ASP.NET 5 and MVC 6
Outline

• Motivation
• DNX
• ASP.NET 5
• MVC 6
Motivation

• Modern web stack
  • Updated build system (no build step)
  • Modern package system (NuGet)
  • Lightweight/composable runtime
  • Dependency injection everywhere
  • Flexible configuration/deployment
  • Unify MVC/Web API

• Cross platform/portability
  • SxS CLR
Core CLR

• Open source version of .NET
  • https://github.com/dotnet/coreclr
  • Contains core runtime and mscorlib (e.g. GC, JIT, BCL)
  • Dot not contain many frameworks (e.g. WCF, WPF)

• Cross platform
  • Windows, Linux, Mac, FreeBSD

• Portable
  • Designed to be ~/bin deployed
The DNX (a .NET Execution Environment) contains the code required to bootstrap and run an application, including the compilation system, SDK tools, and the native CLR hosts.

DNX

• SDK/tooling to use a CLR
  • dnvm, dnx, dnu, project.json
• Runtime host to load a CLR
  • Command line or from other host (e.g. IIS)
• Application host
  • Compile application with Roslyn
  • Invoke application with dependency injection
Comparison to Node.js

<table>
<thead>
<tr>
<th></th>
<th>Node</th>
<th>DNX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Runtime</strong></td>
<td>JavaScript</td>
<td>C#/Roslyn</td>
</tr>
<tr>
<td></td>
<td>V8</td>
<td>CoreCLR</td>
</tr>
<tr>
<td><strong>Tooling</strong></td>
<td>npm</td>
<td>dnu/NuGet</td>
</tr>
<tr>
<td></td>
<td>Node</td>
<td>dnx</td>
</tr>
<tr>
<td><strong>Frameworks</strong></td>
<td>Connect*</td>
<td>ASP.NET 5</td>
</tr>
<tr>
<td></td>
<td>Express*</td>
<td>MVC 6</td>
</tr>
<tr>
<td></td>
<td>Sequelize*</td>
<td>EF 7</td>
</tr>
<tr>
<td></td>
<td>Socket.io*</td>
<td>SignalR 3</td>
</tr>
</tbody>
</table>

* and typically between 5 and 20 other choices
Getting DNX (without VisualStudio)

• Windows

```powershell
@powershell -NoProfile -ExecutionPolicy unrestricted -Command "&{$Branch='dev';iex ((new-object net.webclient).DownloadString('https://raw.githubusercontent.com/aspnet/Home/dev/dnvminstall.ps1'))}"
```

• Mac

```bash
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/aspnet/Home/dev/dnvminstall.ps1)"
brew tap aspnet/dnx
brew update
brew install dnvm
```
C:\Users\brock> @powershell -NoProfile -ExecutionPolicy unrestricted -Command "&{ $Branch='dev';iex ((new-object net.webclient).DownloadString('https://raw.githubusercontent.com/aspnet/Home/dev/dnvminstall.ps1'))}"

Using temporary directory: C:\Users\brock\AppData\Local\Temp\dnvminstall

Downloading DNVM.ps1 to

Downloading DNVM.cmd to

Installing DNVM

Installing .NET Version Manager to C:\Users\brock\.dnx\bin

Creating destination folder 'C:\Users\brock\.dnx\bin'

Installing 'dnvm.ps1' to 'C:\Users\brock\.dnx\bin' ...

Installing 'dnvm.cmd' to 'C:\Users\brock\.dnx\bin' ...

Adding C:\Users\brock\.dnx\bin to Process PATH

Adding C:\Users\brock\.dnx\bin to User PATH

Adding C:\Users\brock\.dnx to Process DNX_HOME

Adding C:\Users\brock\.dnx to User DNX_HOME

C:\Users\brock> which dnvm
C:\Users\brock\.dnx\bin\dnvm.cmd

C:\Users\brock> _
DNX command line tools

• DNVM – CLR installer/chooser
  • PATH contains %USERPROFILE%\dnx\bin
  • Contains dnvm.cmd and dnvm.ps1

• Runtime specific tools
  • DNU – project utility
  • DNX – application loader
DNVM

- Installs/uses a version of the CLR
- Commands
  - list
  - install
  - use
- "use" adds a runtime to PATH
DNX

• Hosts an application
  • Loads CLR
  • Compiles application code
  • Executes Main

```csharp
using System;

public class Program
{
    public void Main()
    {
        Console.WriteLine("Hello DNX!"无可救药的修复;}
    }
}
```
DNX

- project.json required
- Configures
  - Runtimes
  - Dependencies
  - Commands
  - And more...

```json
{
  "dependencies": {
    "Microsoft.AspNet.Mvc": "6.0.0-beta4"
  },

  "frameworks":{
    "dnx451":{},
    "dnxcore50": {
      "dependencies": {
        "System.Console": "4.0.0-beta-22816"
      }
    }
  },

  "commands" : {
    "my_command": "YourApp"
  }
}
```
Running DNX

- dnx <path> <command>
  - <path> to application or project.json
  - <command> is project name or command from project.json

- dnx <path> run
  - <path> to application or project.json
  - "run" is hard coded command to use <path>'s project name
DNU

- Utility to manage:
  - Dependencies
  - Packaging
  - Publishing
  - Other utilities

- Uses project.json
  - Produces project.lock.json
Application development details

• Command line arguments
• Inter-project dependencies
• Dependency injection
Passing command line arguments

• Parameters passed after `<command>` are passed to application

```csharp
using System;
using System.Linq;

public class Program
{
    public void Main(string[] args)
    {
        Console.WriteLine(
            args.Aggregate((x,y) => x + ", " + y)
        );
    }
}
```
Inter-project dependencies

• Projects can reference other projects
  • Source-level dependency, rather than NuGet packages

• Dependencies are located in:
  • Implicit via project's parent directory
  • Explicit via global.json
    • global.json resides in ancestor directory

```json
{
  "projects" : [ 
    "src",
    "test",
    "c:\\other"
  ]
}
```
Dependency injection

- Dependencies are injected into Program's ctor

```csharp
using System;
using Microsoft.Framework.Runtime;

public class Program
{
    private readonly IApplicationEnvironment _env;

    public Program(IApplicationEnvironment env)
    {
        _env = env;
    }

    public void Main(string[] args)
    {
        Console.WriteLine(_env.ApplicationName);
        Console.WriteLine(_env.ApplicationBasePath);
        Console.WriteLine(_env.RuntimeFramework);
    }
}
```
Visual Studio 2015

• Project.json is project file
  • Runs the command line tools

• Adds bells and whistles
  • Intellisense/quick actions
  • UI for NuGet
  • UI for tooling
  • Debugger
OmniSharp

• Roslyn as a service
  • Intellisense
  • Statement completion
  • Refactoring

• Plugins for common editors
  • Sublime
  • Atom
  • VIM
  • Brackets
  • VS Code
  • Emacs
ASP.NET 5

- New hosting model
- New HTTP pipeline
ASP.NET 5

- ASP.NET 5 is HTTP pipeline implementation
  - supports various servers (IIS, WebListener, Kestrel..)
  - can run OWIN and "native" middleware
  - Uses a higher level abstraction over OWIN concept (`HttpContext` & `RequestDelegate`)

- MVC 6 is Microsoft's application framework
  - is OWIN compatible
How ASP.NET 5 gets invoked

- `dnx . command`
  - Microsoft.Framework.ApplicationHost
  - Helios
  - Microsoft.AspNet.Hosting (WebListener, Kestrel, IIS)
  - Startup.cs
    - ConfigureServices(...)
    - Configure(...)
Pipeline primitives

```
app.Use(context, next)
app.Map("/path")
app.Use(context, next)
app.Run(context)
```
IApplicationBuilder.Run(RequestDelegate handler)

namespace Microsoft.AspNet.Builder
{
    public delegate Task RequestDelegate(HttpContext context);
}

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello ASP.NET5");
});
IApplicationBuilder **Map**(string path, Action<IApplicationBuilder> app)

```csharp
app.Map("/hello", helloApp =>
{
    helloApp.Run(async (HttpContext context) =>
    {
        await context.Response.WriteAsync("Hello ASP.NET5");
    });
});
```
Use(
Func<RequestDelegate, RequestDelegate> middleware)
Middleware classes

```csharp
public class InspectionMiddleware
{
    private readonly RequestDelegate _next;

    public InspectionMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task Invoke(HttpContext context)
    {
        Console.WriteLine($"request: {context.Request.Path}");
        await _next(context);
    }
}

app.UseMiddleware<InspectionMiddleware>();
```
Dependency Injection

• DI (almost) everywhere
  • Startup ctor
  • ConfigureServices/Configure
  • Inline middleware
  • Middleware classes

• Host provided dependencies (e.g. IApplicationEnvironment, LoggerFactory)

• Dependencies provided in ConfigureServices
public class Startup
{
    public Startup(IApplicationEnvironment environment)
    { /* stuff */ }

    public void ConfigureServices(IServiceCollection services, ILoggerFactory factory)
    { /* register more stuff */ }

    public void Configure(IApplicationBuilder app, ISomeService someService)
    {
        app.Run(async (HttpContext context, IMyCustomService custom) =>
        {
            await context.Response.WriteAsync(c.Foo());
        });
    }
}
Registering dependencies

- New instance "per call"

```csharp
services.AddTransient<IMyCustomService, MyCustomService>();
```

- New instance per HTTP request

```csharp
services.AddScoped<IMyCustomService, MyCustomService>();
```

- Singleton

```csharp
services.AddSingleton<IMyCustomService, MyCustomService>();
```
Example: Inject current user into dependency

```csharp
public void ConfigureServices(IServiceCollection services, ILoggerFactory factory)
{
    services.AddTransient<IMyCustomService>(provider =>
    {
        var context = provider.GetRequiredService<IHttpContextAccessor>();
        return new MyCustomServiceWithUser(context.HttpContext.User);
    });
}
```
Configuration

• web.config is no more

• New configuration system based on key/value pairs
  • command line
  • environment variables
  • JSON files
  • INI files

• Configuration can come from multiple sources
  • last source wins
Example

```csharp
public class Startup
{
    public IConfiguration Configuration { get; set; }

    public Startup(IHostingEnvironment env)
    {
        Configuration = new Configuration()
            .AddJsonFile("config.json")
            .AddJsonFile($"config.{env.EnvironmentName}.json", optional: true)
            .AddEnvironmentVariables();
    }

    // more
}
```
Using configuration

```csharp
public class Startup
{
    IConfiguration _configuration;

    public Startup()
    {
        _configuration = new Configuration()
            .AddJsonFile("config.json");
    }

    public void Configure(IApplicationBuilder app)
    {
        var copyright = new Copyright
        {
            Company = _configuration.Get("copyright:company"),
            Year = _configuration.Get("copyright:year")
        };

        app.Run(async (context) =>
        {
            await context.Response.WriteAsync($"Copyright {copyright.Year}, {copyright.Company}"));
        });
    }
}
```

```json
{
    "copyright": {
        "year": "2015",
        "company": "Foo Industries"
    }
}
```
Options

• Options is a pattern introduced by DNX
  • Convert raw name/value pairs into strongly typed classes
  • Put options into DI system

• Heavily used throughout ASP.NET 5 / MVC 6
Example

```csharp
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        // initialize options system
        services.AddOptions();

        // de-serialize and register config settings
        services.Configure<Copyright>(_configuration.GetSubKey("copyright"));
    }

    public void Configure(IApplicationBuilder app)
    {
        app.Run(async (HttpContext context, IOptions<Copyright> copyright) =>
        {
            await context.Response.WriteAsync($"Copyright {copyright.Options.Year}, {copyright.Options.Company}"));
        });
    }
}
```
Packaging & Deployment

• **dnu pack**
  • Creates Nuget package containing all compilation targets

• **dnu publish**
  • Creates deployable web application
  • Self-contained for DNXCore
MVC 6

• Packaging
• Middleware
• Routing and action selection
• Controller initialization
• Model binding changes
• Razor
• Filters
• APIs
• Error handling
Packaging

• MVC 6 is packaged entirely as a NuGet
  • Microsoft.AspNet.Mvc

```json
{
  "dependencies": {
    "Microsoft.AspNet.Server.IIS": "1.0.0-beta4",
    "Microsoft.AspNet.Mvc": "6.0.0-beta4"
  }
}
```
Middleware

- MVC 6 is configured as middleware
  - In ConfigureServices via AddMvc
  - In Configure via UseMvc

```csharp
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app)
    {
        app.UseMvc();
    }
}
```
Overriding default settings

• ConfigureMvc used to override defaults

```csharp
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc().ConfigureMvc(mvc =>
        {
            mvc.AntiForgeryOptions.CookieName = "__antixsrf";
            mvc.Filters.Add(...);
            mvc.ViewEngine.Add(...);
        });
    }
}
```
Routing

• Routes configured via UseMvc
  • RouteParameters.Optional from MVC 5 removed

```csharp
public void Configure(IApplicationBuilder app)
{
    app.UseMvc(routes =>
    {
        routes.MapRoute("old_default",
            "{controller}/{action}",
            new {
                controller = "Home", action="Index"
            });

        routes.MapRoute("new_default",
            "{controller=Home}/{action=Index}/{id?}");
    });
}
```
Controllers

• Controller base class still provided
  • Action results now implement IActionResult
  • Controller base provides many helpers to create action results
    • View(), Content(), Created(), HttpNotFoundException, Unauthorized(), BadRequest()

```csharp
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```
Attribute routing

- Attribute routing enabled by default

```csharp
public class HomeController : Controller
{
    // ~/ or ~/hello-world
    [Route("/")] [Route("/hello-world")]
    public IActionResult Index()
    {
        return View();
    }
}
```
Attribute routing

• Attribute routing can be applied to class
• [controller] and [action] act as tokens

```csharp
[Route("[controller]/[action]"edm)
public class HomeController : Controller
{
    // ~/Home/Index
    public IActionResult Index()
    {
        return View();
    }
}
```
Combining Route attributes

• Route attributes inherit path
  • RoutePrefix from MVC 5 removed
• Can replace inherited path
  • If template starts with "/" or "~/"

```csharp
[Route("[controller]")] public class HomeController : Controller {
    // ~/Home/hello
    [Route("hello")] public IActionResult Index() {
        return View();
    }

    // ~/hello
    [Route("/hello")] public IActionResult Index2() {
        return View();
    }
}
```
Route parameters

• [Route] allows parameters
  • With {param} syntax

• Supports filters
  • With {param:filter} syntax

```csharp
[Route("[controller]/[action]")]
public class HomeController : Controller
{
    // GET ~/Home/Index
    public IActionResult Index()
    {
        return View();
    }

    // GET ~/Home/Index/5
    [Route("{id:int}\")]
    public IActionResult Index(int id)
    {
        return View();
    }
}
```
HTTP method based routes

- HttpGet, HttpPost, HttpPut, HttpDelete, HttpPatch
  - Filter action method on request method
  - Build on [Route] semantics

```csharp
[Routed("[controller]/[action]")]
public class HomeController : Controller
{
    // GET ~/Home/Index
    [HttpGet]
    public IActionResult Index()
    {
        return View();
    }

    // ~/Submit
    [HttpPost("/Submit")]
    public IActionResult Submit()
    {
        return View();
    }
}
```
Areas

• Areas defined with the [Area] attribute
  • Used to match an {area} route param
  • Attribute routing allows [area] route token
• Views must still reside under ~/Areas/<area>/Views/<controller>

```csharp
public void Configure(IApplicationBuilder app)
{
    app.UseMvc(routes =>
    {
        routes.MapRoute("new_default",
            "{area}/{controller=Home}/{action=Index}/{id?}");
    });
}
```

```
[Area("account")]
public class HomeController : Controller
{
    // ...
}
```
POCO controllers

• Controller classes can be POCO
  • Discovered in projects that reference Microsoft.AspNet.Mvc.*
  • Identified by "Controller" class name suffix
  • [NonController] disables
Dependecy injection

- Can inject dependencies into controllers via:
  - Constructor
  - Properties via [FromServices]
  - Action parameters via [FromServices]

```csharp
public class HomeController
{
    IHttpContextAccessor _accessor;

    public HomeController(IHttpContextAccessor accessor)
    {
        _accessor = accessor;
    }

    [FromServices]
    public IHttpContextAccessor Accessor { get; set; }

    public IActionResult Index()
    {
        return new ViewResult()
        {
            ViewName = "Index"
        };
    }
}
```
Per-request context objects

• Can inject certain MVC requests objects
  • HttpContext, ActionContext, ModelStateDictionary, ViewDataDictionary, etc.
  • Properties via [Activate]
  • Action parameters via [Activate]

```csharp
public class HomeController
{
    [Activate]
    public ViewDataDictionary ViewData { get; set; }

    [Activate]
    public HttpContext Context { get; set; }

    public IActionResult Index()
    {
        ViewData["message"] = "Hello MVC 6!";
        return new ViewResult()
        {
            ViewName = "Index",
            ViewData = ViewData
        };
    }
}
```
Model binding changes

• Implicit model binding from route, query, and form
  • Default binding order changed to: 1) route, 2) query, 3) form

• Explicit model binding possible using:
  • [FromRoute]
  • [FromQuery]
  • [FromForm]
  • [FromHeader]
  • [FromServices]
  • [FromBody]
Razor

• Shared config
  • _ViewStart and _GlobalImport

• Chunks
  • @ directives

• TagHelpers
  • Like WebForms custom controls

• ViewComponents
  • Child action replacements
Shared razor configuration

• _ViewStart.cshtml still exists
  • Can now easily be put in application root
  • Layout assignment no longer is full path

• _GlobalImport.cshtml is new
  • Soon to be called _ViewImports.cshtml
  • Allows for sharing @using, @addTagHelper chunks across views
  • Can be located in the same places as _ViewStart.cshtml
Razor directives (aka chunks)

• @model, @using, @section, @functions still exist
• @helper is gone
• @inject, @addTagHelper are new

• Also, @await Html.PartialAsync() is new
@inject

- Allows dependency injection into view
  - @inject <type> <property>

```csharp
@using Microsoft.Framework.OptionsModel
@inject IOptions<MyConfig> Config

<h2>@Config.Options.SiteName</h2>
```
Tag helpers

• Like custom controls for MVC
  • Allow server-side code to inspect the element
  • Can modify attributes, tag, and/or contents

• @addTagHelper "Namespace.ClassName, Assembly"
  • Or @addTagHelper ",* , Assembly"

@addTagHelper "SpanTagHelper, YourProjectName"

<span emoji="smile" />
Tag helper implementation

• TagHelper base class
  • Class name used to match element

• Override Process or ProcessAsync
  • Inspect element via TagHelperContext
  • Alter output via TagHelperOutput

```csharp
public class SpanTagHelper : TagHelper
{
    public override void Process(
        TagHelperContext context, TagHelperOutput output)
    {
        if (context.AllAttributes.ContainsKey("emoji") &&
            "smile" == context.AllAttributes["emoji"].ToString())
        {
            output.Attributes.Add("title", "smile");
            output.Content.SetContent(" :) ");
            output.SelfClosing = false;
        }
    }
}
```
Tag helper implementation

- `[TargetElement]` can be used to match element
  - Attributes can be used to filter
- `[HtmlAttributeName]` will read incoming attribute
  - Will remove from output

```csharp
[TestElement("span", Attributes = "emoji")]
public class EmojiTagHelper : TagHelper
{
    [HtmlAttributeName("emoji")]
    public string Emoji { get; set; }

    public override void Process(
        TagHelperContext context,
        TagHelperOutput output)
    {
        if ("smile" == Emoji)
        {
            output.Attributes.Add("title", "smile");
            output.Content.SetContent(" :) ");
            output.SelfClosing = false;
        }
    }
}
```
MVC tag helpers

<a asp-controller="Manage" asp-action="Index">Manage Your Account</a>

<form asp-controller="Account" asp-action="LogOff" method="post"></form>

<environment names="Staging,Production">
  <h1>You're in production!</h1>
</environment>

<link rel="stylesheet"
  href="/ajax.aspnetcdn.com/ajax/bootstrap/3.0.0/css/bootstrap.min.css"
  asp-fallback-href="~/lib/bootstrap/css/bootstrap.min.css"
  asp-fallback-test-class="hidden"
  asp-fallback-test-property="visibility"
  asp-fallback-test-value="hidden" />

<script src="/ajax.aspnetcdn.com/ajax/jquery.validation/1.11.1/jquery.validate.min.js"
  asp-fallback-src="~/lib/jquery-validation/jquery.validate.js"
  asp-fallback-test="window.jquery && window.jquery.validator">
</script>
Validation tag helpers

```html
<form asp-controller="Account" asp-action="ForgotPassword" method="post">
    <h4>Enter your email.</h4>
    <div asp-validation-summary="ValidationSummary.All" class="text-danger"></div>
    <div>
        <label asp-for="Email"></label>
        <input asp-for="Email" class="form-control" />
        <span asp-validation-for="Email" class="text-danger"></span>
    </div>
</form>
```
View components

• Replacement for child actions
  • Partial views still exist

• Allow for a partial view that runs controller-like code
  • Supports dependency injection

@Component.Invoke("Menu", 3)

Or

@await Component.InvokeAsync("Menu", 3)
View components

- **ViewComponent base class**
  - Matched by class prefix
  - Or can use `[ViewComponent]` on POCO
- **Implement Invoke or InvokeAsync**
  - Returns `IViewComponentResult` or `Task<IViewComponentResult>`

```csharp
class MenuViewComponent : ViewComponent
{
    ICustomMenuService _menu;

    public MenuViewComponent(ICustomMenuService menu)
    {
        _menu = menu;
    }

    public IViewComponentResult Invoke(int depth)
    {
        var menuModel = _menu.GetMenu(depth);
        return View("Index", menuModel);
    }
}
```
View components

- View component views are under:
  - ~/Views/<controller>/Components/<component>
  - ~/Views/Shared/Components/<component>
Filters

• Dependency injection
• Resource filter
• Async support
TypeFilter

- Allows for filters that require dependency injection
  - Implemented via IFilterFactory

```csharp
public class MyActionFilter : Attribute, IActionFilter
{
    private IHostingEnvironment _env;

    public MyActionFilter(IHostingEnvironment env)
    {
        _env = env;
    }

    // ...
}

[TypeFilter(typeof(MyFilter))]
public IActionResult Index()
{
    // ...
}
```
IResourceFilter

• Surrounds model binding, action, and result (including those filters)

```csharp
public interface IResourceFilter : IFilter
{
    void OnResourceExecuting(ResourceExecutingContext context);
    void OnResourceExecuted(ResourceExecutedContext context);
}
```

• ResourceExecutingContext
  • Value providers, model binders, input formatters, validation providers
  • Can alter these on each request
Async filters

• All filters now have IAsync<Filter> support
  • Authorization, Resource, Action, Result, Exception
  • Pattern similar to middleware pipeline

```csharp
public class MyResourceFilter : Attribute, IAsyncResourceFilter
{
    public async Task OnResourceExecutionAsync(
        ResourceExecutingContext context,
        ResourceExecutionDelegate next)
    {
        // pre
        var resourceExecutedContext = await next();
        // post
    }
}
```
Web API

• Formatters
  • Content negotiation
  • Format filters
  • XML support
Formatters

• Formatters have been split into two groups
  • Input formatters triggered via [FromBody]
  • Output formatters triggered via ObjectResult
Input formatters

- InputFormatter base class provides starting point
  - SupportedMediaTypes property used to match Content-Type header
- Unsupported Media Type (415) returned only if [Consumes] filter used

<table>
<thead>
<tr>
<th>Formatter</th>
<th>Content type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>StringInputFormatter</td>
<td>text/plain</td>
<td></td>
</tr>
<tr>
<td>JsonInputFormatter</td>
<td>application/json, text/json</td>
<td></td>
</tr>
<tr>
<td>XmlSerializerInputFormatter</td>
<td>application/xml, text/xml</td>
<td>Not registered by default</td>
</tr>
<tr>
<td>XmlDataContractSerializerInputFormatter</td>
<td>application/xml, text/xml</td>
<td>Not registered by default</td>
</tr>
</tbody>
</table>
Output formatters

• ObjectResult chooses formatter from Accept header
  • ContentTypes property can be set explicitly to limit formatters
  • OutputFormatter base class has SupportedMediaTypes property
  • If Accept contains "*/*") then rest of Accept values ignored
    • RespectBrowserAcceptHeader onMvcOptions can change behavior
  • Not Acceptable (406) returned if no formatter able to format result

<table>
<thead>
<tr>
<th>Formatter</th>
<th>Accept type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>StringOutputFormatter</td>
<td>text/plain</td>
<td></td>
</tr>
<tr>
<td>JsonOutputFormatter</td>
<td>application/json, text/json</td>
<td></td>
</tr>
<tr>
<td>XmlSerializerOutputFormatter</td>
<td>application/xml, text/xml</td>
<td>Not registered by default</td>
</tr>
<tr>
<td>XmlDataContractSerializerOutputFormatter</td>
<td>application/xml, text/xml</td>
<td>Not registered by default</td>
</tr>
</tbody>
</table>
FormatFilter & FormatFilterAttribute

- Allows overriding of Accept header
  - Looks for "format" route or query param
  - FormatterMappings on MvcOptions indicates format to media type mapping
- Sets ContentTypes on ObjectResult

```csharp
public void Configure(IApplicationBuilder app)
{
    app.UseMvc(routes =>
    {
        routes.MapRoute("default",
        "api/{controller} ");

        routes.MapRoute("formatted",
        "api/{controller}.{format}");
    });
}
```
ProducesAttribute

- Result filter to set ContentTypes on ObjectResult
  - Does not override format filters

```csharp
[HttpGet]
[Produces("application/json", "application/xml")]
public object Get()
{
    return new {...};
}
```
XML compatibility shim

- Library to help migrate from Web API to MVC 6
- Provides old classes that map to the new framework
  - ApiController
  - FromUriAttribute
  - HttpResponseMessage helpers/extensions/model binder
  - HttpResponseMessage helpers/extensions/formatter
  - HttpResponseException
  - HttpResponseException
  - HttpNotFound
Error handling

• HandleError from MVC 5 has been removed
• Resource filter's post processing runs after exception filters
  • Last chance place to "handle" exceptions with a result
  • Or just can log exceptions
Error pages

• Diagnostics middleware
  • Microsoft.AspNet.Diagnostics
  • UseErrorPages useful for development/debugging error info
  • UseErrorHandler useful for production error pages
    • Logs error information
    • Invokes error path
Summary

• Brave new .NET world
• ASP.NET 5 is a node/OWIN-like HTTP pipeline
• MVC 6 is quite a make over of MVC 5 and Web API