Arguably the simplest kind of action result that is defined by ASP.NET Core MVC is the collection of Status Code Results. These results merely return an HTTP status code to the client.

OkResult

The `OkResult` (short method: `Ok()`) return the *200 OK* status code.

```
public IActionResult OkResult()
{
    return Ok();
}
```

CreatedResult

The `CreatedResult` (short method: `Created()`) returns *201 Created* with a URL to the created resource.

```
public IActionResult CreatedResult()
{
    return Created("http://example.org/myitem", new { name = "testitem" });
}
```

NoContentResult

The `NoContentResult` (short method: `NoContent()`) returns a *204 No Content* status code, indicating that the server successfully processed the request, but that there is nothing to return.

```
public IActionResult NoContentResult()
{
    return NoContent();
}
```

BadRequestResult

The `BadRequestResult` (short method: `BadRequest()`) return *400 Bad Request*, which indicates that the server cannot process the request due to an error in said request. This is often used in APIs when validation of the request fails (and there isn't a more specific code that would fit better).

```
public IActionResult BadRequestResult()
{
    return BadRequest();
```

```
}
```

UnauthorizedResult

The `UnauthorizedResult` (short method: `Unauthorized()`) returns *401 Unauthorized*, indicating that the request cannot be processed because the user making the request doesn't have the appropriate authentication to do so (meaning this status code should really have been called *401 Unauthenticated*).

```
public IActionResult UnauthorizedResult()
{
    return Unauthorized();
}
```

NotFoundResult

The `NotFoundResult` (short method: `NotFound()`) returns the *404 Not Found* status code, indicating that the requested resource, for whatever reason, was not found on the server.

```
public IActionResult NotFoundResult()
{
    return NotFound();
}
```

UnsupportedMediaTypeResult

The `UnsupportedMediaTypeResult`, which doesn't have a short method at the time of writing, returns *415 Unsupported Media Type*, indicating that the media type (e.g. the Content-Type header on the request) is not supported by this server. For example, a server might return this status code if the user attempts to upload an image in the *.bmp* format, but the server only accepts *.jpeg*.

```
public IActionResult UnsupportedMediaTypeResult()
{
    return new UnsupportedMediaTypeResult();
}
```

Other Status Codes

The above status code results do not cover all the possible HTTP status codes (of which there are many). For situations in which you need to return a status code which isn't given a dedicated action result, we can use the generic `StatusCodeResult` (short method: `StatusCode()`).

```
public IActionResult StatusCodeResult(int statusCode)
{
    return StatusCode(statusCode);
}
```

Status Code with Object Results

These action results are, for the most part, overloads of the results seen in the previous section. However, they are handled differently by the browser or other requesters due to content negotiation.

OkObjectResult

The `OkObjectResult` returns *200 OK* as well as an object.

```
public IActionResult OkObjectResult()
{
    var result = new OkObjectResult(new { message = "200 OK", currentDate = DateTime.Now });
    return result;
}
```

CreateObjectResult

The `CreatedObjectResult` returns *201 Created* and a custom object.

```
public IActionResult CreatedObjectResult()
{
    var result = new CreatedAtActionResult("createdobjectresult", "statuscodeobjects", "", new { message = "201 Created", currentDate = DateTime.Now });
    return result;
}
```

BadRequestObjectResult

The `BadRequestObjectResult` does exactly what you think it does; it returns *400 Bad Request* and an object.

```
public IActionResult BadRequestObjectResult()
{
```

```
    var result = new BadRequestObjectResult(new { message = "400 Bad Request",
currentDate = DateTime.Now });
    return result;
}
```

NotFoundObjectRequest

I imagine by now you're seeing the pattern?
The `NotFoundObjectRequest` returns *404 Not Found* and an object.

```
public IActionResult NotFoundObjectResult()
{
    var result = new NotFoundObjectResult(new { message = "404 Not Found",
currentDate = DateTime.Now });
    return result;
}
```

ObjectResult

For scenarios which aren't covered by the above types, we have the `ObjectResult` class. It returns the specified status code and an object.

```
public IActionResult ObjectResult(int statusCode)
{
    var result = new ObjectResult(new { statusCode = statusCode, currentDate =
DateTime.Now });
    result.StatusCode = statusCode;
    return result;
}
```

Redirect Results

Sometimes we will need to tell the client (e.g. the browser) to redirect to another location. That's where the redirect results come in: they tell the client where to redirect to. Sometimes we just need to go to another action in the same project, but other times we will need to redirect to an external resource.

RedirectResult

The basic `RedirectResult` class (short method: `Redirect()`) redirects to a specified URL.

```
public IActionResult RedirectResult()
```

```
{
    return Redirect("https://www.exceptionnotfound.net");
}
```

LocalRedirectResult

The `LocalRedirectResult` (short method: `LocalRedirect()`) redirects to a URL within the same application. For example, if your site is http://www.mysite.com and you want to redirect to the URL http://www.mysite.com/redirects/target (e.g. the "target" action in the "redirects" controller), you could do so like this:

```
public IActionResult LocalRedirectResult()
{
    return LocalRedirect("/redirects/target");
}
```

RedirectToActionResult

The very common `RedirectToActionResult` class (short method: `RedirectToAction()`) redirects the client to a particular action and controller within the same application. If you wanted to do the same redirect as in the `LocalRedirectResult` example, you would do the following:

```
public IActionResult RedirectToActionResult()
{
    return RedirectToAction("target");
}
```

RedirectToRouteResult

ASP.NET Core MVC has the concept of Routing, by which we can create URL templates which map to specific controllers and actions. Correspondingly, we also have the result `RedirectToRouteResult` (short method: `RedirectToRoute()`) which redirects to a specific route already defined in the application.
Let's say we have the following route defined in our Startup.cs class:

```
app.UseMvc(routes =>
{
    routes.MapRoute(
```

```
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
});
```

We could redirect using that specific route and the `RedirectToRoute()` short method and end up at the same Target action as the previous two examples:

```
public IActionResult RedirectToRouteResult()
{
    return RedirectToRoute("default", new { action = "target", controller = "redirects"
});
}
```

File Results

If we need to return a file to the requester, the File Results let us do so using a variety of formats.

In this demo, we have a file called pdf-sample.pdf in the wwwroot/downloads folder, and we will use that file to demonstrate how various File Result classes work.

FileResult

The basic `FileResult` class (short method: `File()`) returns a file at a given path. In our case, the path is /wwwroot/downloads, and so our action will look as follows:

```
public IActionResult FileResult()
{
    return File("~/downloads/pdf-sample.pdf", "application/pdf");
}
```

Note that "application/pdf" is the MIME type associated with this file.

FileContentResult

There may come a time when we only want to return the content of a given file as a byte array (`byte[]`), not the entire file. For this scenario, we can use the `FileContentResult` class. Note that we still need to specify a MIME type:

```
public IActionResult FileContentResult()
{
    //Get the byte array for the document
    var pdfBytes = System.IO.File.ReadAllBytes("wwwroot/downloads/pdf-sample.pdf");

    //FileContentResult needs a byte array and returns a file with the specified content type.
    return new FileContentResult(pdfBytes, "application/pdf");
}
```

VirtualFileResult

We can also use the `VirtualFileResult` class to get files out of the /wwwroot folder in our project, like so:

```
public IActionResult VirtualFileResult()
{
    //Paths given to the VirtualFileResult are relative to the wwwroot folder.
    return new VirtualFileResult("/downloads/pdf-sample.pdf", "application/pdf");
}
```

PhysicalFileResult

Finally, if we need to get a file from a physical path on our server that isn't necessarily part of our project, we can use the `PhysicalFileResult` class:

```
public IActionResult PhysicalFileResult()
{
    return new PhysicalFileResult(_hostingEnvironment.ContentRootPath +
"/wwwroot/downloads/pdf-sample.pdf", "application/pdf");
}
```

Note that `_hostingEnvironment.ContentRootPath` is the path to the application root, not the /wwwroot folder.

Content Results

The final set of Result classes are the Content Result classes, which are designed to return various kinds of content to the controller.

ViewResult

Possibly the most basic Result class in all of ASP.NET Core MVC is the `ViewResult` class (short method: `View()`), which returns a view.

```
public IActionResult Index()
{
    return View();
}
```

Note that, by default, the `View()` method returns a view with the same name as the action it is called from, in a folder with the same name as the controller. In our case, the controller is "Content" and the action is "Index" so ASP.NET Core MVC will look for a file at /Views/Content/Index.cshtml. You can specify that other views get returned by using overloads of the `View()` short method.

PartialViewResult

It is also possible to return a partial view from an action using the `PartialViewResult` class (short method: `PartialView()`), like so:

```
public IActionResult PartialViewResult()
{
    return PartialView();
}
```

In our demo, the code above will look for a view named "PartialViewResult" in both the /Views/Content directory and the /Views/Shared directory, and will find it in /Views/Shared.

JsonResult

You can easily return JavaScript Object Notation (JSON) content from your application by using the `JsonResult` class (short method: `Json()`).

```
public IActionResult JsonResult()
{
    return Json(new { message = "This is a JSON result.", date = DateTime.Now });
}
```

ContentResult

If you need to return content which doesn't fall into one of the above categories, you can use the general `ContentResult` object (short method: `Content()`) to return your content. In our demo, we will return a simple message, but you can use this class to return more complex content by specifying the MediaTypeHeaderValue or the content type.

```
public IActionResult ContentResult()
{
    return Content("Here's the ContentResult message.");
}
```

Summary

The Action Result classes in ASP.NET Core MVC provide a significant chunk of the functionality you'll be using in your controllers. They return status codes, objects, files, other content, and even redirect the client. As you get more familiar with ASP.NET Core MVC and its functionality, these classes will become second nature; until then, use this post as a quick reference to get your code written.