NetCoreControls")" rel="stylesheet">
```

On the bottom of your page, just above the `</body>` tag, insert the following:

```
<script type="text/javascript" src="@Url.Action("GetNccJsFile", "NetCoreControls")"></
↪script>
```

Note: Although the tag that links to the stylesheet is optional, the script is mandatory and should be placed after the jQuery link.

Daily builds

To use the latest daily builds of the controls, please add the following MyGet repo to download latest binaries:

```
https://www.myget.org/F/netcorecontrols/api/v3/index.json
```

Add to `project.json` the following dependency:

```
"NetCoreControls" : "1.0.0-beta-*"
```

1.2 Basic Usage

Just use any of the available controls using the corresponding taghelper tag.

All tags are prefixed with `ncc:`.

All taghelpers that are attributes are prefixed with `ncc-`.

1.2.1 Basic control usage

To use any control, two steps are required:

- Create a control context
- Use the tag of the control

The example of a basic Grid is as follows.

1. Create a method that gets the data from a datasource such as a database (the example uses Dapper for data access):

```
public List<dynamic> GetProductList()
{
  var sqlCommand = $@" SELECT * FROM Products";
  return Task.Factory.StartNew(() =>
  {
    using (var connection = new SqlConnection(_connStrings.Value.LocalDb))
      return connection.Query<dynamic>(sqlCommand);
    }).Result.ToList();
  }
}
```

2. On your Controller define a NccContext (e.g. ‘‘NccGridContext‘‘) object and set required parameters. Pass it to the View using ‘‘Model‘‘, ‘‘ViewBag‘‘, ‘‘ViewData‘‘ or any other method

```
using ByteNuts.NetCoreControls.Models.Grid;
...
var context = new NccGridContext
{
  Id = "SimpleGrid",
  DataAccessClass = typeof(IDataAccess).AssemblyQualifiedName,
  SelectMethod = "GetProductList",
  UseDependencyInjection = true,
  ViewPaths = new ViewsPathsModel { ViewPath = "/Views/NccGrid/SimpleGrid.cshtml"}
};
ViewData[context.Id] = context;
```

3. On your View simply use the tag helper ‘‘ncc:grid‘‘, along with the available nested tags. Set the grid context, and use the @Model inside the control tags to access all the properties available in each list item

```
@using ByteNuts.NetCoreControls.Models.Grid
@{
  var context = ViewData["SimpleGrid"] as NccGridContext;
}
<ncc:grid Context="@context">
  <ncc:grid-columns>
    <ncc:grid-columnbound DataValue="@Model.ProductID" HeaderText="Product Id"></
↪ncc:grid-columnbound>
```

(continues on next page)

(continued from previous page)

```
    <ncc:grid-columnbound DataValue="@Model.ProductName" HeaderText="Product Name"></
↪ncc:grid-columnbound>
    <ncc:grid-columnbound DataValue="@Model.SupplierID" HeaderText="Supplier ID"></
↪ncc:grid-columnbound>
    <ncc:grid-columnbound DataValue="@Model.CategoryID" HeaderText="Category ID"></
↪ncc:grid-columnbound>
    <ncc:grid-columnbound DataValue="@Model.QuantityPerUnit" HeaderText="Quantity Per␣
↪Unit"></ncc:grid-columnbound>
    <ncc:grid-columnbound DataValue="@($"{Model.UnitPrice:0.00} €")" HeaderText="Unit␣
↪Price"></ncc:grid-columnbound>
    <ncc:grid-columnbound DataValue="@Model.UnitsInStock" HeaderText="Units In Stock">
↪</ncc:grid-columnbound>
    <ncc:grid-columnbound DataValue="@Model.UnitsOnOrder" HeaderText="Units On Order">
↪</ncc:grid-columnbound>
    <ncc:grid-columnbound DataValue="@Model.ReorderLevel" HeaderText="Reorder Level">
↪</ncc:grid-columnbound>
    <ncc:grid-columnbound DataValue="@(Model.Discontinued ? "Discontinued" : "Active")
↪" HeaderText="Discontinued"></ncc:grid-columnbound>
  </ncc:grid-columns>
</ncc:grid>
```

Warning: The grid shall be placed alone in a partial view. When an action occurs on the control, the partial view is full rendered.

1.2.2 Filters

Filters are an attribute taghelper and are global to all controls.

One single filter can be applied to multiple controls.

Filters can be applied to `input`, `select` and `button` html tags, just using the `ncc-filter-targets` as follow:

```
<select ncc-filter-targets="MultiGridWithFilter1,MultiGridWithFilter2" name="orderId"␣
↪asp-items="@(new SelectList(ViewData["Orders"] as IEnumerable, "OrderID", "OrderID
↪"))" >
  <option value="">--- Choose an order ---</option>
</select>
```

1.2.3 Events

To be able to use events, the class containing these events must inherit from the control events class.

Beside inheriting, it must be referenced in the control context that is created, using the property `EventHandlerClass`:

e.g. `EventHandlerClass = typeof(ExampleGridEvents).AssemblyQualifiedName`

The event handler class must inherit from one of the following two:

- **ByteNuts.NetCoreControls.Controls.NccEvents** –> this class defines the shared events;
- **ByteNuts.NetCoreControls.Controls.[ControlName].Events.[ControlName]Events** –> this class defines control specific events.

Hint: If you inherit from the base events class NccEvents, you may only subscribe to control shared events and not to control specific events.

1.3 Advanced Setup

There are some more additional settings that can be used to setup controls.

1.3.1 Settings

This settings are global to the controls, and are placed on the `appsettings.json` file, within a section named `NccSettings`:

```
{
    (...)
    "Logging": {
        (...)
    },
    "NccSettings": {
        "UseDependencyInjection": "true",
        "DataProtectionKey": "11111111-2222-3333-4444-555555555555"
    }
}
```

- **UseDependencyInjection** *(default: true)* - indicates wether the data access class should be requested directly from the iOC or if it should be instantiated.
- **DataProtectionKey** - defines the key to be used when encrypting the context of each control. The example uses a Guid, but it can be any type of string.

1.3.2 Exception Handling

Exceptions within tag helpers are contained and isolated from the page, preventing that an error in a control blocks the rendering of a page.

Neverthless, if the error occurs on the Razor code, such as trying to read an inexisting property, these errors cannot be handled by the NccControls and prevent the page from rendering.

To prevent this from happening, you can use the `RenderControl` TagHelper to render the controls. So, instead of using:

```
@await Html.PartialAsync("/Views/NccGrid/Partials/_GridExample.cshtml")
```

just use:

```
<ncc:render-control Context="@(ViewData["GridExample"] as NccGridContext)"></
↪ncc:render-control>
```

1.4 Shared Base

All controls share the same base.

This section highlights what belongs to the base and can be used within all the controls.

1.4.1 Context

The base context class is `NccContext`.

- **Id** - this is the id of the control. It must be unique in the page, and it's used to reference the control within any filter or other event.
- **RenderForm** *(default: true)* - a flag indicating if the control can render the form automatically or if the user wants to take control of it. A form is not mandatory for simple control interacting, such as filters and paging, but it's mandatory when actions such as update are used.
- **AutoBind** *(default: true)* - indicates if the control shall read and bind the data from the data access method, or if the data is passed in the `DataSource` property and it should always use that data to perform all actions.
- **DataSource** - if the data is only read once, it can be passed in this property. `AutoBind` property must be set to false, so this data don't get override.
- **Visible** *(default: true)* - indicates wether a control is visible on page or not. The control may exist on the page so filters can be applied to him, but it may be invisble unless the filter has a specific value.
- **DataAccessClass** - the class to request or instantiate where the data access methods exist.
- **DataAccessParameters** - parameters required by the constructor of the `DataAccessClass`. Only required when the constructor has parameters AND the global setting `UseDependencyInjection` is set to false.
- **SelectMethod** - the method used to select the data.
- **SelectParameters** - the parameters required by the `SelectMethod`. Can be static parameters or filters that can be override later.
- **UpdateMethod** - the method used to update the data.
- **UpdateParameters** - the parameters required by the `UpdateMethod`. The POCO model used to pass the data form the control is not required to be indicated here.
- **DatabaseModelType** - the Type of the POCO model that maps the data and is used by the `UpdateMethod`.
- **EventHandlerClass** - the class where the custom events are defined for the control.
- **Filters** - a `Dictionary<string, string>` that contain the filters applied to the `SelectMethod`. Normally this property is set by the filter attribute later.
- **AdditionalData** - a `Dictionary<string, object>` that can be used to store data that is used inside the control to render data such as options for a Select tag.
- **ViewPaths** - a `ViewsPathsModel` object that contain the paths to the views. At the moment, there is only a `ViewPath` property that must be set with the control partial path.

1.4.2 Filters

Filters are global and can be set to any control existing on page, from any HTML element such as `input`, `select`, `button`.

To use the filter, an attribute must be placed in the HTML element.

- **ncc-filter-targets** - allows to indicate the id's of the controls to whom the filter applies. Multiple controls may be set, with comma separated id's.
- **ncc-filter-ids** - the id(s) of the HTML element that contain the values of the filter. This option MUST exist if the filter attribute is placed in a `button` element.

- **ncc-js-events** - the javascript event that submits the filter. By default, `onchange` is used by Select, `onkeyup` is used by Input and `onclick` on Button.

Note: It is possible to set multiple inputs with different filters, and allow a button to submit all filters at once, using the `ncc-filter-ids` attribute.

1.4.3 Events

Events are pre-defined within the base and are also specific to controls.

Events that are common to all controls are:

- **Load**
- **DataBound**
- **PreRender**
- **PostBack**

To use an event, just add the following 2 attributes to any element:

- **ncc-event** - the name of the event to raise
- **ncc-event-target** - the id(s) of the controls that this event applies.

Note: It is possible to use these events to raise events that are specific to a control, that are not mentioned here.

1.5 NCC Grid

A grid control that renders data as a table.

1.5.1 Context

The context class for the Grid control is `NccGridContext`.

- **DataKeys** - a comma separated data keys for the data. These keys are not rendered on the client, and cannot be overriden by hidden fields.
- **DataKeysValues** *(read-only)* - a read-only `List<Dictionary<string, object>>` that contains all the data keys for the grid.
- **PageNumber** *(default: 1)* - page number to start from.
- **TotalItems** *(read-only)* - a read-only items counter.
- **AllowPaging** *(default: false)* - a flag indicating if the grid is paginated.
- **PageSize** *(default: 10)* - if `AllowPaging` is set to true, defines the size of a grid page.
- **PagerNavSize** *(default: 10)* - if `AllowPaging` is set to true, defines how many pages are shown in the navigation pager.
- **AutoGenerateEditButton** *(default:false)* - a flag indicating if the table is editable row by row.

1.5.2 Tags

The Grid control is composed by various tags that can be coupled together.

Each one of them contain custom attributes that can be set and some may override the settings placed on context.

<ncc-grid>

Attributes

- Context
- DataKeys
- AllowPaging
- RenderForm
- PageSize
- AutoGenerateEditButton
- PagerNavSize
- CssClass
- BodyCssClass
- HeaderCssClass
- FooterCssClass

Allowed child tags - ncc:grid-content - ncc:grid-columns

<ncc:grid-content>

Attributes

- ContentType

Allowed parent tags

- ncc:grid

<ncc:grid-columns>

Allowed parent tags

- ncc:grid

Allowed child tags

- ncc:grid-columnbound
- ncc:grid-columntemplate

<ncc:grid-columnbound>

Attributes

- DataValue
- DataField
- HeaderText
- ShowHeader
- Visible
- Aggregate
- CssClass

Allowed parent tags

- ncc:grid

<ncc:grid-columntemplate>

Attributes

- ShowHeader
- Visible

Allowed parent tags

- ncc:grid-columns

Allowed child tags

- ncc:grid-headertemplate
- ncc:grid-itemtemplate
- ncc:grid-edittemplate

<ncc:grid-headertemplate>

Attributes

- CssClass

Allowed parent tags

- ncc:grid-columntemplate

<ncc:grid-itemtemplate>

Attributes

- CssClass
- Aggregate

Allowed parent tags

- ncc:grid-columntemplate

<ncc:grid-edittemplate>

Attributes

- CssClass
- Aggregate

Allowed parent tags

- ncc:grid-columntemplate

1.5.3 Events

For subscribing to these events, the `EventHandlerClass` property of the context must be set with the reference for a class that derives from `NccGridEvents`.

The following are Grid specific events:

- **RowDataBound**
- **RowCreated**
- **Update**
- **UpdateRow**
- **DeleteRow**

1.5.4 Actions

Actions allows the user to raise any of the referred grid events, and can be associated with any HTML element.

To use an action, the following two first attributes must be set:

- **ncc-grid-action** - the name of the action to raise.
- **ncc-grid-action-target** - the id(s) of the controls that will raise the event (it can raise Update simultaneously on multiple grids).
- **ncc-grid-row** *(optional)* - if the action requires the row number which will raise the event, this attribute must be set.

Note: Within the Grid, you can use `@Model.NccRowNumber` property to insert the row number.

1.6 NCC Select

A select HTML tag that allows linking other controls and creates dependencies among the data loaded by each control.

1.6.1 Context

The context class for the Grid control is `NccSelectContext`.

- **TextValue** - a string name of a property that will be rendered as the text of the option.
- **DataValue** - a string name of a property that will be rendered as the value of the option.

- **SelectedValue** - the default value to be automatic selected.
- **FirstItem** - the value that the first element must contain. The value for this item is always an empty string.

1.6.2 Tags

The Select control is composed by a single tag.

Each custom attributes set on the tag will override the settings placed on context.

<ncc-select>

Attributes

- Context
- TextValue
- DataValue
- SelectedValue
- FirstItem

1.6.3 Events

For subscribing to these events, the `EventHandlerClass` property of the context must be set with the reference for a class that derives from `NccSelectEvents`.

The following are Select specific events:

- **OptionBound** - fires on each option before added to select

1.6.4 Link control

To link this select with other controls on page, whether they are ncc:select controls, or any other NetCoreControls, their id's must be included in a specific attribute tag:

- **ncc-link-targets** - a comma separated and ordenated list of the controls that load their data according to the previous control selected.

Note: At the moment, **ncc-link-targets** can only be placed on NccSelect controls. Nevertheless, they can contain NccGrid id's for example, but any action taken on the grid will not have any efect on the NccSelect controls. Please, see samples for a working example.