Marius Schulz
Front End Engineer

Inline Route Constraints in ASP.NET Core MVC

March 31, 2016

One of the big additions of ASP.NET MVC 5 and Web API 2 was attribute routing, which allows for specifying route templates in `[Route]` attributes. That way, the route to a given action was placed right next to the action itself:

```
public class MessagesController : ApiController
{
    [Route("messages/{messageId}")]
    public Message Get(int messageId)
    {
        // ...
    }
}
```

Attribute routing allows you to add **inline route constraints** to the parameters in the route template using the `{parameter:constraint}` syntax. In the above example, we could restrict the `messageId` parameter to match only integers, like this:

```
public class MessagesController : ApiController
{
    [Route("messages/{messageId:int}")]
    public Message Get(int messageId)
    {
        // ...
    }
}
```

If we wanted to add a second action which accepts an ID of type `Guid` rather than `int`, we could use the `guid` route constraint. Without the constraints, both actions would have the same route template; adding an inline constraint would allow the routing engine to differentiate between the two actions:

```
public class MessagesController : ApiController
{
    [Route("messages/{messageId:int}")]
    public Message Get(int messageId)
    {
        // ...
    }
```

```
    [Route("messages/{messageId:guid}")]
    public Message Get(Guid messageId)
    {
        // ...
    }
}
```

The `int` and `guid` inline constraints are just two of many supported route constraints. For a full list, check out Attribute Routing in ASP.NET Web API 2.

Inline Route Constraints in Centralized Routes

In ASP.NET MVC 5 and Web API 2, inline route constraints were only supported in route templates defined within `[Route]` attributes. That is, you could only use inline route constraints in conjunction with attribute routing.

The centralized routing strategy, on the other hand, did **not** support inline route constraints. To constrain certain route parameters, you'd have to use the fourth parameter of the `MapRoute` or `MapHttpRoute` method:

```
routes.MapHttpRoute("Messages", "messages/{messageId}",
    new { controller = "Messages" }
    new { messageId = new IntRouteConstraint() });
```

Luckily, **ASP.NET Core MVC supports inline constraints for routes defined using centralized routing**. That means we can now define routes in our `Startup.cs` file like this:

```
routes.MapRoute("Messages", "messages/{messageId:int}",
    new { controller = "Messages", action = "Get" });
```

I took a look at the aspnet/Routing repository on GitHub and found the following snippet in the `RouteOptions` class. It shows the names and types of all constraints that are supported out of the box:

```
private static IDictionary<string, Type> GetDefaultConstraintMap()
{
    return new Dictionary<string, Type>(StringComparer.OrdinalIgnoreCase)
    {
        // Type-specific constraints
        { "int", typeof(IntRouteConstraint) },
        { "bool", typeof(BoolRouteConstraint) },
        { "datetime", typeof(DateTimeRouteConstraint) },
        { "decimal", typeof(DecimalRouteConstraint) },
        { "double", typeof(DoubleRouteConstraint) },
        { "float", typeof(FloatRouteConstraint) },
        { "guid", typeof(GuidRouteConstraint) },
        { "long", typeof(LongRouteConstraint) },
```

```
            // Length constraints
            { "minlength", typeof(MinLengthRouteConstraint) },
            { "maxlength", typeof(MaxLengthRouteConstraint) },
            { "length", typeof(LengthRouteConstraint) },

            // Min/Max value constraints
            { "min", typeof(MinRouteConstraint) },
            { "max", typeof(MaxRouteConstraint) },
            { "range", typeof(RangeRouteConstraint) },

            // Regex-based constraints
            { "alpha", typeof(AlphaRouteConstraint) },
            { "regex", typeof(RegexInlineRouteConstraint) },

            { "required", typeof(RequiredRouteConstraint) },
        };
    }
```

We can constrain route parameters to be of a certain **type**, have a given **length**, be in a specified **range**, or match a given **regular expression** — all of that inline, directly in the route template. Sweet!