<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/view-components?view=aspnetcore-2.1>

### The view component class

A view component class can be created by any of the following:

* Deriving from ViewComponent
* Decorating a class with the [ViewComponent] attribute, or deriving from a class with the [ViewComponent] attribute
* Creating a class where the name ends with the suffix ViewComponent

Like controllers, view components must be public, non-nested, and non-abstract classes. The view component name is the class name with the "ViewComponent" suffix removed. It can also be explicitly specified using the ViewComponentAttribute.Name property.

A view component class:

* Fully supports constructor [dependency injection](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-2.1)
* Doesn't take part in the controller lifecycle, which means you can't use [filters](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-2.1) in a view component

## Invoking a view component

To use the view component, call the following inside a view:

CSHTML Copy

@Component.InvokeAsync("Name of view component", <anonymous type containing parameters>)

The parameters will be passed to the InvokeAsync method. The PriorityList view component developed in the article is invoked from the Views/Todo/Index.cshtml view file. In the following, the InvokeAsync method is called with two parameters:

CSHTML Copy

@await Component.InvokeAsync("PriorityList", new { maxPriority = 4, isDone = true })

## Invoking a view component as a Tag Helper

For ASP.NET Core 1.1 and higher, you can invoke a view component as a [Tag Helper](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-2.1):

CSHTML Copy

<vc:priority-list max-priority="2" is-done="false">

</vc:priority-list>

Pascal-cased class and method parameters for Tag Helpers are translated into their [lower kebab case](https://stackoverflow.com/questions/11273282/whats-the-name-for-dash-separated-case/12273101). The Tag Helper to invoke a view component uses the <vc></vc> element. The view component is specified as follows:

CSHTML Copy

<vc:[view-component-name]

parameter1="parameter1 value"

parameter2="parameter2 value">

</vc:[view-component-name]>

Note: In order to use a View Component as a Tag Helper, you must register the assembly containing the View Component using the @addTagHelper directive. For example, if your View Component is in an assembly called "MyWebApp", add the following directive to the \_ViewImports.cshtml file:

CSHTML Copy

@addTagHelper \*, MyWebApp

You can register a View Component as a Tag Helper to any file that references the View Component. See [Managing Tag Helper Scope](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-2.1#managing-tag-helper-scope) for more information on how to register Tag Helpers.

The InvokeAsync method used in this tutorial:

CSHTML Copy

@await Component.InvokeAsync("PriorityList", new { maxPriority = 4, isDone = true })

In Tag Helper markup:

CSHTML Copy

<vc:priority-list max-priority="2" is-done="false">

</vc:priority-list>

In the sample above, the PriorityList view component becomes priority-list. The parameters to the view component are passed as attributes in lower kebab case.

### Invoking a view component directly from a controller

View components are typically invoked from a view, but you can invoke them directly from a controller method. While view components don't define endpoints like controllers, you can easily implement a controller action that returns the content of a ViewComponentResult.

In this example, the view component is called directly from the controller:

C# Copy

public IActionResult IndexVC()

{

return ViewComponent("PriorityList", new { maxPriority = 3, isDone = false });

}

## Walkthrough: Creating a simple view component

[Download](https://github.com/aspnet/Docs/tree/master/aspnetcore/mvc/views/view-components/sample), build and test the starter code. It's a simple project with a Todo controller that displays a list of Todo items.

### Add a ViewComponent class

Create a ViewComponents folder and add the following PriorityListViewComponent class:

C# Copy

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using ViewComponentSample.Models;

namespace ViewComponentSample.ViewComponents

{

public class PriorityListViewComponent : ViewComponent

{

private readonly ToDoContext db;

public PriorityListViewComponent(ToDoContext context)

{

db = context;

}

public async Task<IViewComponentResult> InvokeAsync(

int maxPriority, bool isDone)

{

var items = await GetItemsAsync(maxPriority, isDone);

return View(items);

}

private Task<List<TodoItem>> GetItemsAsync(int maxPriority, bool isDone)

{

return db.ToDo.Where(x => x.IsDone == isDone &&

x.Priority <= maxPriority).ToListAsync();

}

}

}

Notes on the code:

* View component classes can be contained in **any** folder in the project.
* Because the class name PriorityList**ViewComponent** ends with the suffix **ViewComponent**, the runtime will use the string "PriorityList" when referencing the class component from a view. I'll explain that in more detail later.
* The [ViewComponent] attribute can change the name used to reference a view component. For example, we could've named the class XYZ and applied the ViewComponent attribute:

C# Copy

[ViewComponent(Name = "PriorityList")]

public class XYZ : ViewComponent

* The [ViewComponent] attribute above tells the view component selector to use the name PriorityList when looking for the views associated with the component, and to use the string "PriorityList" when referencing the class component from a view. I'll explain that in more detail later.
* The component uses [dependency injection](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-2.1) to make the data context available.
* InvokeAsync exposes a method which can be called from a view, and it can take an arbitrary number of arguments.
* The InvokeAsync method returns the set of ToDo items that satisfy the isDone and maxPriority parameters.

### Create the view component Razor view

* Create the Views/Shared/Components folder. This folder **must** be named Components.
* Create the Views/Shared/Components/PriorityList folder. This folder name must match the name of the view component class, or the name of the class minus the suffix (if we followed convention and used the ViewComponent suffix in the class name). If you used the ViewComponent attribute, the class name would need to match the attribute designation.
* Create a Views/Shared/Components/PriorityList/Default.cshtml Razor view:

CSHTML Copy

@model IEnumerable<ViewComponentSample.Models.TodoItem>

<h3>Priority Items</h3>

<ul>

@foreach (var todo in Model)

{

<li>@todo.Name</li>

}

</ul>

The Razor view takes a list of TodoItem and displays them. If the view component InvokeAsync method doesn't pass the name of the view (as in our sample), Default is used for the view name by convention. Later in the tutorial, I'll show you how to pass the name of the view. To override the default styling for a specific controller, add a view to the controller-specific view folder (for example Views/Todo/Components/PriorityList/Default.cshtml).

If the view component is controller-specific, you can add it to the controller-specific folder (Views/Todo/Components/PriorityList/Default.cshtml).

* Add a div containing a call to the priority list component to the bottom of the Views/Todo/index.cshtml file:

CSHTML Copy

</table>

<div>

@await Component.InvokeAsync("PriorityList", new { maxPriority = 2, isDone = false })

</div>

The markup @await Component.InvokeAsync shows the syntax for calling view components. The first argument is the name of the component we want to invoke or call. Subsequent parameters are passed to the component. InvokeAsync can take an arbitrary number of arguments.

Test the app. The following image shows the ToDo list and the priority items:

You can also call the view component directly from the controller:

C# Copy

public IActionResult IndexVC()

{

return ViewComponent("PriorityList", new { maxPriority = 3, isDone = false });

}

### Specifying a view name

A complex view component might need to specify a non-default view under some conditions. The following code shows how to specify the "PVC" view from the InvokeAsync method. Update the InvokeAsync method in the PriorityListViewComponent class.

C# Copy

public async Task<IViewComponentResult> InvokeAsync(

int maxPriority, bool isDone)

{

string MyView = "Default";

// If asking for all completed tasks, render with the "PVC" view.

if (maxPriority > 3 && isDone == true)

{

MyView = "PVC";

}

var items = await GetItemsAsync(maxPriority, isDone);

return View(MyView, items);

}

Copy the Views/Shared/Components/PriorityList/Default.cshtml file to a view named Views/Shared/Components/PriorityList/PVC.cshtml. Add a heading to indicate the PVC view is being used.

CSHTML Copy

@model IEnumerable<ViewComponentSample.Models.TodoItem>

<h2> PVC Named Priority Component View</h2>

<h4>@ViewBag.PriorityMessage</h4>

<ul>

@foreach (var todo in Model)

{

<li>@todo.Name</li>

}

</ul>

Update Views/TodoList/Index.cshtml:

CSHTML Copy

@await Component.InvokeAsync("PriorityList", new { maxPriority = 4, isDone = true })

Run the app and verify PVC view.

If the PVC view isn't rendered, verify you are calling the view component with a priority of 4 or higher.

### Examine the view path

* Change the priority parameter to three or less so the priority view isn't returned.
* Temporarily rename the Views/Todo/Components/PriorityList/Default.cshtml to 1Default.cshtml.
* Test the app, you'll get the following error:

Copy

An unhandled exception occurred while processing the request.

InvalidOperationException: The view 'Components/PriorityList/Default' wasn't found. The following locations were searched:

/Views/ToDo/Components/PriorityList/Default.cshtml

/Views/Shared/Components/PriorityList/Default.cshtml

EnsureSuccessful

* Copy Views/Todo/Components/PriorityList/1Default.cshtml to Views/Shared/Components/PriorityList/Default.cshtml.
* Add some markup to the Shared Todo view component view to indicate the view is from the Shared folder.
* Test the **Shared** component view.

### Avoiding magic strings

If you want compile time safety, you can replace the hard-coded view component name with the class name. Create the view component without the "ViewComponent" suffix:

C# Copy

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using ViewComponentSample.Models;

namespace ViewComponentSample.ViewComponents

{

public class PriorityList : ViewComponent

{

private readonly ToDoContext db;

public PriorityList(ToDoContext context)

{

db = context;

}

public async Task<IViewComponentResult> InvokeAsync(

int maxPriority, bool isDone)

{

var items = await GetItemsAsync(maxPriority, isDone);

return View(items);

}

private Task<List<TodoItem>> GetItemsAsync(int maxPriority, bool isDone)

{

return db.ToDo.Where(x => x.IsDone == isDone &&

x.Priority <= maxPriority).ToListAsync();

}

}

}

Add a using statement to your Razor view file, and use the nameof operator:

CSHTML Copy

@using ViewComponentSample.Models

@using ViewComponentSample.ViewComponents

@model IEnumerable<TodoItem>

<h2>ToDo nameof</h2>

<!-- Markup removed for brevity. -->

<div>

@await Component.InvokeAsync(nameof(PriorityList), new { maxPriority = 4, isDone = true })

</div>

## Additional resources

* [Dependency injection into views](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/dependency-injection?view=aspnetcore-2.1)

Note

The feedback system for this content will be changing soon. Old comments will not be carried over. If content within a comment thread is important to you, please save a copy. For more information on the upcoming change, [we invite you to read our blog post](https://docs.microsoft.com/teamblog/a-new-feedback-system-is-coming-to-docs).