Wintellect®
Know how.

Best Practices in Modern Web Development

Wintellect NOW

Consulting / Training
wintellect.com

Wintellect®
Know how.

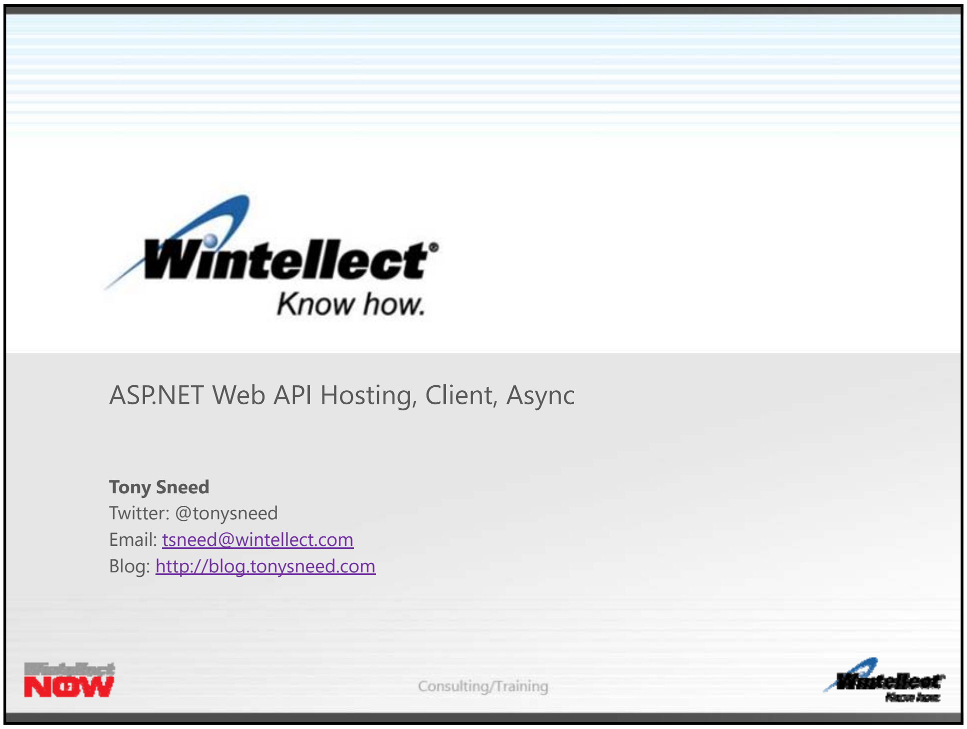
Best Practices in Modern Web Development

Wintellect®
Know how.

PUBLISHED BY
Wintellect, LLC
990 Hammond Drive
Bldg. One, Suite 302
Atlanta, GA 30328

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

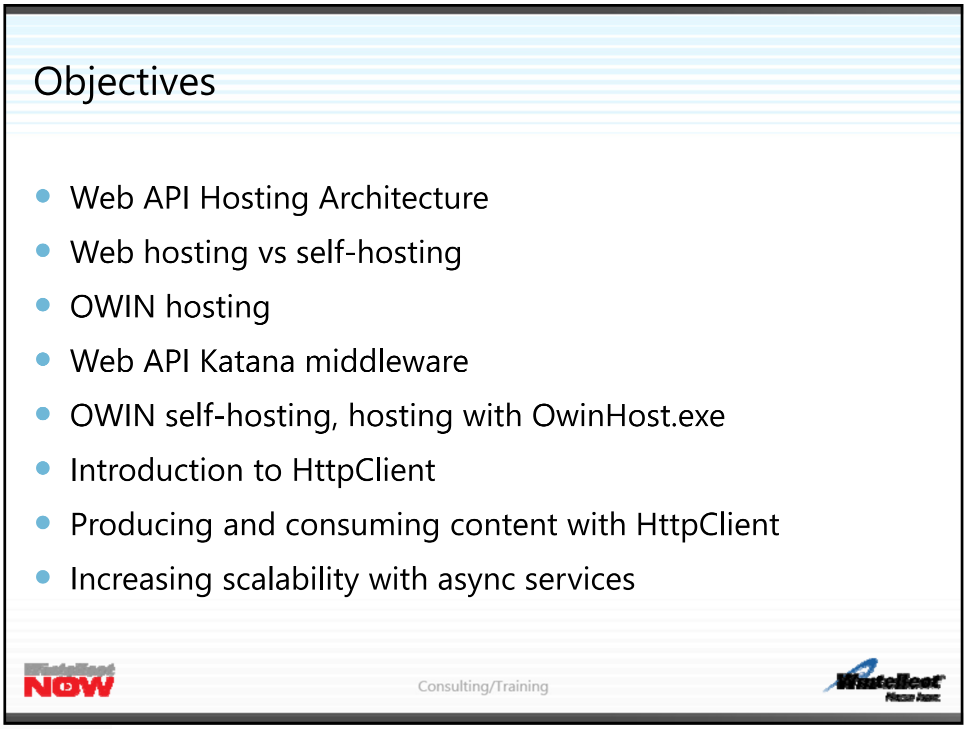
Table of Contents



ASP.NET Web API Hosting, Client, Async

Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

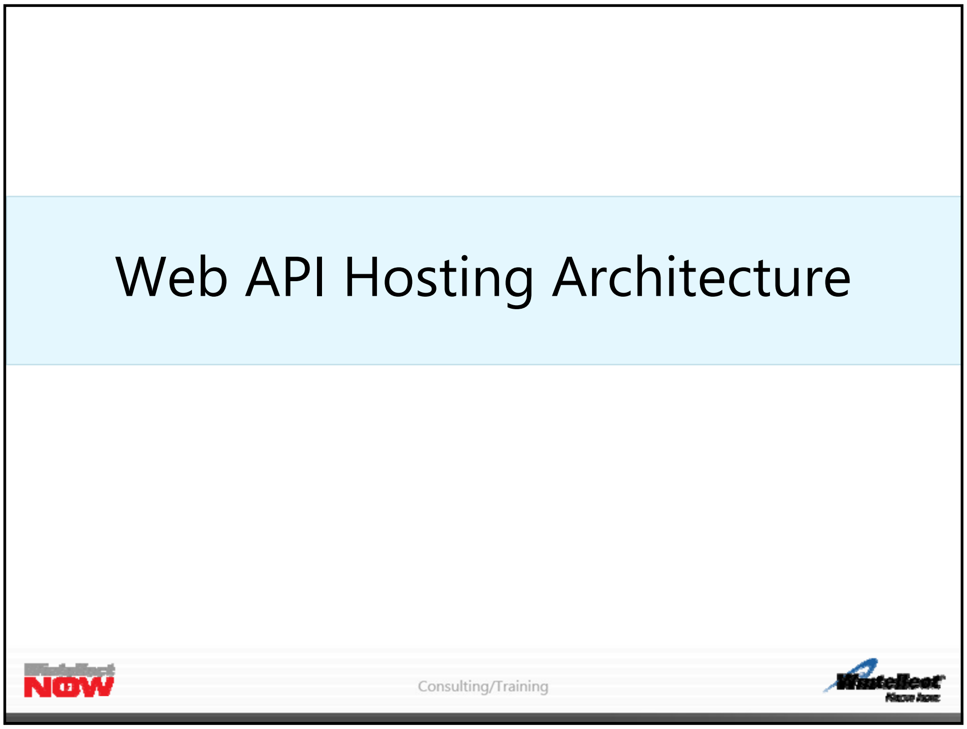
Consulting/Training



Objectives

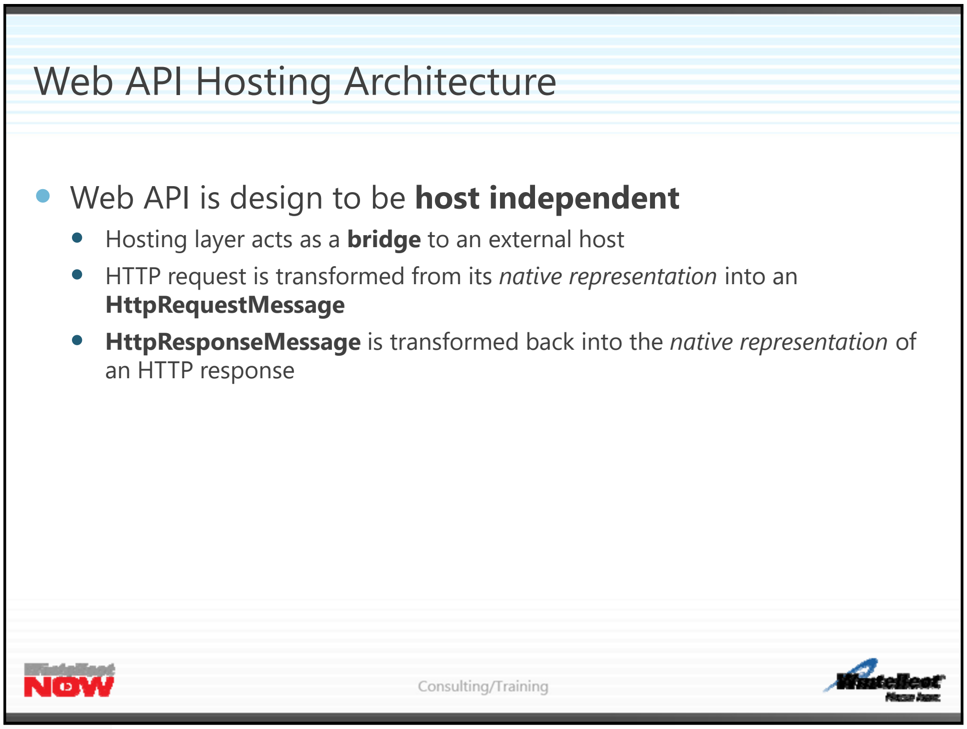
- Web API Hosting Architecture
- Web hosting vs self-hosting
- OWIN hosting
- Web API Katana middleware
- OWIN self-hosting, hosting with OwinHost.exe
- Introduction to HttpClient
- Producing and consuming content with HttpClient
- Increasing scalability with async services

Consulting/Training



Web API Hosting Architecture

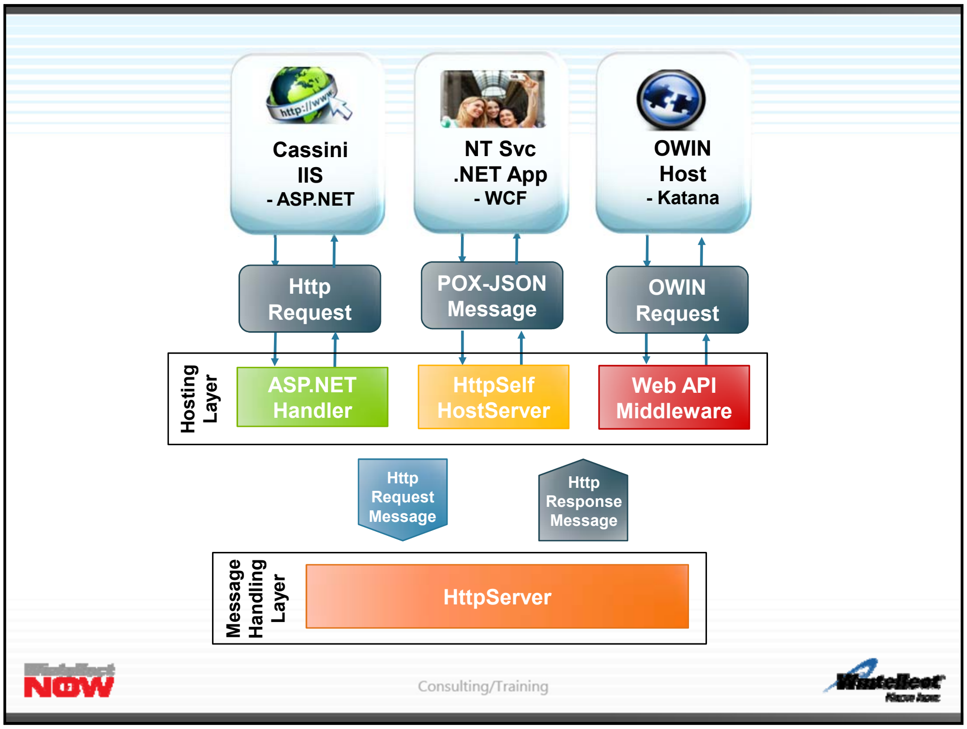
Consulting/Training



Web API Hosting Architecture

- Web API is design to be **host independent**
  - Hosting layer acts as a **bridge** to an external host
  - HTTP request is transformed from its *native representation* into an **HttpRequestMessage**
  - **HttpResponseMessage** is transformed back into the *native representation* of an HTTP response

Consulting/Training



Cassini
IIS
- ASP.NET

NT Svc
.NET App
- WCF

OWIN
Host
- Katana

Http Request

POX-JSON Message

OWIN Request

Hosting Layer

ASP.NET Handler

HttpSelf HostServer

Web API Middleware

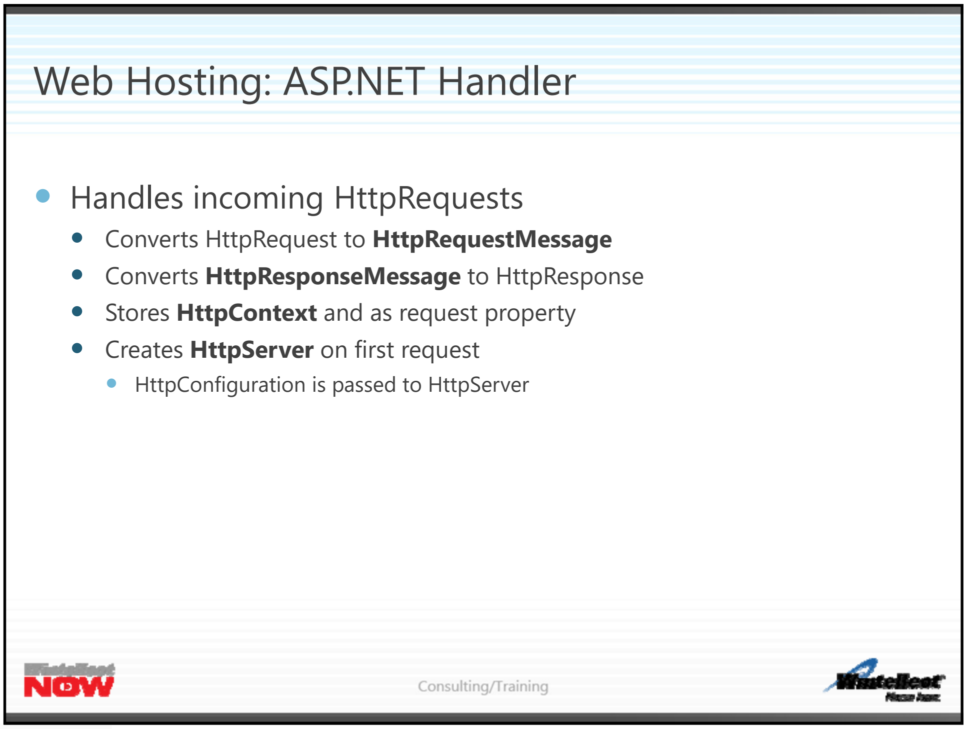
Http Request Message

Http Response Message

Message Handling Layer

HttpServer

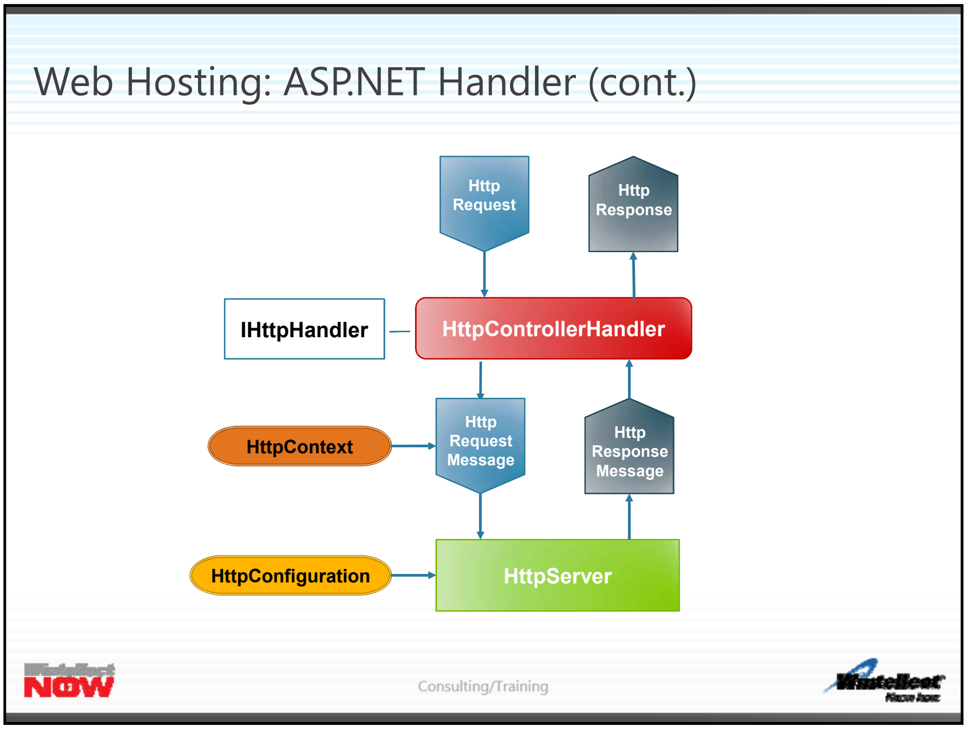
Consulting/Training



Web Hosting: ASP.NET Handler

- Handles incoming HttpRequests
  - Converts HttpRequest to **HttpRequestMessage**
  - Converts **HttpResponseMessage** to HttpResponse
  - Stores **HttpContext** and as request property
  - Creates **HttpServer** on first request
    - HttpConfiguration is passed to HttpServer

Consulting/Training



Web Hosting: ASP.NET Handler (cont.)

Http Request

Http Response

IHttpHandler

HttpControllerHandler

HttpContext

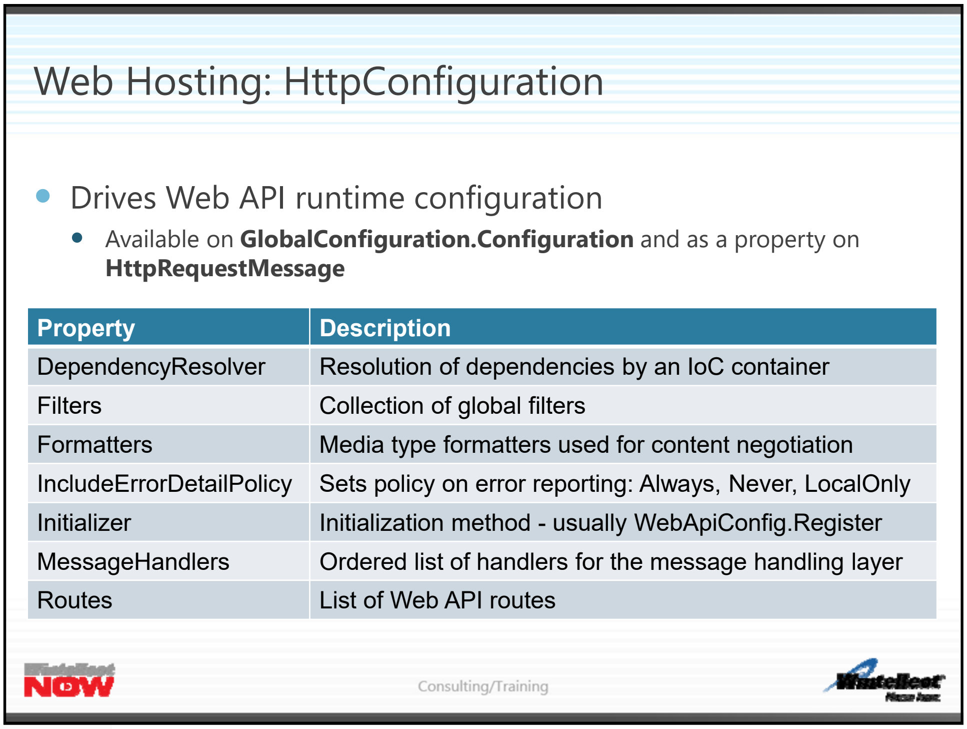
Http Request Message

Http Response Message

HttpConfiguration

HttpServer

Consulting/Training

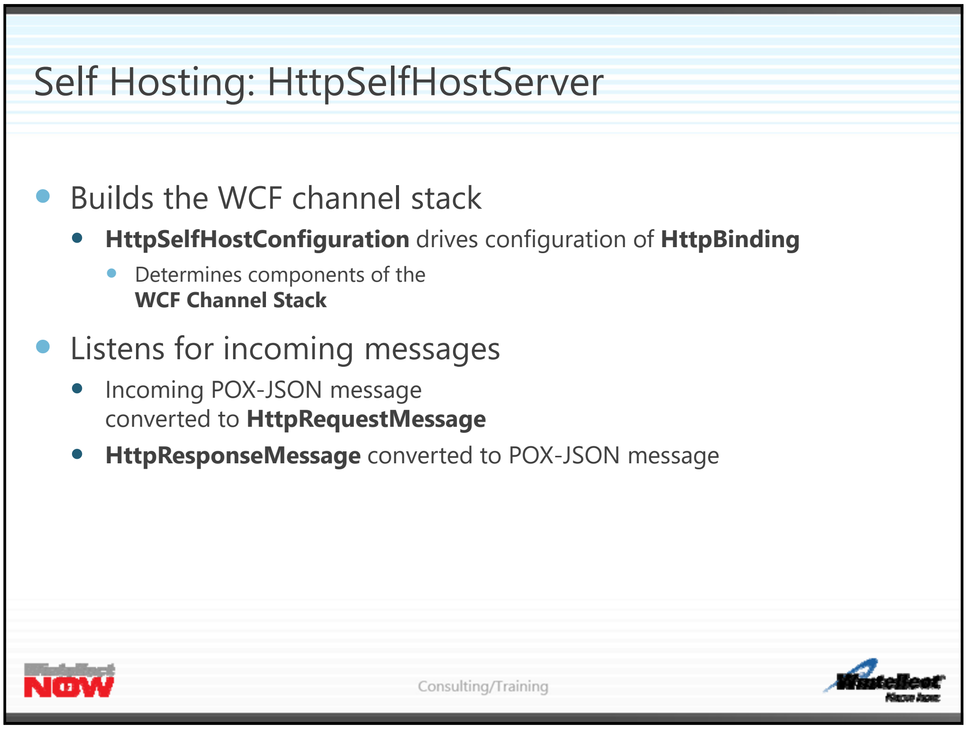


Web Hosting: HttpConfiguration

- Drives Web API runtime configuration
  - Available on **GlobalConfiguration.Configuration** and as a property on **HttpRequestMessage**

Property	Description
DependencyResolver	Resolution of dependencies by an IoC container
Filters	Collection of global filters
Formatters	Media type formatters used for content negotiation
IncludeErrorDetailPolicy	Sets policy on error reporting: Always, Never, LocalOnly
Initializer	Initialization method - usually WebApiConfig.Register
MessageHandlers	Ordered list of handlers for the message handling layer
Routes	List of Web API routes

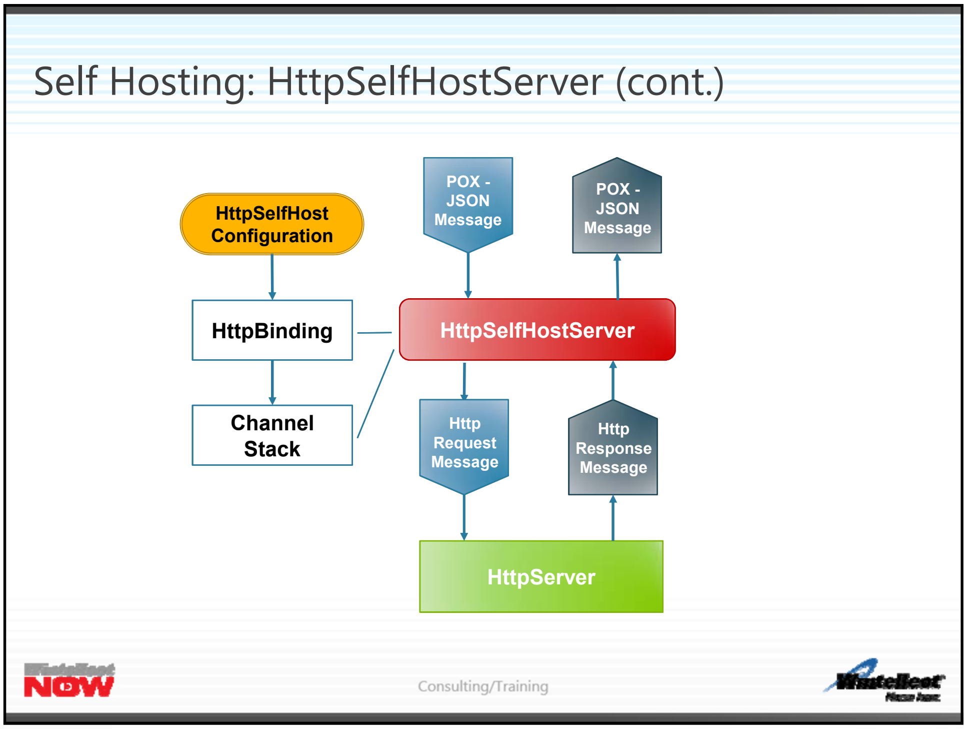
Consulting/Training



Self Hosting: HttpSelfHostServer

- Builds the WCF channel stack
  - **HttpSelfHostConfiguration** drives configuration of **HttpBinding**
    - Determines components of the **WCF Channel Stack**
- Listens for incoming messages
  - Incoming POX-JSON message converted to **HttpRequestMessage**
  - **HttpResponseMessage** converted to POX-JSON message

Consulting/Training



Self Hosting: HttpSelfHostServer (cont.)

HttpSelfHost Configuration

HttpBinding

Channel Stack

POX - JSON Message

POX - JSON Message

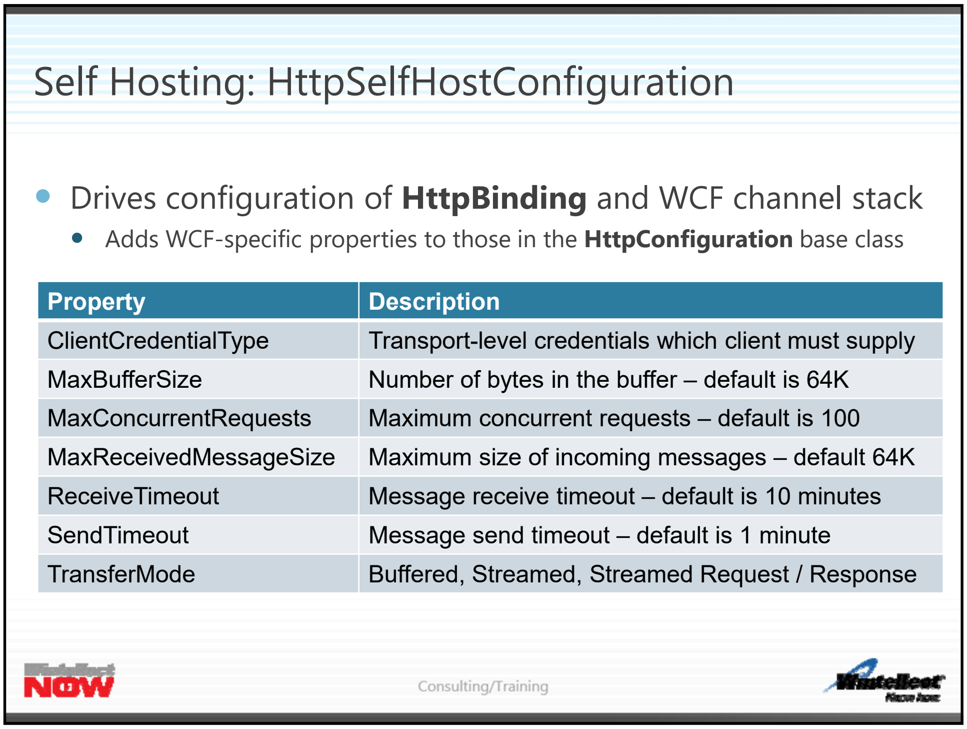
HttpSelfHostServer

Http Request Message

Http Response Message

HttpServer

Consulting/Training

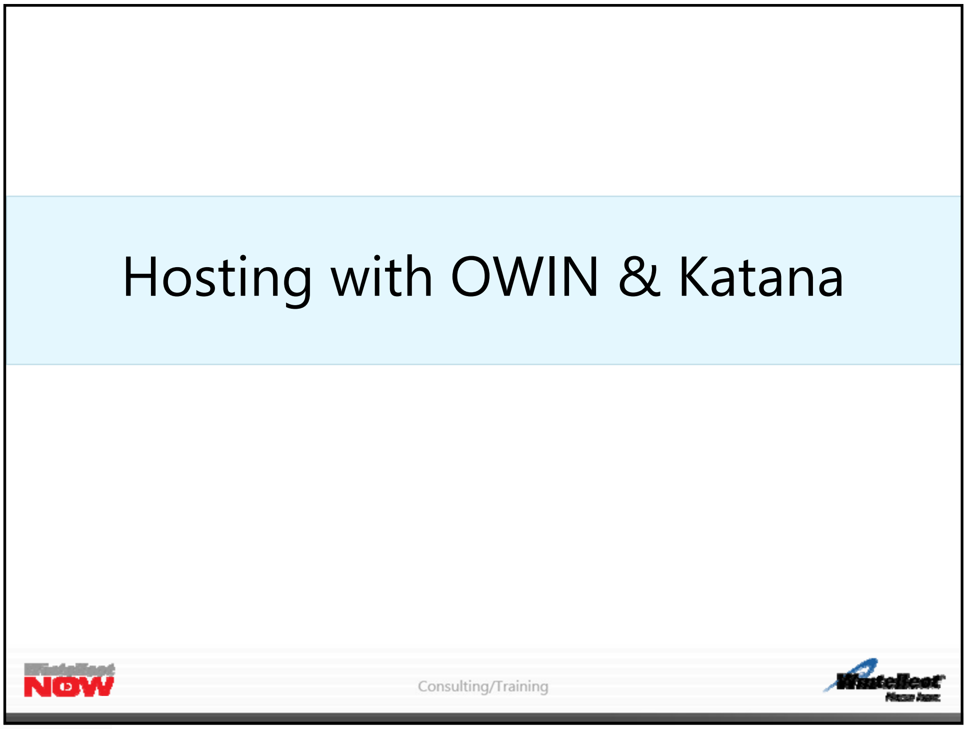


Self Hosting: HttpSelfHostConfiguration

- Drives configuration of **HttpBinding** and WCF channel stack
  - Adds WCF-specific properties to those in the **HttpConfiguration** base class

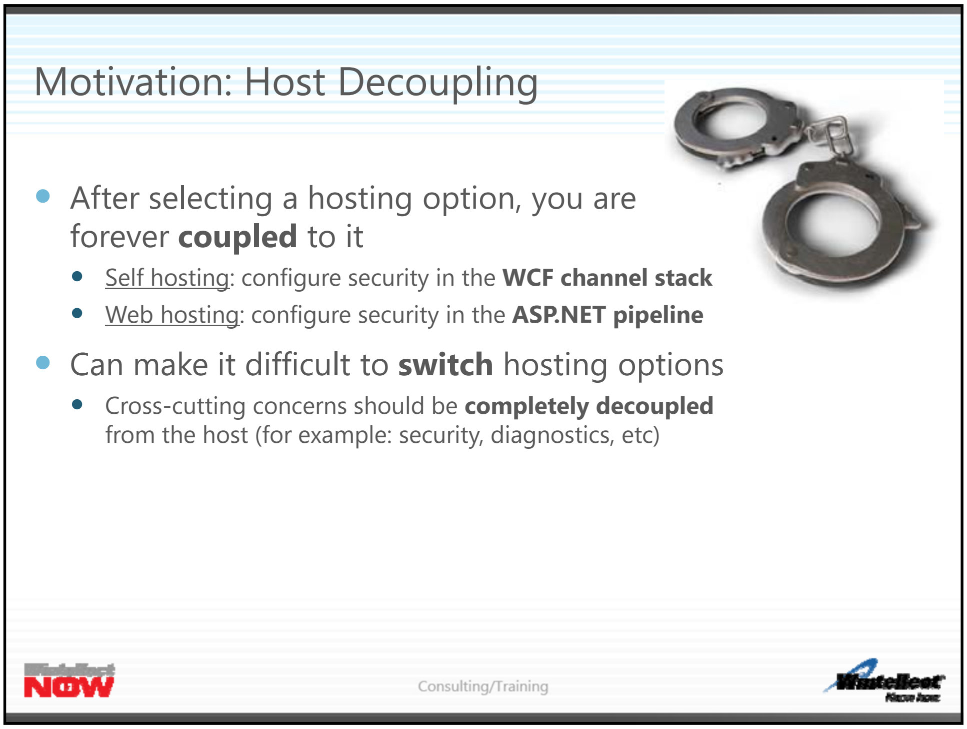
Property	Description
ClientCredentialType	Transport-level credentials which client must supply
MaxBufferSize	Number of bytes in the buffer – default is 64K
MaxConcurrentRequests	Maximum concurrent requests – default is 100
MaxReceivedMessageSize	Maximum size of incoming messages – default 64K
ReceiveTimeout	Message receive timeout – default is 10 minutes
SendTimeout	Message send timeout – default is 1 minute
TransferMode	Buffered, Streamed, Streamed Request / Response

Consulting/Training



Hosting with OWIN & Katana

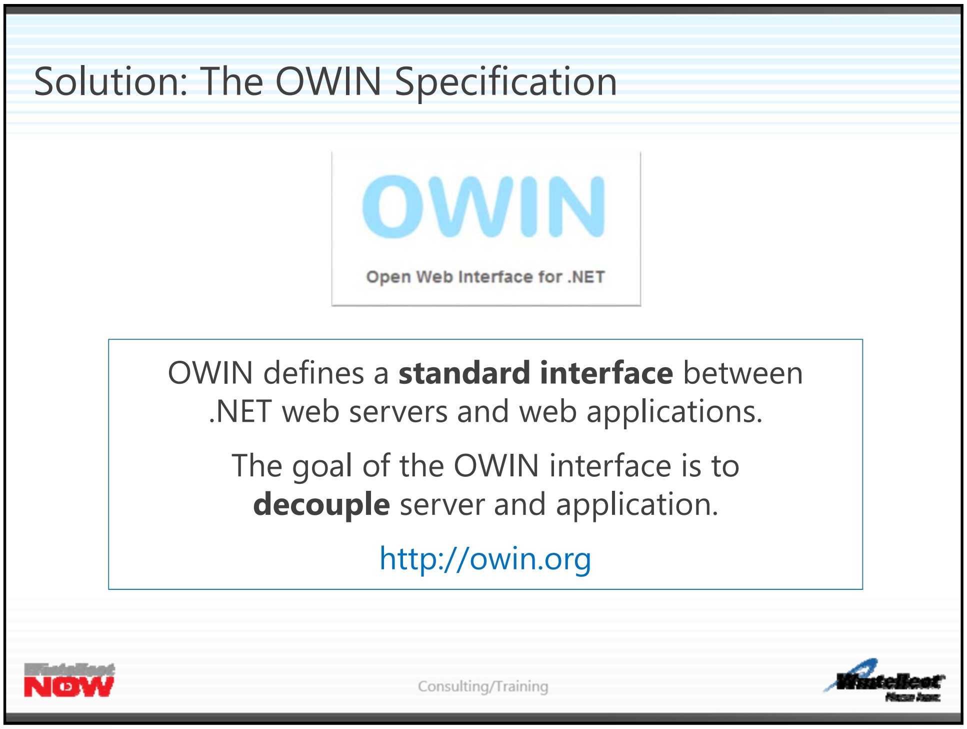
Consulting/Training



Motivation: Host Decoupling

- After selecting a hosting option, you are forever **coupled** to it
  - Self hosting: configure security in the **WCF channel stack**
  - Web hosting: configure security in the **ASP.NET pipeline**
- Can make it difficult to **switch** hosting options
  - Cross-cutting concerns should be **completely decoupled** from the host (for example: security, diagnostics, etc)

Consulting/Training



Solution: The OWIN Specification

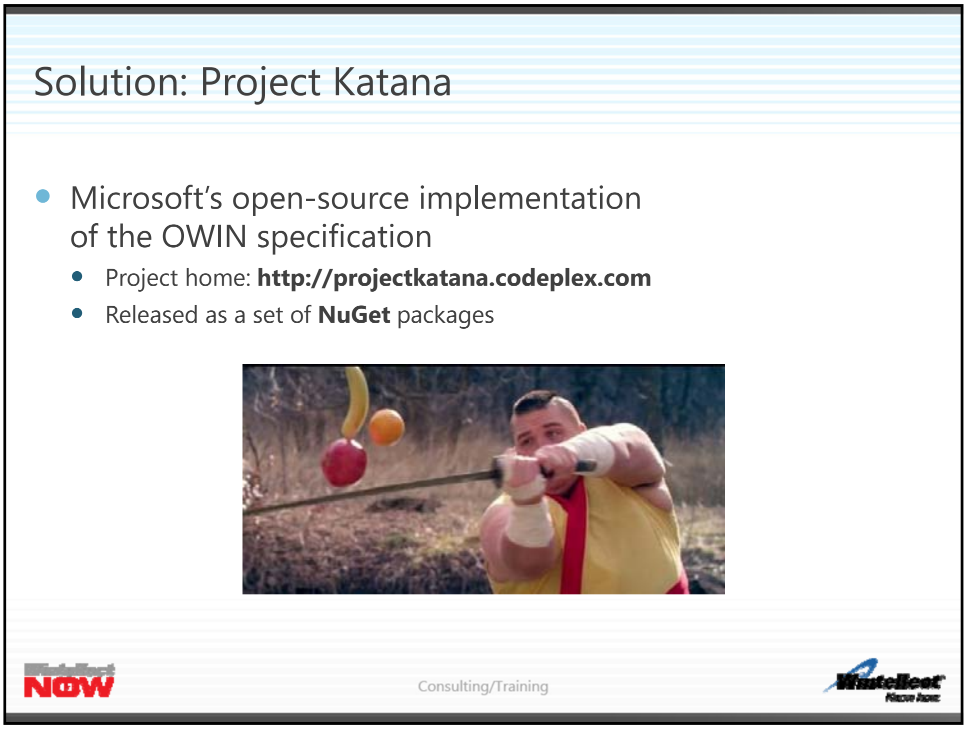
OWIN
Open Web Interface for .NET

OWIN defines a **standard interface** between .NET web servers and web applications.

The goal of the OWIN interface is to **decouple** server and application.

http://owin.org

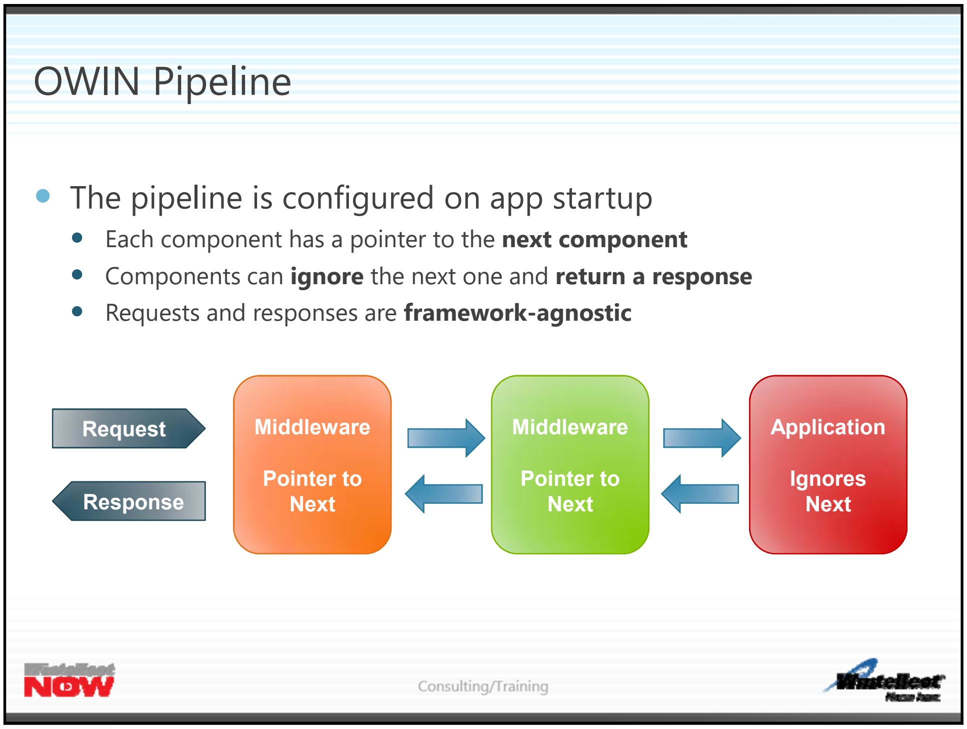
Consulting/Training



Solution: Project Katana

- Microsoft's open-source implementation of the OWIN specification
  - Project home: **http://projectkatana.codeplex.com**
  - Released as a set of **NuGet** packages

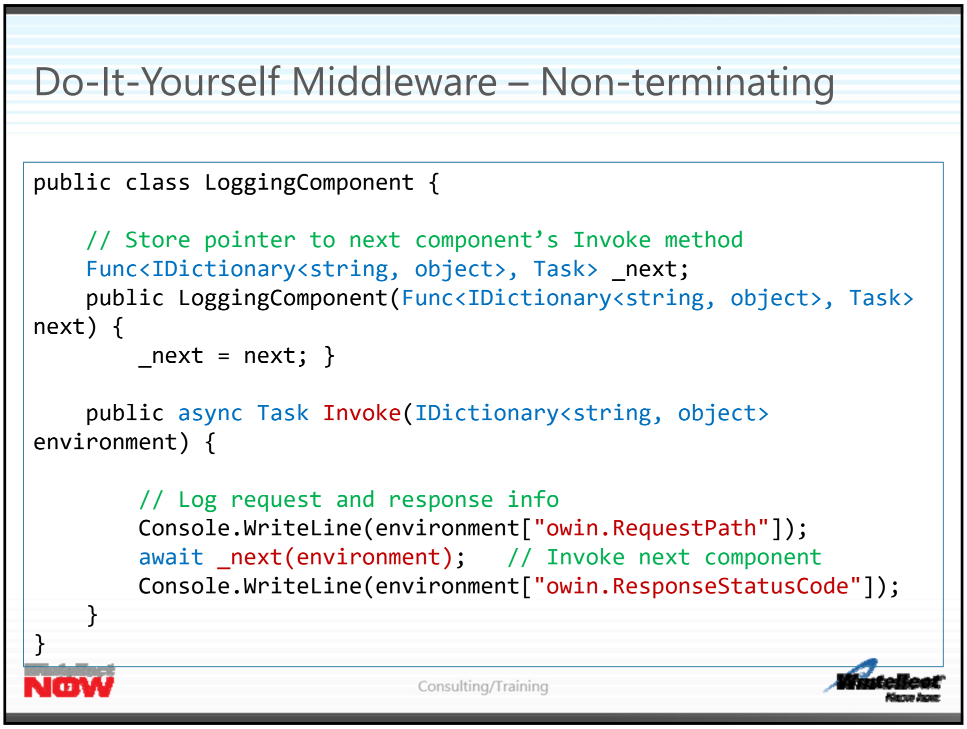
Consulting/Training



OWIN Pipeline

- The pipeline is configured on app startup
  - Each component has a pointer to the **next component**
  - Components can **ignore** the next one and **return a response**
  - Requests and responses are **framework-agnostic**

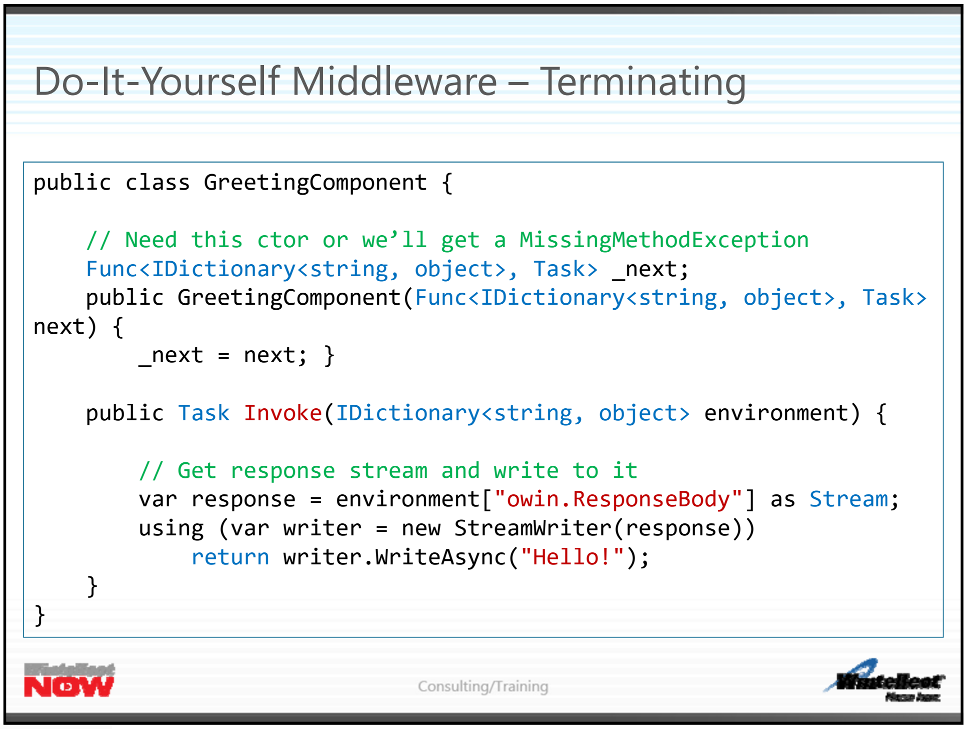
Consulting/Training



Do-It-Yourself Middleware – Non-terminating

```
public class LoggingComponent {

    // Store pointer to next component’s Invoke method
    Func<IDictionary<string, object>, Task> _next;
    public LoggingComponent(Func<IDictionary<string, object>, Task>
next) {
        _next = next; }

    public async Task Invoke(IDictionary<string, object>
environment) {

        // Log request and response info
        Console.WriteLine(environment["owin.RequestPath"]);
        await _next(environment);   // Invoke next component
        Console.WriteLine(environment["owin.ResponseStatusCode"]);
    }
}
```

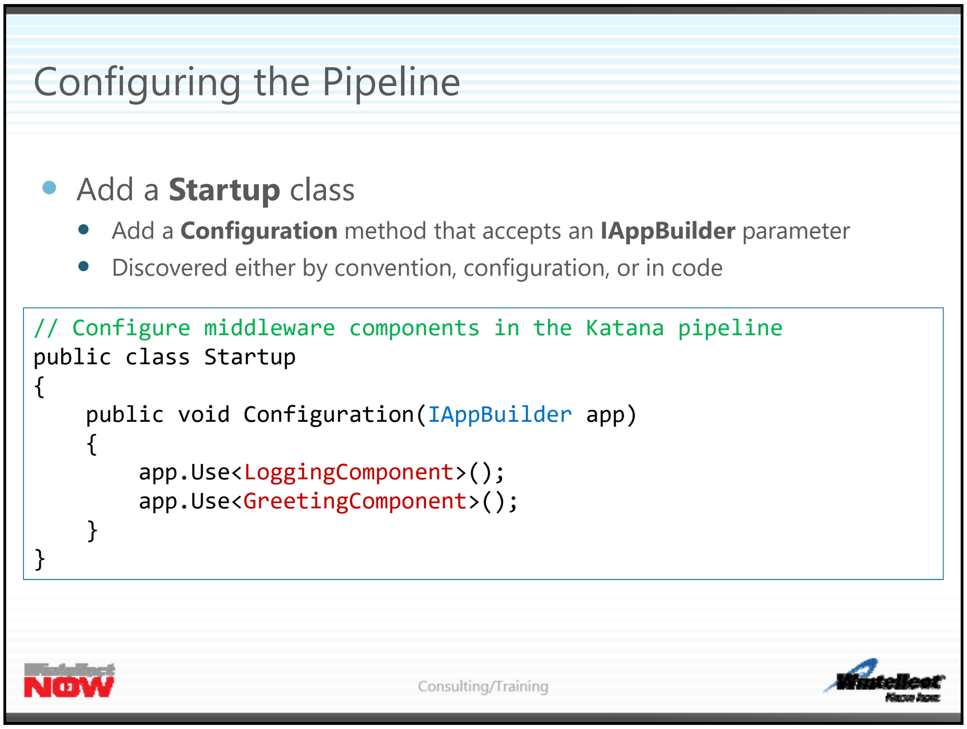
Consulting/Training



Do-It-Yourself Middleware – Terminating

```
public class GreetingComponent {

    // Need this ctor or we’ll get a MissingMethodException
    Func<IDictionary<string, object>, Task> _next;
    public GreetingComponent(Func<IDictionary<string, object>, Task>
next) {
        _next = next; }

    public Task Invoke(IDictionary<string, object> environment) {

        // Get response stream and write to it
        var response = environment["owin.ResponseBody"] as Stream;
        using (var writer = new StreamWriter(response))
            return writer.WriteAsync("Hello!");
    }
}
```

Consulting/Training

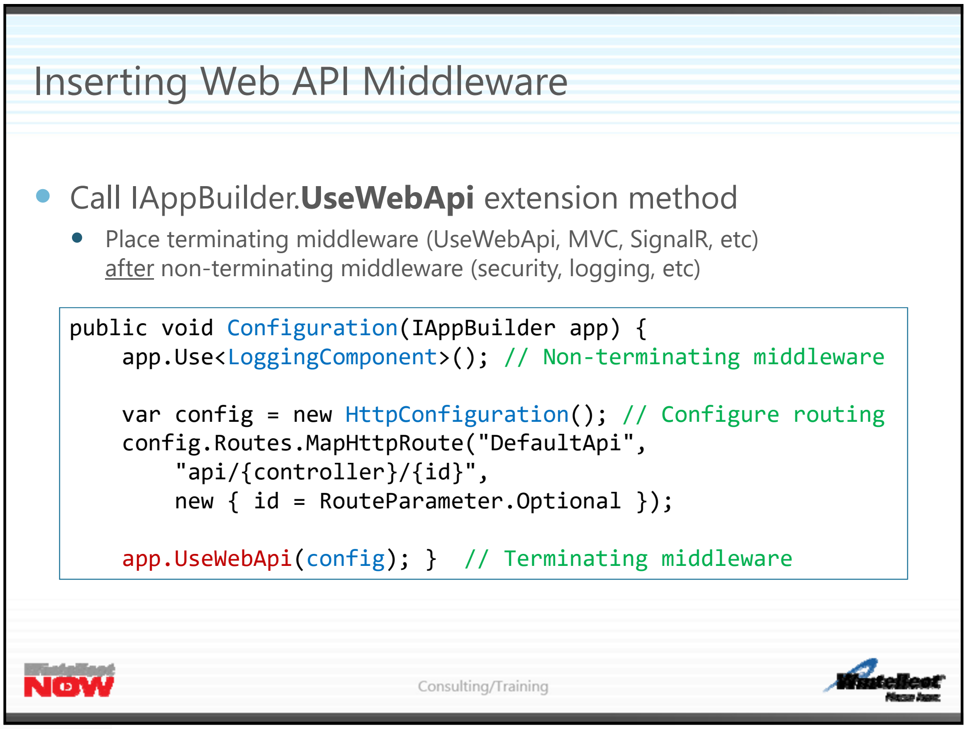


Configuring the Pipeline

- Add a **Startup** class
  - Add a **Configuration** method that accepts an **IAppBuilder** parameter
  - Discovered either by convention, configuration, or in code

```
// Configure middleware components in the Katana pipeline
public class Startup
{
    public void Configuration(IAppBuilder app)
    {
        app.Use<LoggingComponent>();
        app.Use<GreetingComponent>();
    }
}
```

Consulting/Training



Inserting Web API Middleware

- Call IAppBuilder.**UseWebApi** extension method
  - Place terminating middleware (UseWebApi, MVC, SignalR, etc) after non-terminating middleware (security, logging, etc)

```
public void Configuration(IAppBuilder app) {
    app.Use<LoggingComponent>(); // Non-terminating middleware

    var config = new HttpConfiguration(); // Configure routing
    config.Routes.MapHttpRoute("DefaultApi",
        "api/{controller}/{id}",
        new { id = RouteParameter.Optional });

    app.UseWebApi(config); }  // Terminating middleware
```

Consulting/Training

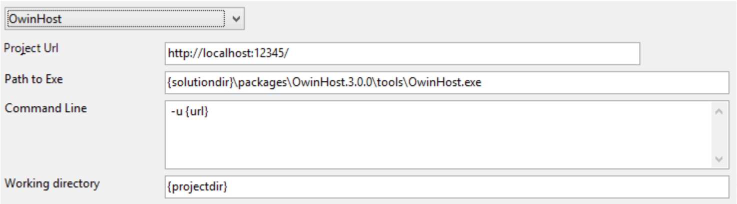
Self Hosting with Katana

- Create a .NET application – Console, WPF, NT Service, etc
  - Add **Microsoft.Owin.SelfHost** NuGet package
  - Add **Microsoft.AspNet.WebApi.OwinSelfHost** NuGet package
- In Main, call **WebApp.Start<Startup>**
  - Pass a **URI** for the application base address
  - Call **Dispose** to clean up when app shuts down (can place in **using** block)

```
static void Main(string[] args) {
    using (WebApp.Start<Startup>("http://localhost:12345/")) {
        Console.WriteLine("Service is running ...");
        Console.ReadLine();
    } }
```

Hosting with OwinHost.exe

- Create an "Empty" Web project in Visual Studio 2013
  - Add **Microsoft.AspNet.WebApi.Owin** NuGet package
  - Add startup and controller classes – there's an "**OWIN Startup class**" template
  - Add **OwinHost** NuGet package
    - On the **Web** tab of the project properties page, select **OwinHost** as the web server
  - **Press F5**, and you're hosting a Web API service *without IIS or ASP.NET!*



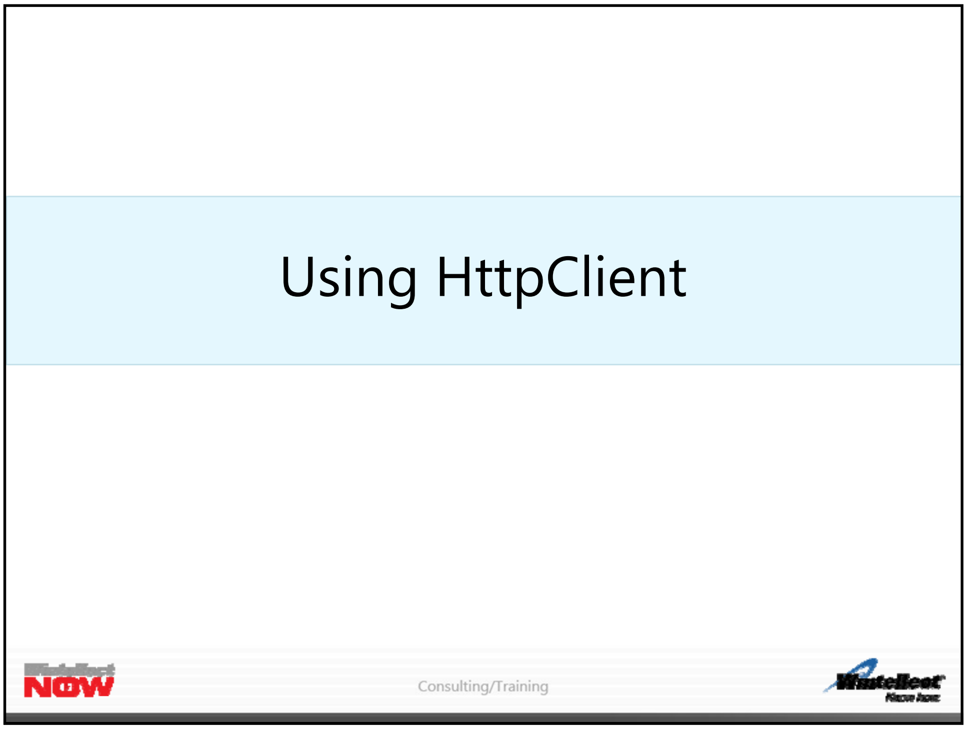
OwinHost

Project Url: http://localhost:12345/

Path to Exe: {solutiondir}\packages\OwinHost.3.0.0\tools\OwinHost.exe

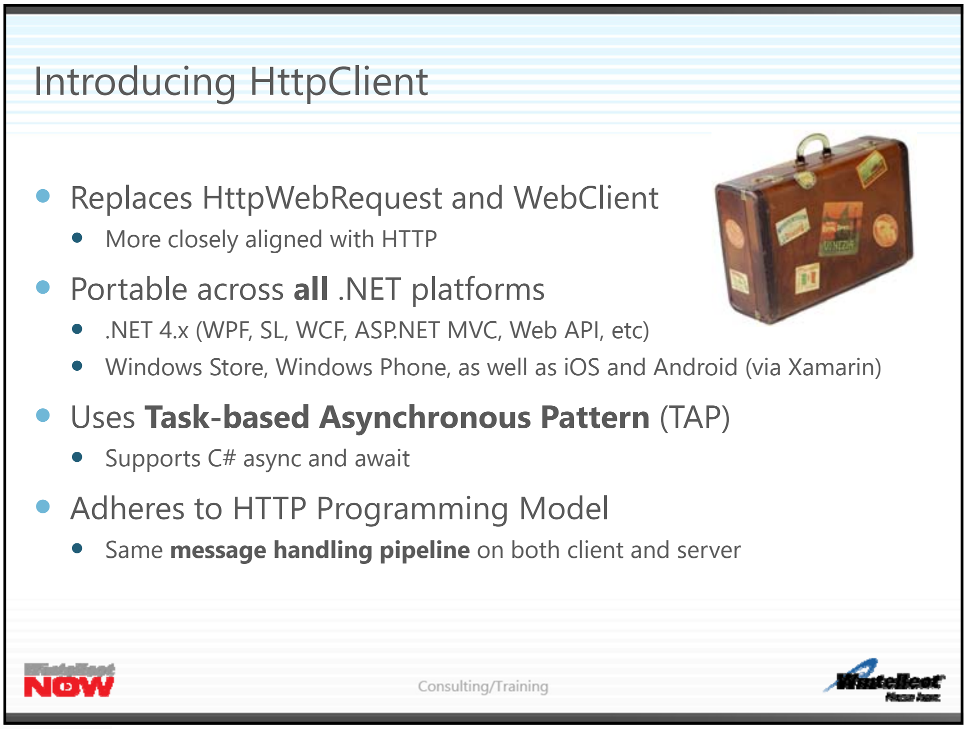
Command Line: -u {url}

Working directory: {projectdir}



Using HttpClient

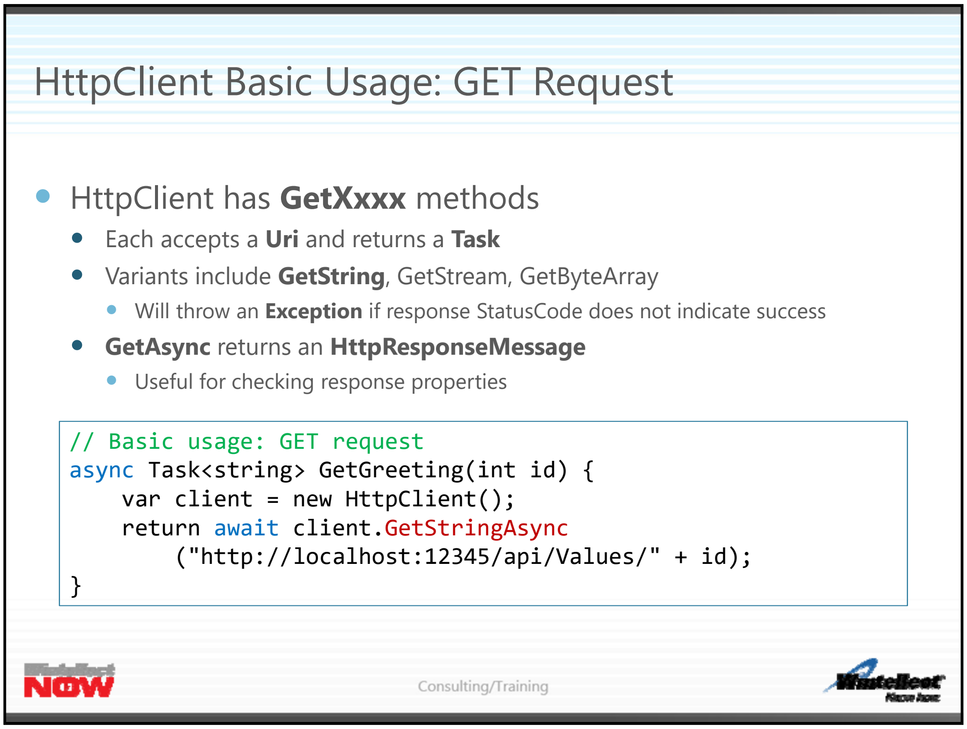
Consulting/Training



Introducing HttpClient

- Replaces HttpWebRequest and WebClient
  - More closely aligned with HTTP
- Portable across **all** .NET platforms
  - .NET 4.x (WPF, SL, WCF, ASP.NET MVC, Web API, etc)
  - Windows Store, Windows Phone, as well as iOS and Android (via Xamarin)
- Uses **Task-based Asynchronous Pattern** (TAP)
  - Supports C# async and await
- Adheres to HTTP Programming Model
  - Same **message handling pipeline** on both client and server

Consulting/Training

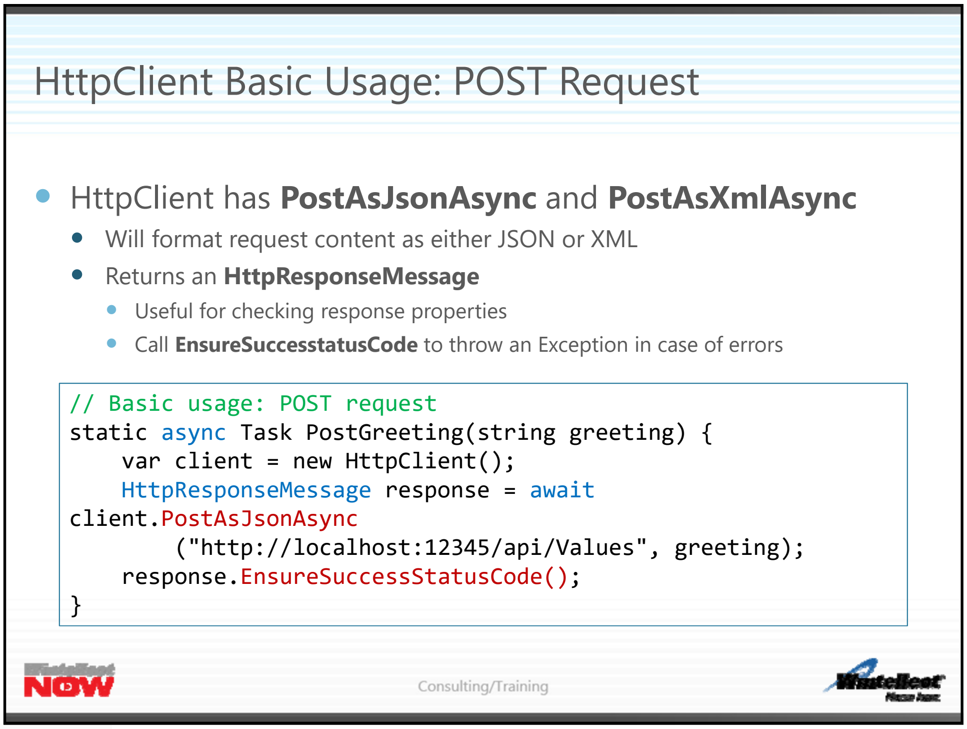


HttpClient Basic Usage: GET Request

- HttpClient has **GetXxxx** methods
  - Each accepts a **Uri** and returns a **Task**
  - Variants include **GetString**, GetStream, GetByteArray
    - Will throw an **Exception** if response StatusCode does not indicate success
  - **GetAsync** returns an **HttpResponseMessage**
    - Useful for checking response properties

```
// Basic usage: GET request
async Task<string> GetGreeting(int id) {
    var client = new HttpClient();
    return await client.GetStringAsync
        ("http://localhost:12345/api/Values/" + id);
}
```

Consulting/Training

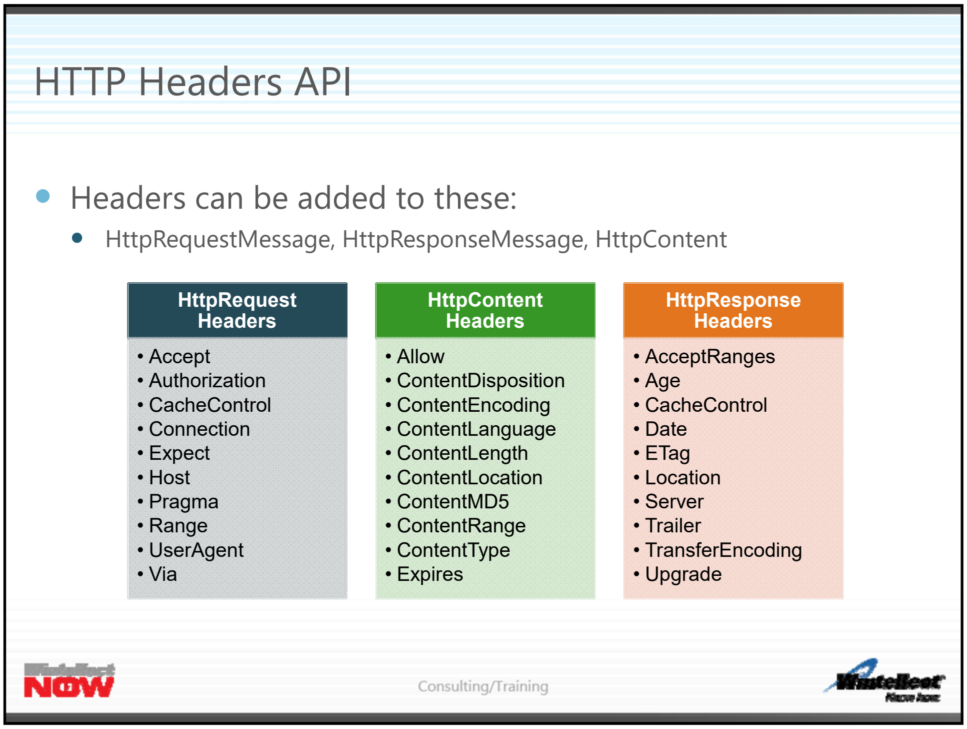


HttpClient Basic Usage: POST Request

- HttpClient has **PostAsJsonAsync** and **PostAsXmlAsync**
  - Will format request content as either JSON or XML
  - Returns an **HttpResponseMessage**
    - Useful for checking response properties
    - Call **EnsureSuccesstatusCode** to throw an Exception in case of errors

```
// Basic usage: POST request
static async Task PostGreeting(string greeting) {
    var client = new HttpClient();
    HttpResponseMessage response = await
client.PostAsJsonAsync
        ("http://localhost:12345/api/Values", greeting);
    response.EnsureSuccessStatusCode();
}
```

Consulting/Training

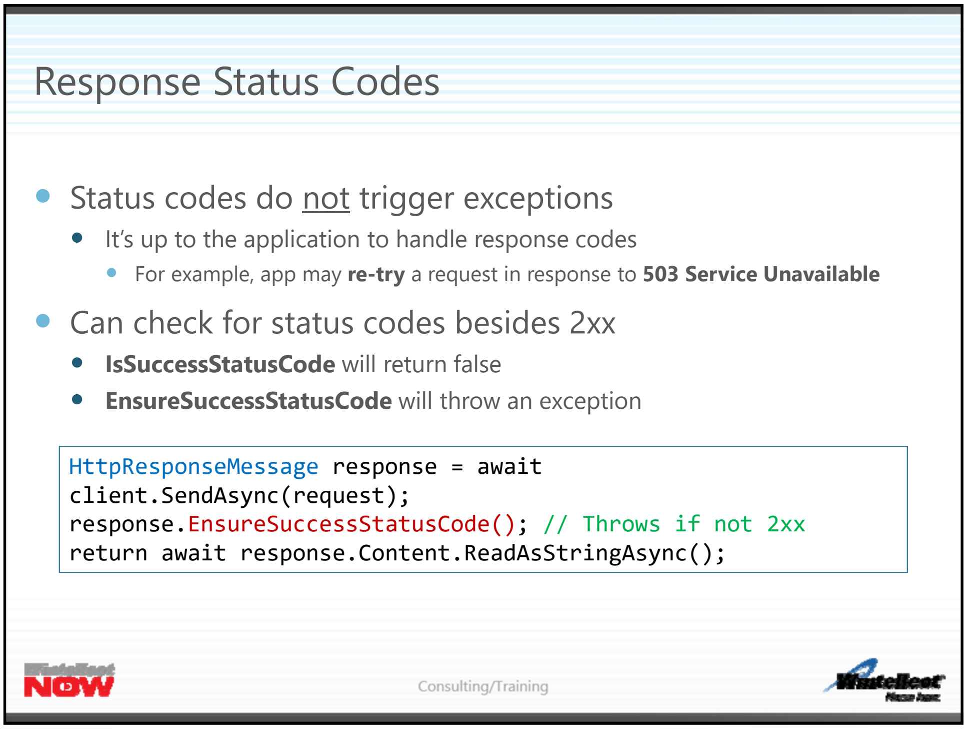


HTTP Headers API

- Headers can be added to these:
  - HttpRequestMessage, HttpResponseMessage, HttpContent

HttpRequest Headers	HttpContent Headers	HttpResponse Headers
• Accept	• Allow	• AcceptRanges
• Authorization	• ContentDisposition	• Age
• CacheControl	• ContentEncoding	• CacheControl
• Connection	• ContentLanguage	• Date
• Expect	• ContentLength	• ETag
• Host	• ContentLocation	• Location
• Pragma	• ContentMD5	• Server
• Range	• ContentRange	• Trailer
• UserAgent	• ContentType	• TransferEncoding
• Via	• Expires	• Upgrade

Consulting/Training

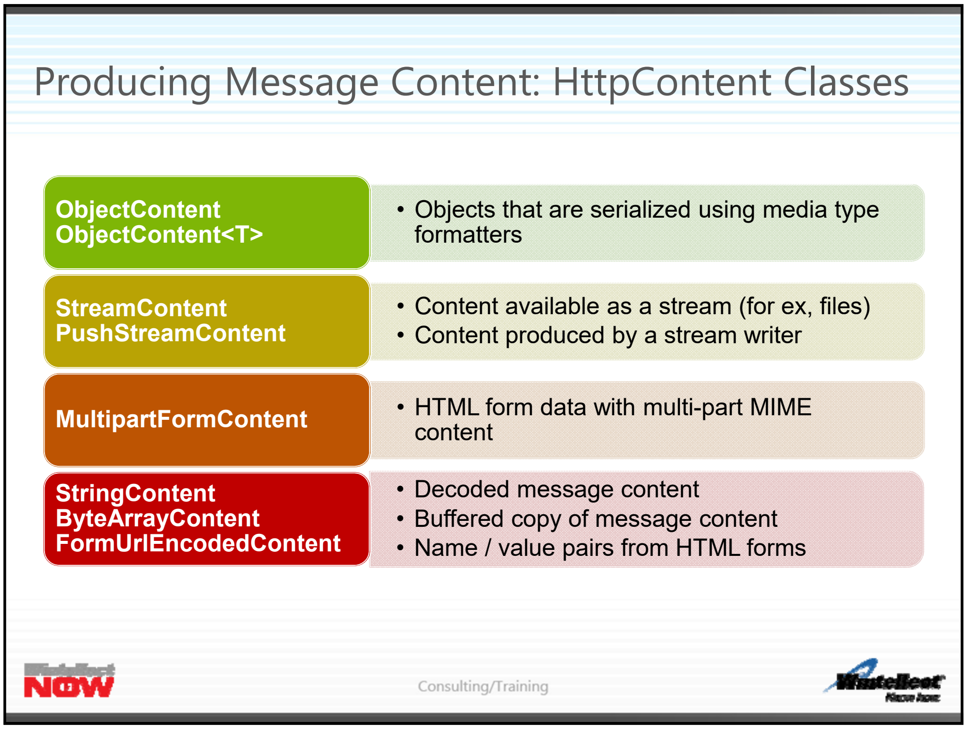


Response Status Codes

- Status codes do not trigger exceptions
  - It's up to the application to handle response codes
    - For example, app may **re-try** a request in response to **503 Service Unavailable**
- Can check for status codes besides 2xx
  - **IsSuccessStatusCode** will return false
  - **EnsureSuccessStatusCode** will throw an exception

```
HttpResponseMessage response = await
client.SendAsync(request);
response.EnsureSuccessStatusCode(); // Throws if not 2xx
return await response.Content.ReadAsStringAsync();
```

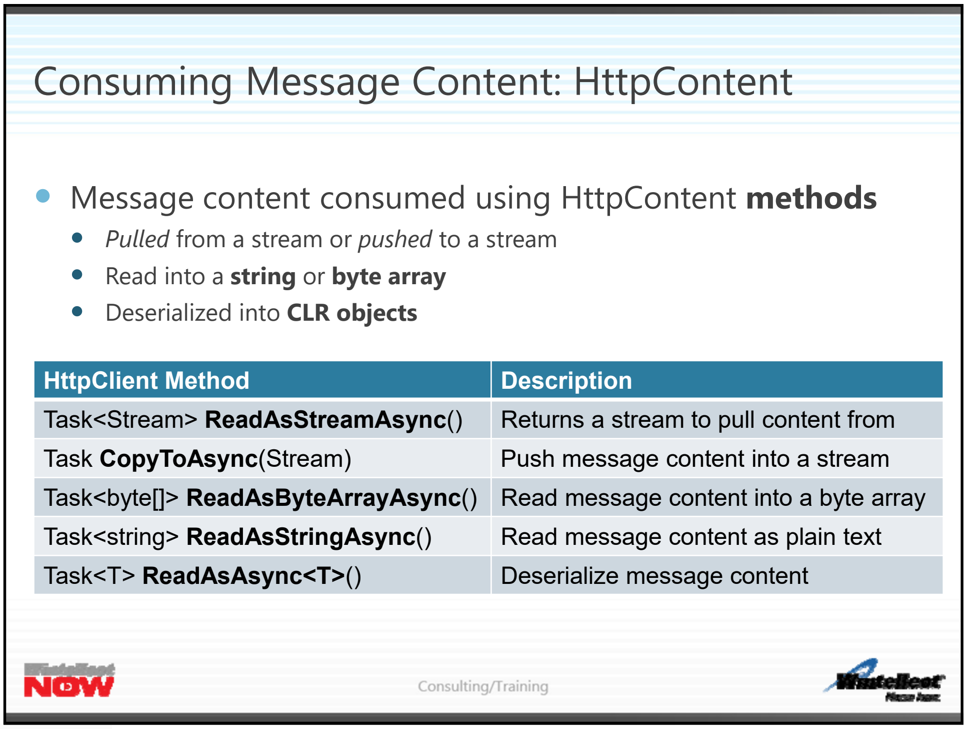
Consulting/Training



Producing Message Content: HttpContent Classes

ObjectContent **ObjectContent<T>**	- Objects that are serialized using media type formatters
StreamContent **PushStreamContent**	- Content available as a stream (for ex, files) - Content produced by a stream writer
MultipartFormContent	- HTML form data with multi-part MIME content
StringContent **ByteArrayContent** **FormUrlEncodedContent**	- Decoded message content - Buffered copy of message content - Name / value pairs from HTML forms

Consulting/Training

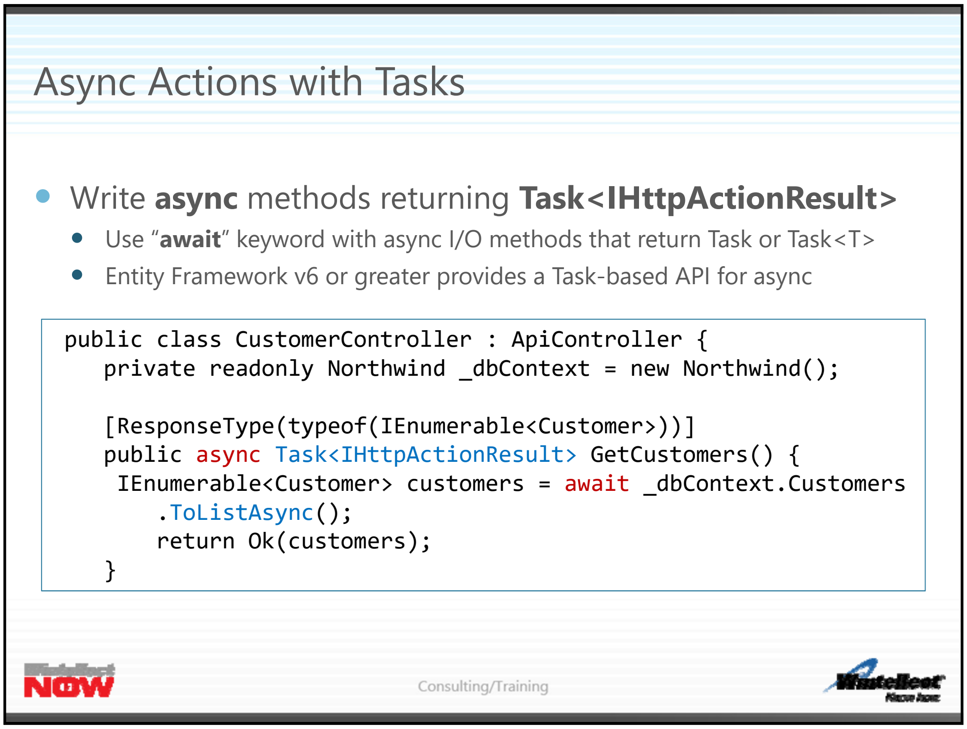


Consuming Message Content: HttpContent

- Message content consumed using HttpContent **methods**
  - *Pulled* from a stream or *pushed* to a stream
  - Read into a **string** or **byte array**
  - Deserialized into **CLR objects**

HttpClient Method	Description
Task<Stream> **ReadAsStreamAsync**()	Returns a stream to pull content from
Task **CopyToAsync**(Stream)	Push message content into a stream
Task<byte[]> **ReadAsByteArrayAsync**()	Read message content into a byte array
Task<string> **ReadAsStringAsync**()	Read message content as plain text
Task<T> **ReadAsAsync<T>**()	Deserialize message content

Consulting/Training



Async Actions with Tasks

- Write **async** methods returning **Task<IHttpActionResult>**
  - Use “**await**” keyword with async I/O methods that return Task or Task<T>
  - Entity Framework v6 or greater provides a Task-based API for async

```
public class CustomerController : ApiController {
   private readonly Northwind _dbContext = new Northwind();

   [ResponseType(typeof(IEnumerable<Customer>))]
   public async Task<IHttpActionResult> GetCustomers() {
    IEnumerable<Customer> customers = await _dbContext.Customers
       .ToListAsync();
       return Ok(customers);
   }
```

Consulting/Training

QUESTIONS

Consulting/Training

ASP.NET Web API Hosting, Client, Async

Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

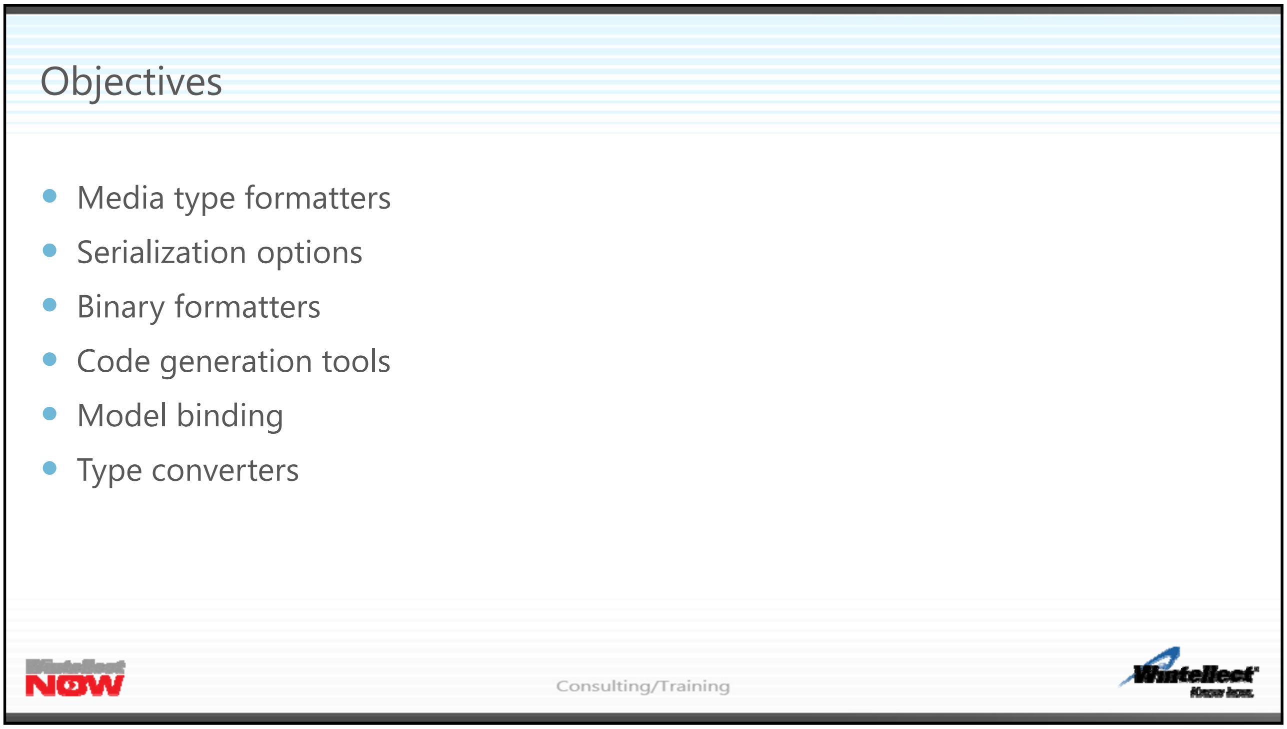
Consulting/Training

Wintellect NOW

Serialization and Model Binding

Tony Sneed
tsneed@wintellect.com
http://blog.tonysneed.com

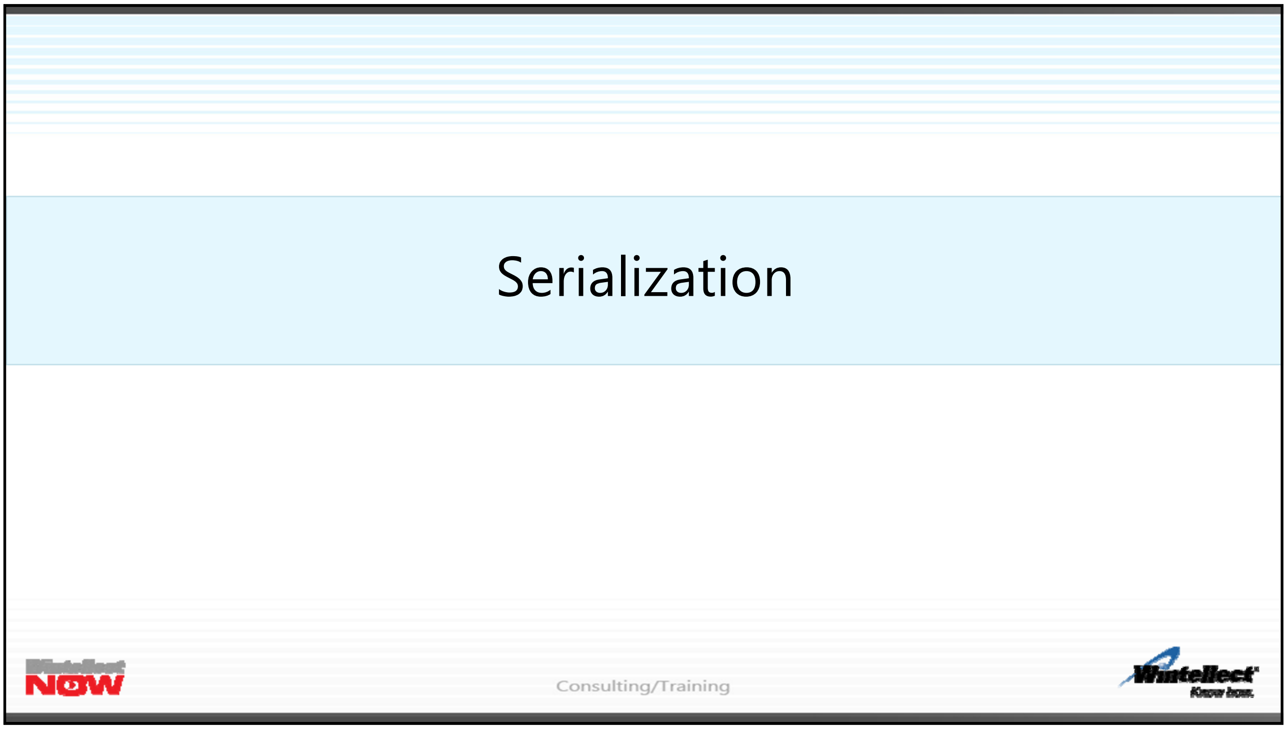
Consulting/Training



Objectives

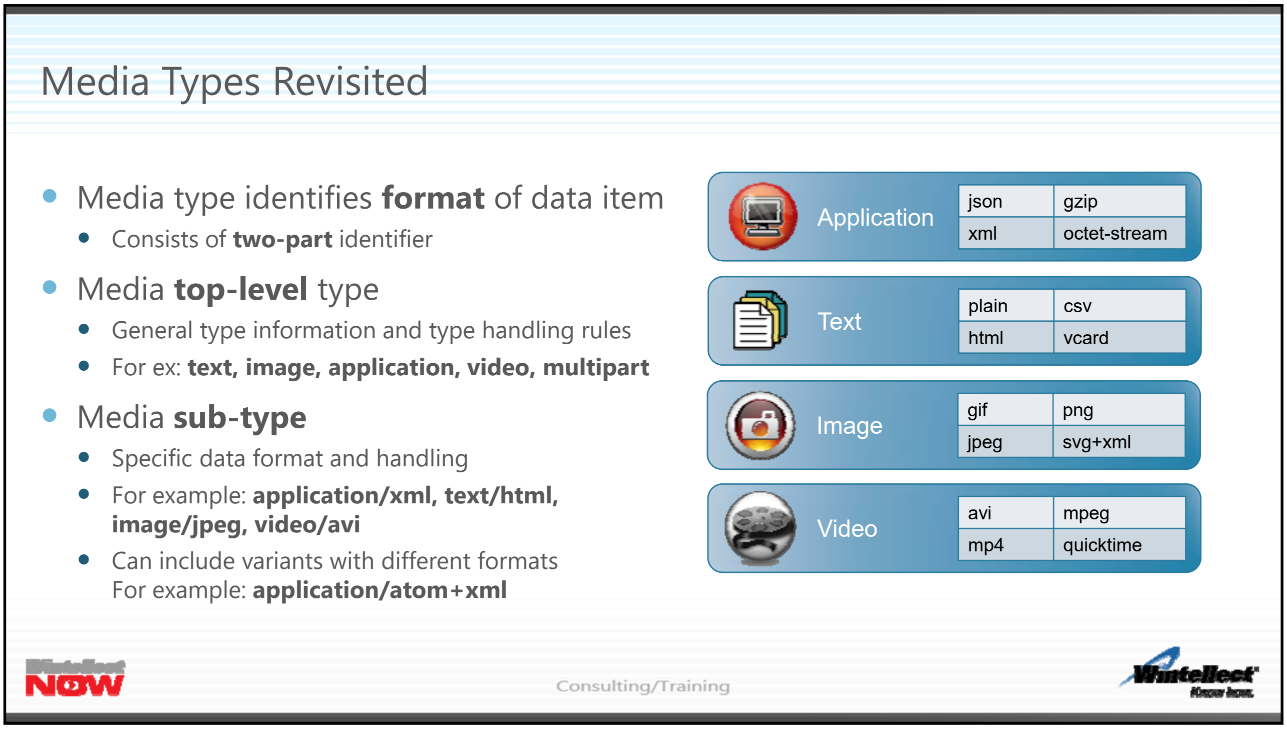
- Media type formatters
- Serialization options
- Binary formatters
- Code generation tools
- Model binding
- Type converters

Consulting/Training



Serialization

Consulting/Training

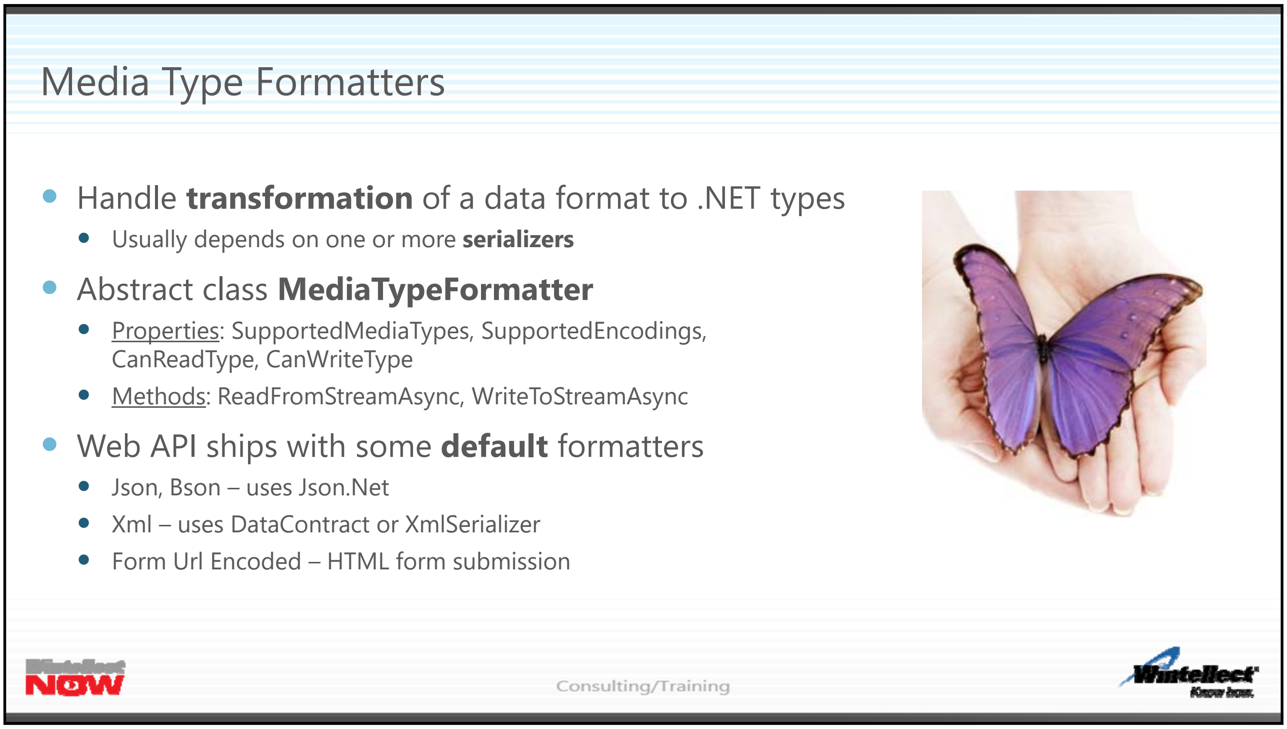


Media Types Revisited

- Media type identifies **format** of data item
  - Consists of **two-part** identifier
- Media **top-level** type
  - General type information and type handling rules
  - For ex: **text, image, application, video, multipart**
- Media **sub-type**
  - Specific data format and handling
  - For example: **application/xml, text/html, image/jpeg, video/avi**
  - Can include variants with different formats For example: **application/atom+xml**

Application	json	gzip
	xml	octet-stream
Text	plain	csv
	html	vcard
Image	gif	png
	jpeg	svg+xml
Video	avi	mpeg
	mp4	quicktime

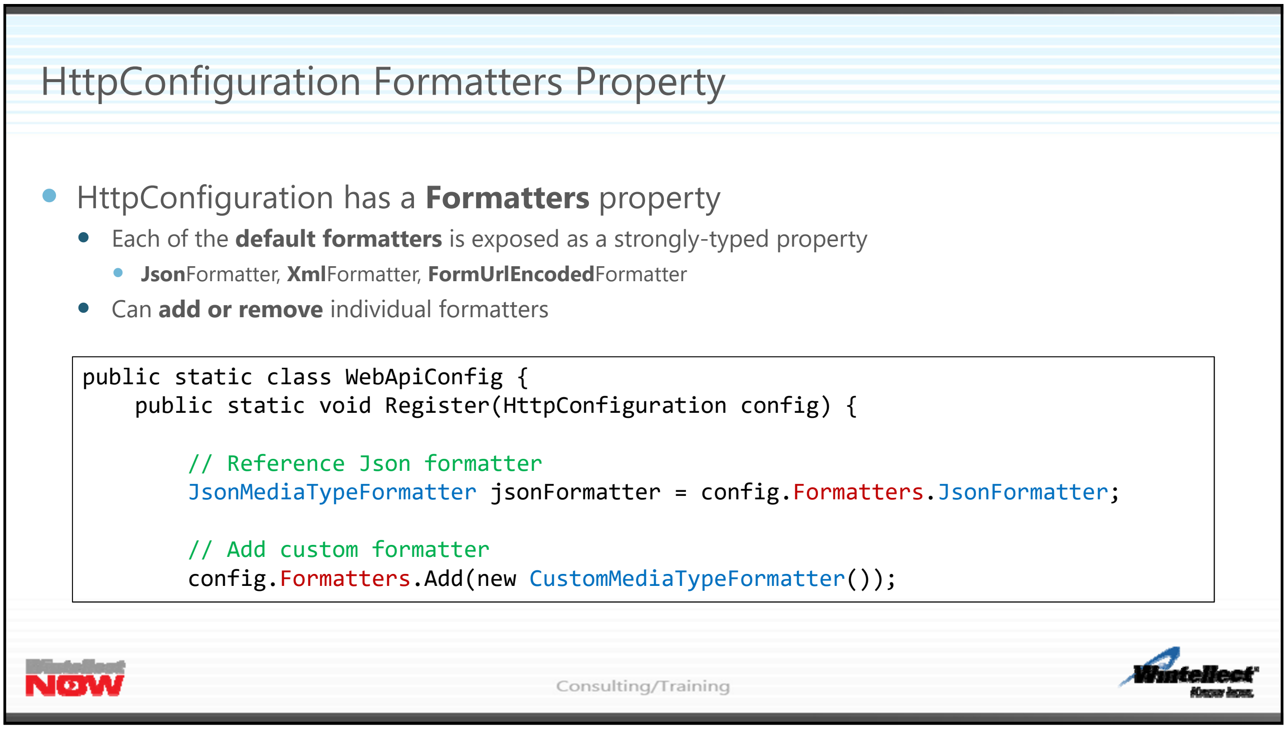
Consulting/Training



Media Type Formatters

- Handle **transformation** of a data format to .NET types
  - Usually depends on one or more **serializers**
- Abstract class **MediaTypeFormatter**
  - Properties: SupportedMediaTypes, SupportedEncodings, CanReadType, CanWriteType
  - Methods: ReadFromStreamAsync, WriteToStreamAsync
- Web API ships with some **default** formatters
  - Json, Bson – uses Json.Net
  - Xml – uses DataContract or XmlSerializer
  - Form Url Encoded – HTML form submission

Consulting/Training

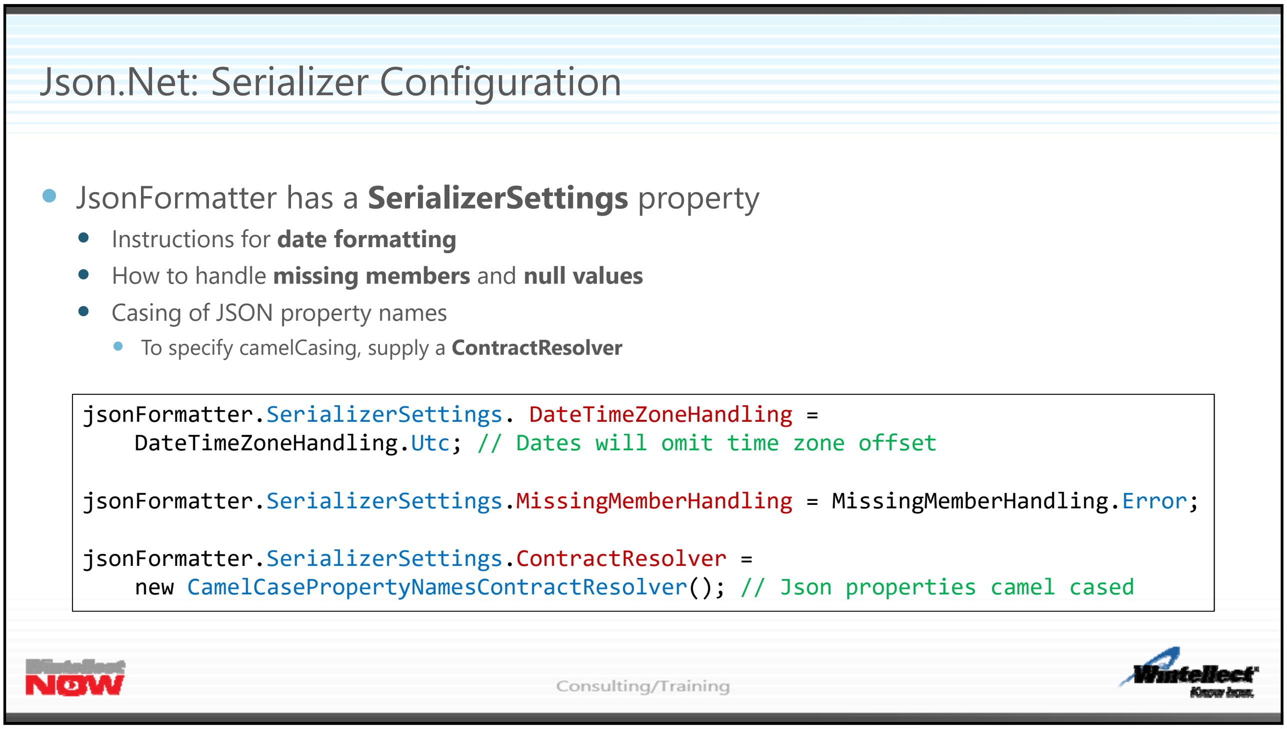


HttpConfiguration Formatters Property

- HttpConfiguration has a **Formatters** property
  - Each of the **default formatters** is exposed as a strongly-typed property
    - **Json**Formatter, **Xml**Formatter, **FormUrlEncoded**Formatter
  - Can **add or remove** individual formatters

```
public static class WebApiConfig {
    public static void Register(HttpConfiguration config) {

        // Reference Json formatter
        JsonMediaTypeFormatter jsonFormatter = config.Formatters.JsonFormatter;

        // Add custom formatter
        config.Formatters.Add(new CustomMediaTypeFormatter());
```

Consulting/Training

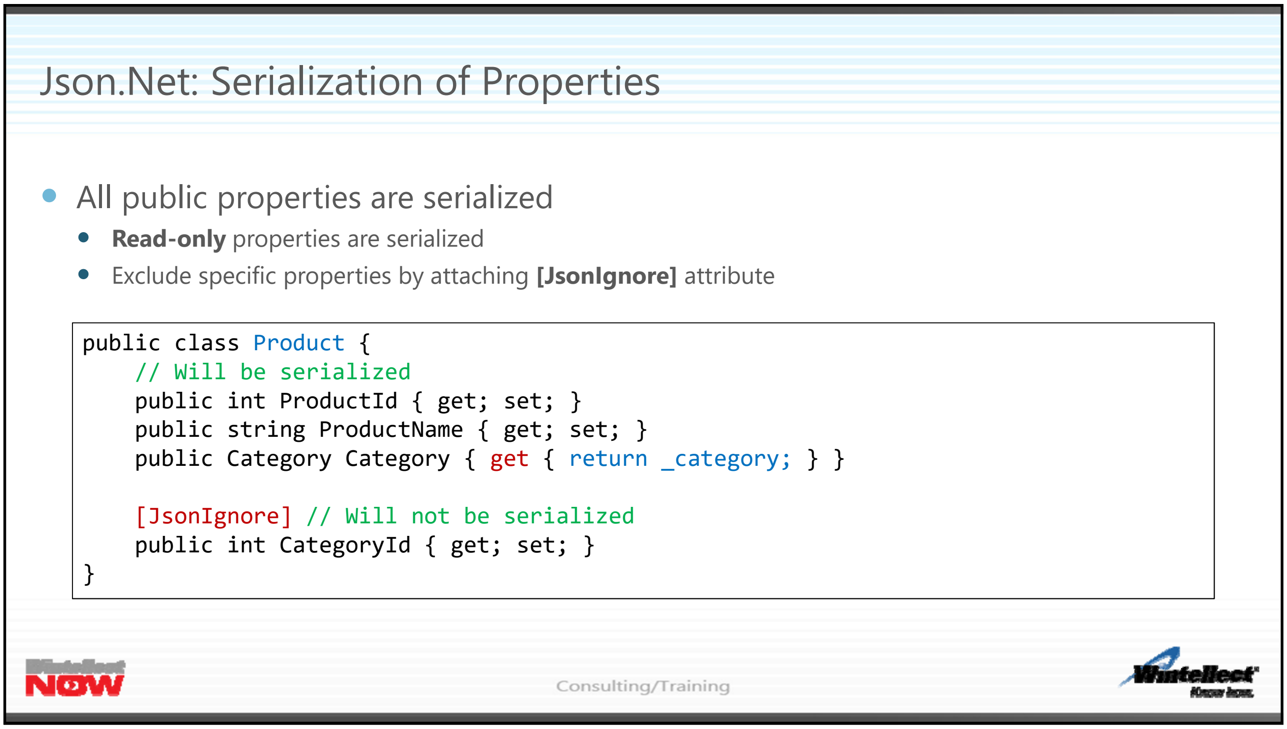


Json.Net: Serializer Configuration

- JsonFormatter has a **SerializerSettings** property
  - Instructions for **date formatting**
  - How to handle **missing members** and **null values**
  - Casing of JSON property names
    - To specify camelCasing, supply a **ContractResolver**

```
jsonFormatter.SerializerSettings. DateTimeZoneHandling =
    DateTimeZoneHandling.Utc; // Dates will omit time zone offset

jsonFormatter.SerializerSettings.MissingMemberHandling = MissingMemberHandling.Error;

jsonFormatter.SerializerSettings.ContractResolver =
    new CamelCasePropertyNamesContractResolver(); // Json properties camel cased
```

Consulting/Training

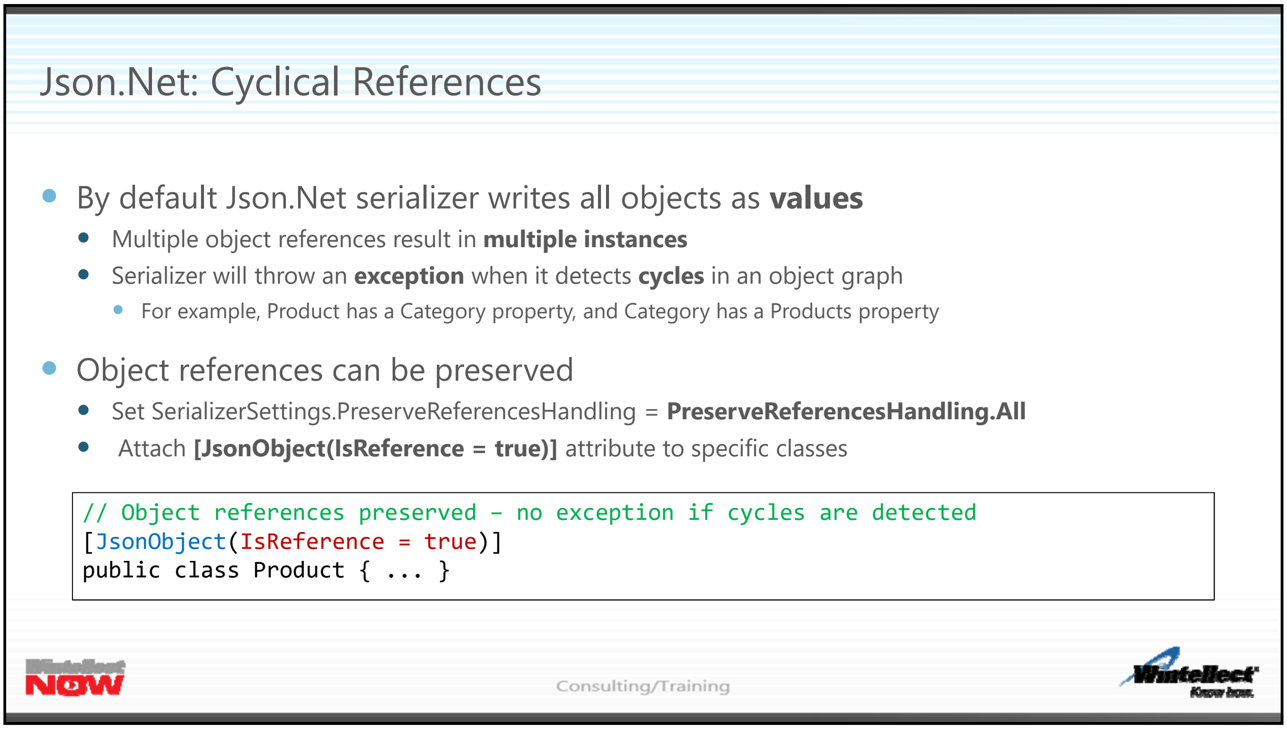


Json.Net: Serialization of Properties

- All public properties are serialized
  - **Read-only** properties are serialized
  - Exclude specific properties by attaching **[JsonIgnore]** attribute

```
public class Product {
    // Will be serialized
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public Category Category { get { return _category; } }

    [JsonIgnore] // Will not be serialized
    public int CategoryId { get; set; }
}
```

Consulting/Training

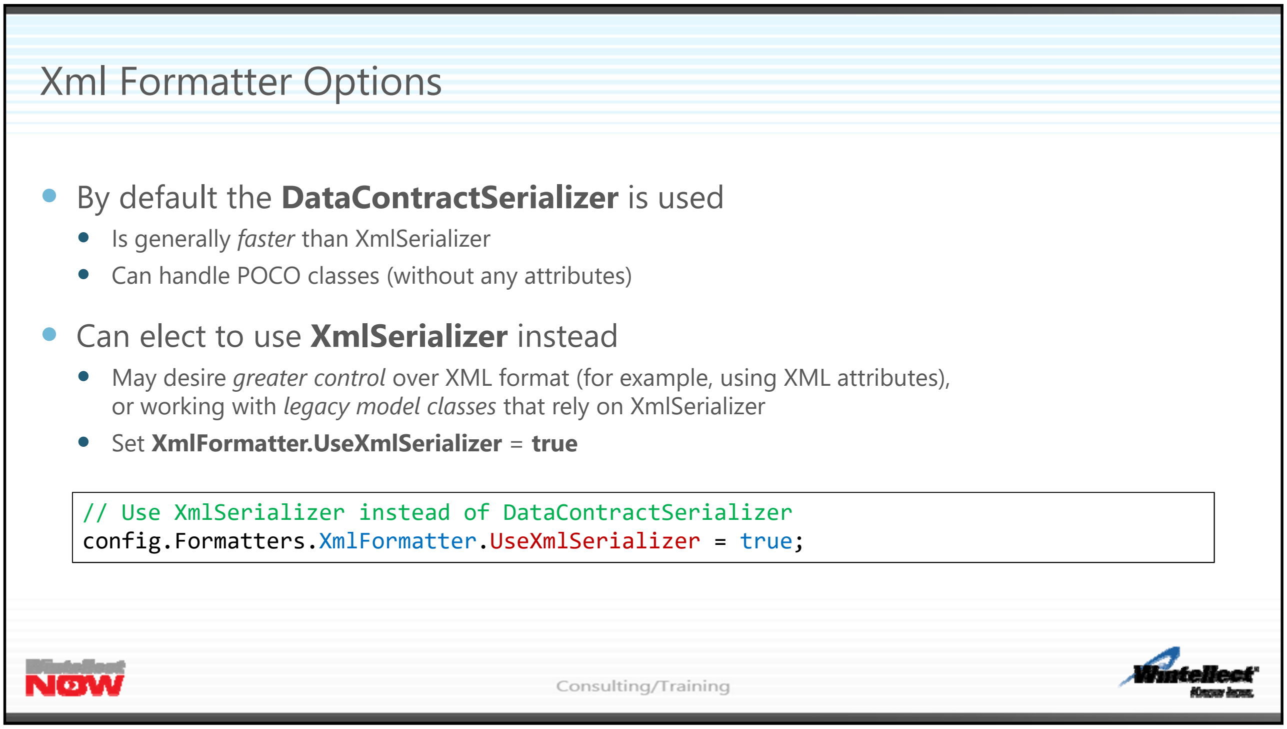


Json.Net: Cyclical References

- By default Json.Net serializer writes all objects as **values**
  - Multiple object references result in **multiple instances**
  - Serializer will throw an **exception** when it detects **cycles** in an object graph
    - For example, Product has a Category property, and Category has a Products property
- Object references can be preserved
  - Set SerializerSettings.PreserveReferencesHandling = **PreserveReferencesHandling.All**
  - Attach **[JsonObject(IsReference = true)]** attribute to specific classes

```
// Object references preserved – no exception if cycles are detected
[JsonObject(IsReference = true)]
public class Product { ... }
```

Consulting/Training

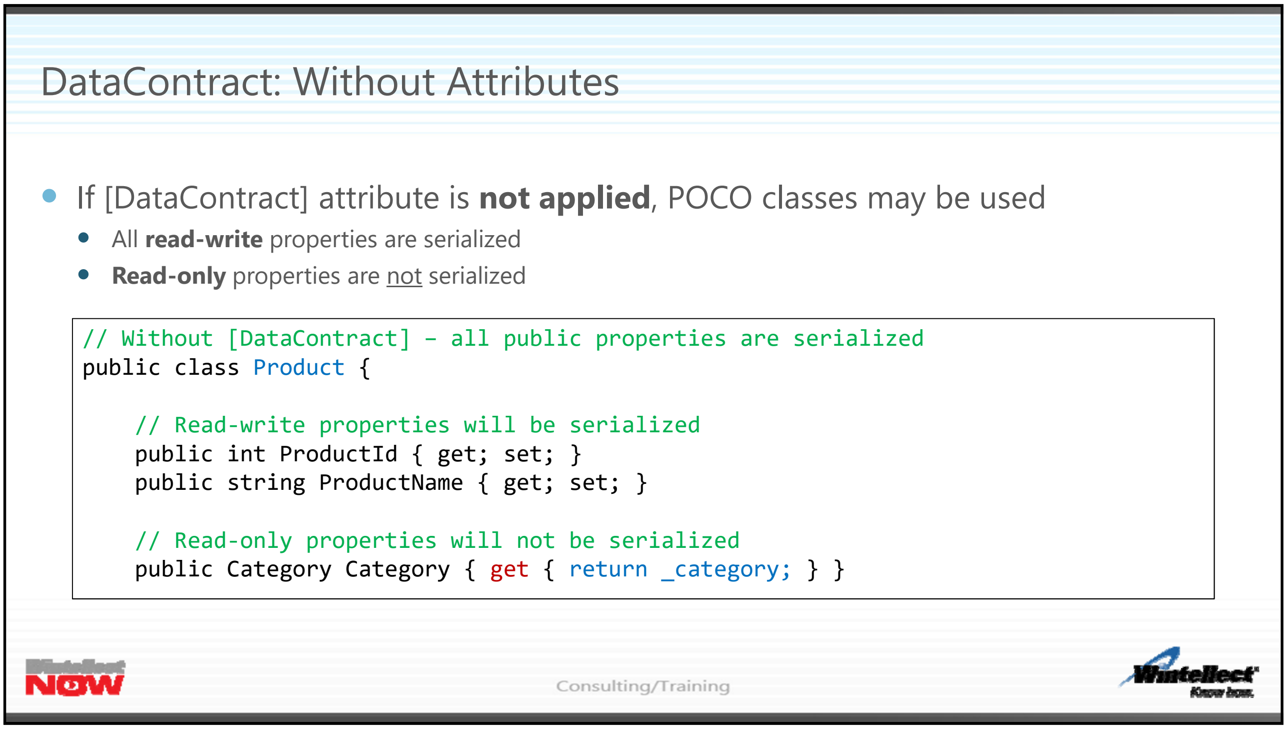


Xml Formatter Options

- By default the **DataContractSerializer** is used
  - Is generally *faster* than XmlSerializer
  - Can handle POCO classes (without any attributes)
- Can elect to use **XmlSerializer** instead
  - May desire *greater control* over XML format (for example, using XML attributes), or working with *legacy model classes* that rely on XmlSerializer
  - Set **XmlFormatter.UseXmlSerializer** = **true**

```
// Use XmlSerializer instead of DataContractSerializer
config.Formatters.XmlFormatter.UseXmlSerializer = true;
```

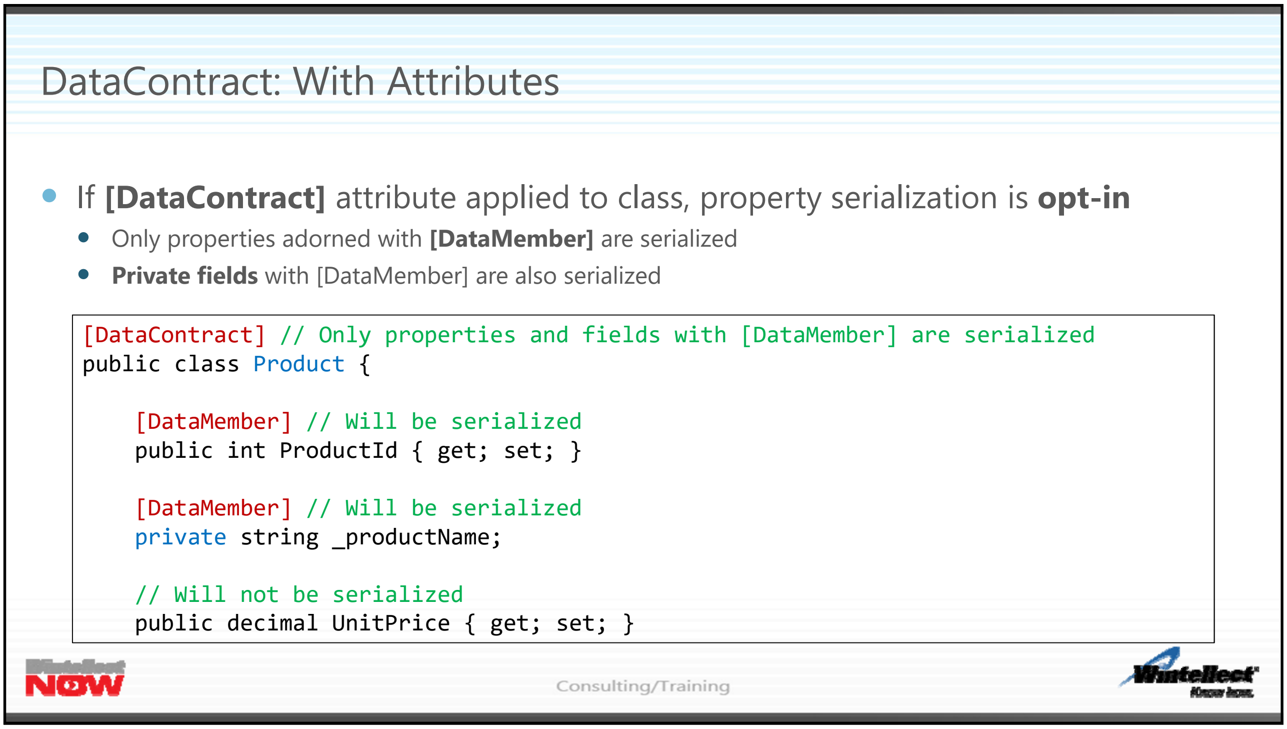
Consulting/Training



DataContract: Without Attributes

- If [DataContract] attribute is **not applied**, POCO classes may be used
  - All **read-write** properties are serialized
  - **Read-only** properties are <u>not</u> serialized

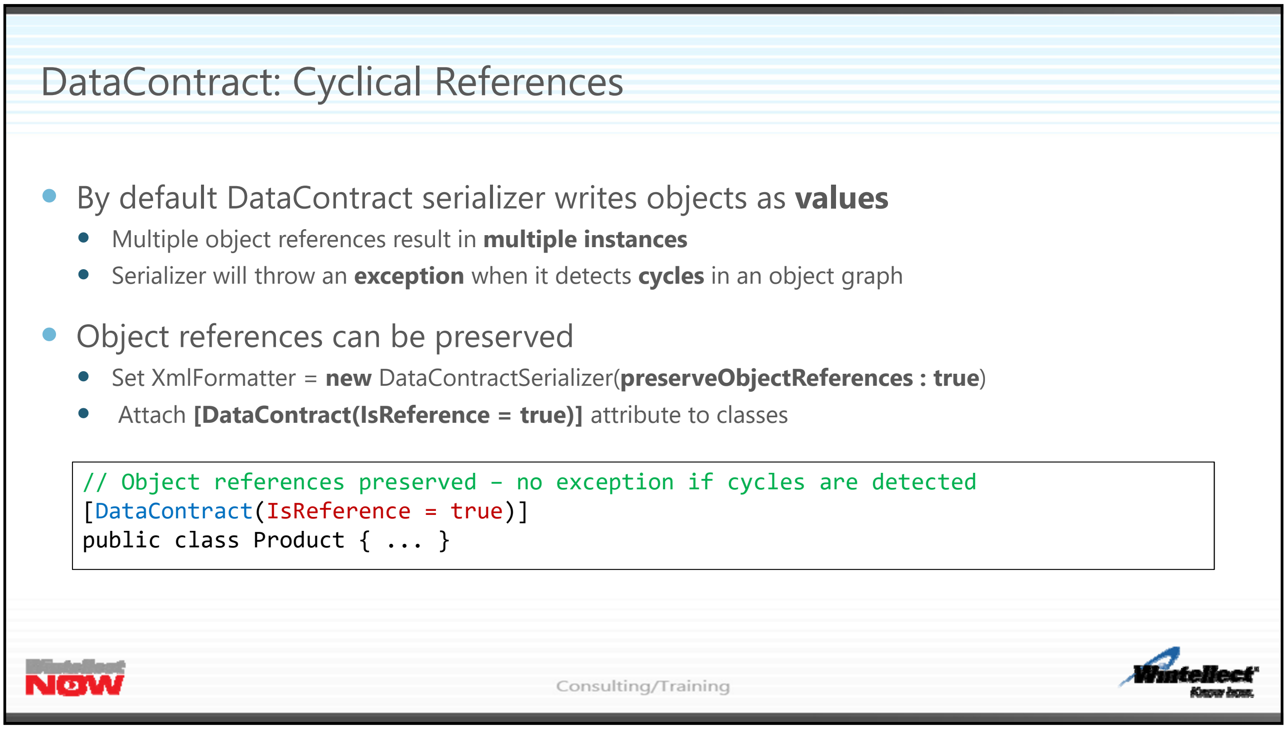
```
// Without [DataContract] – all public properties are serialized
public class Product {

    // Read-write properties will be serialized
    public int ProductId { get; set; }
    public string ProductName { get; set; }

    // Read-only properties will not be serialized
    public Category Category { get { return _category; } }
```



DataContract: With Attributes

- If **[DataContract]** attribute applied to class, property serialization is **opt-in**
  - Only properties adorned with **[DataMember]** are serialized
  - **Private fields** with [DataMember] are also serialized

```
[DataContract] // Only properties and fields with [DataMember] are serialized
public class Product {

    [DataMember] // Will be serialized
    public int ProductId { get; set; }

    [DataMember] // Will be serialized
    private string _productName;

    // Will not be serialized
    public decimal UnitPrice { get; set; }
```

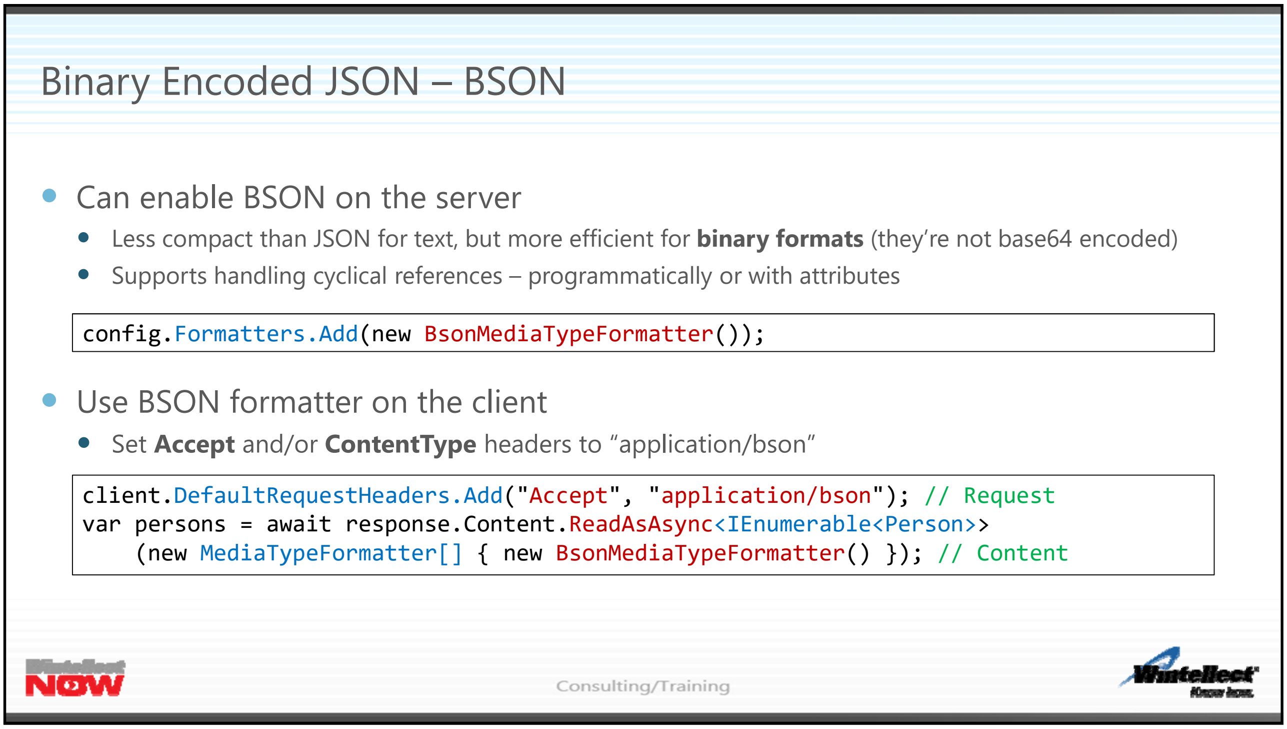


DataContract: Cyclical References

- By default DataContract serializer writes objects as **values**
  - Multiple object references result in **multiple instances**
  - Serializer will throw an **exception** when it detects **cycles** in an object graph
- Object references can be preserved
  - Set XmlFormatter = **new** DataContractSerializer(**preserveObjectReferences : true**)
  - Attach **[DataContract(IsReference = true)]** attribute to classes

```
// Object references preserved – no exception if cycles are detected
[DataContract(IsReference = true)]
public class Product { ... }
```

Consulting/Training



Binary Encoded JSON – BSON

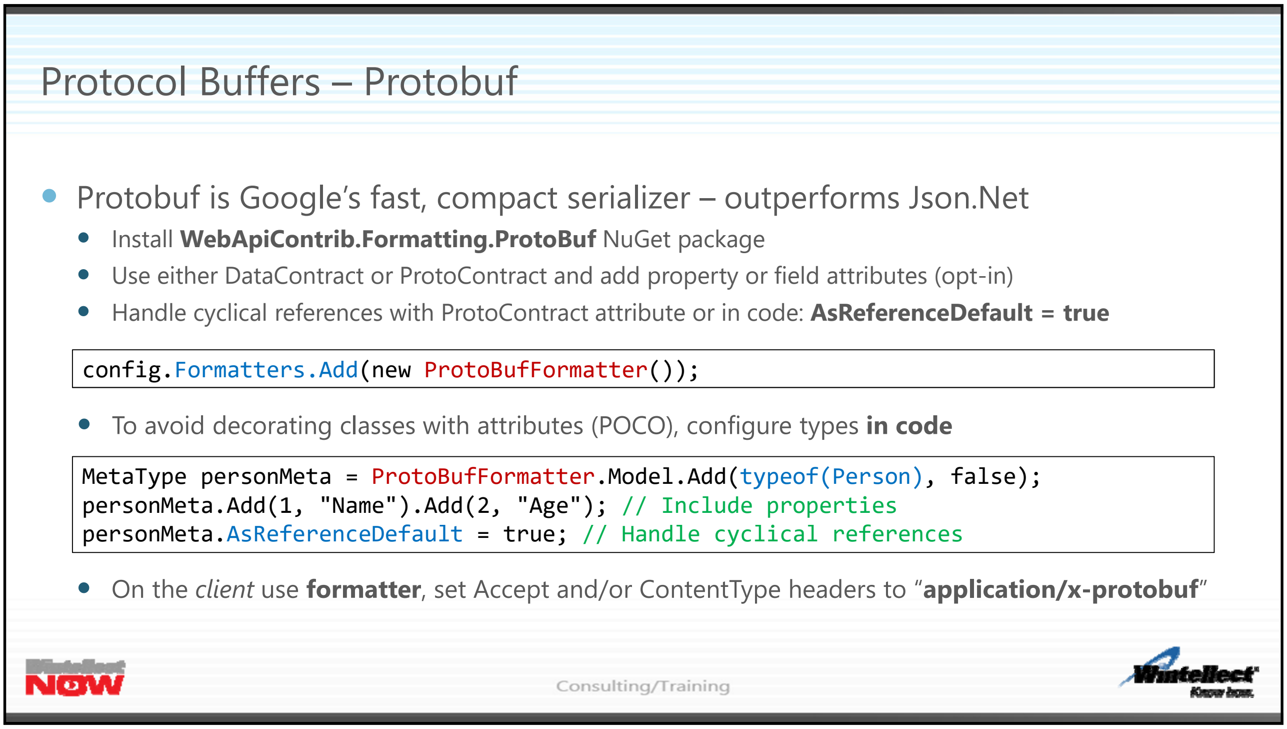
- Can enable BSON on the server
  - Less compact than JSON for text, but more efficient for **binary formats** (they're not base64 encoded)
  - Supports handling cyclical references – programmatically or with attributes

```
config.Formatters.Add(new BsonMediaTypeFormatter());
```

- Use BSON formatter on the client
  - Set **Accept** and/or **ContentType** headers to "application/bson"

```
client.DefaultRequestHeaders.Add("Accept", "application/bson"); // Request
var persons = await response.Content.ReadAsAsync<IEnumerable<Person>>
    (new MediaTypeFormatter[] { new BsonMediaTypeFormatter() }); // Content
```

Consulting/Training



Protocol Buffers – Protobuf

- Protobuf is Google's fast, compact serializer – outperforms Json.Net
  - Install **WebApiContrib.Formatting.ProtoBuf** NuGet package
  - Use either DataContract or ProtoContract and add property or field attributes (opt-in)
  - Handle cyclical references with ProtoContract attribute or in code: **AsReferenceDefault = true**

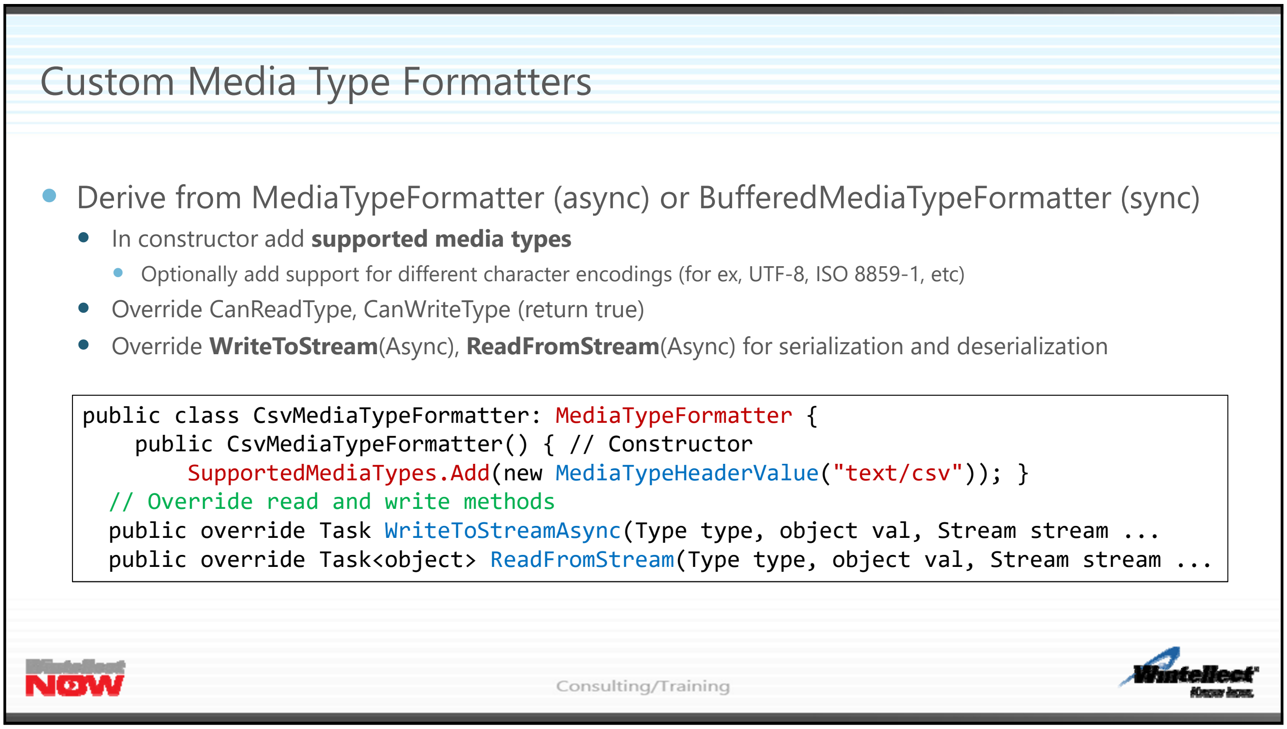
```
config.Formatters.Add(new ProtoBufFormatter());
```

  - To avoid decorating classes with attributes (POCO), configure types **in code**

```
MetaType personMeta = ProtoBufFormatter.Model.Add(typeof(Person), false);
personMeta.Add(1, "Name").Add(2, "Age"); // Include properties
personMeta.AsReferenceDefault = true; // Handle cyclical references
```

  - On the *client* use **formatter**, set Accept and/or ContentType headers to "**application/x-protobuf**"

Consulting/Training



Custom Media Type Formatters

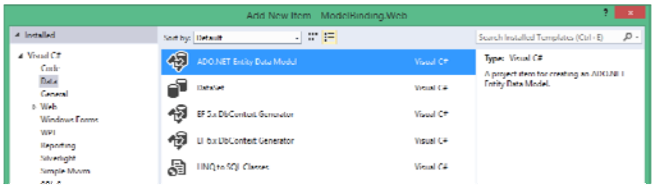
- Derive from MediaTypeFormatter (async) or BufferedMediaTypeFormatter (sync)
  - In constructor add **supported media types**
    - Optionally add support for different character encodings (for ex, UTF-8, ISO 8859-1, etc)
  - Override CanReadType, CanWriteType (return true)
  - Override **WriteToStream**(Async), **ReadFromStream**(Async) for serialization and deserialization

```
public class CsvMediaTypeFormatter: MediaTypeFormatter {
    public CsvMediaTypeFormatter() { // Constructor
        SupportedMediaTypes.Add(new MediaTypeHeaderValue("text/csv")); }
  // Override read and write methods
  public override Task WriteToStreamAsync(Type type, object val, Stream stream ...
  public override Task<object> ReadFromStream(Type type, object val, Stream stream ...
```

Consulting/Training

Code Generation Tools

- What happened to “Add Service Reference”?
  - Web API’s do not expose **metadata** (WSDL) for code-generation tools to use
  - RESTful approach favors embedded hyperlinks over strict contracts
- Entity Framework tooling offers another approach
  - Generate model classes based on an **Entity Data Model**, which is a *conceptual view* of the database
  - Add an EDM, then select either “EF Designer from Database” or “Code First from Database”
  - Code generation can be customized with **T4 templates**
    - Install **EntityFramework. CodeTemplates.CSharp** NuGet package



A project item for creating an ADO.NET Entity Data Model.

Consulting/Training

Entity Framework Gotchas

- Entity Framework generates **dynamic runtime proxies**
  - Used to support features such as Lazy Loading
    - Usually not required for n-tier scenarios
  - Enabled when all properties are defined as **virtual**
    - Should be explicitly **disabled** because runtime proxies are <u>not</u> serializable

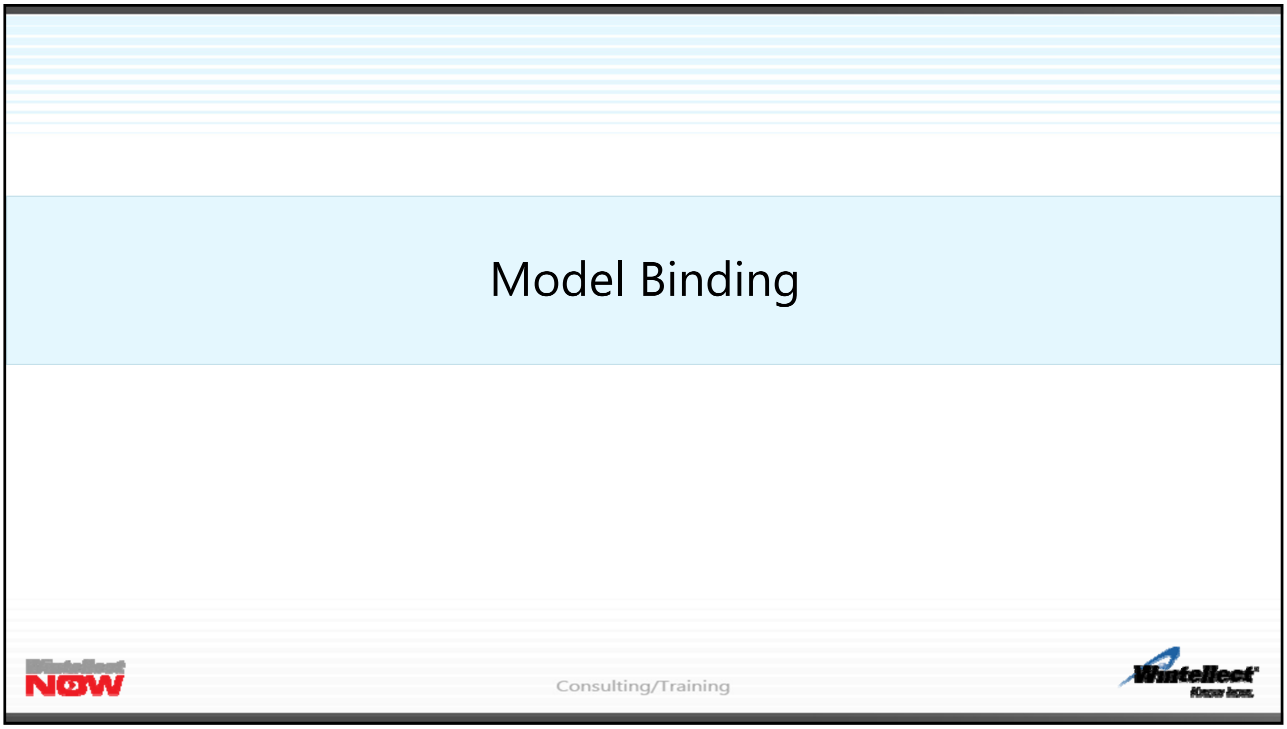
```
public partial class Northwind : DbContext {
   public Northwind() : base("name=Northwind") {

       // Disable dynamic proxies, which are not serializable
       Configuration.ProxyCreationEnabled = false;
   }
```

Consulting/Training

DEMO

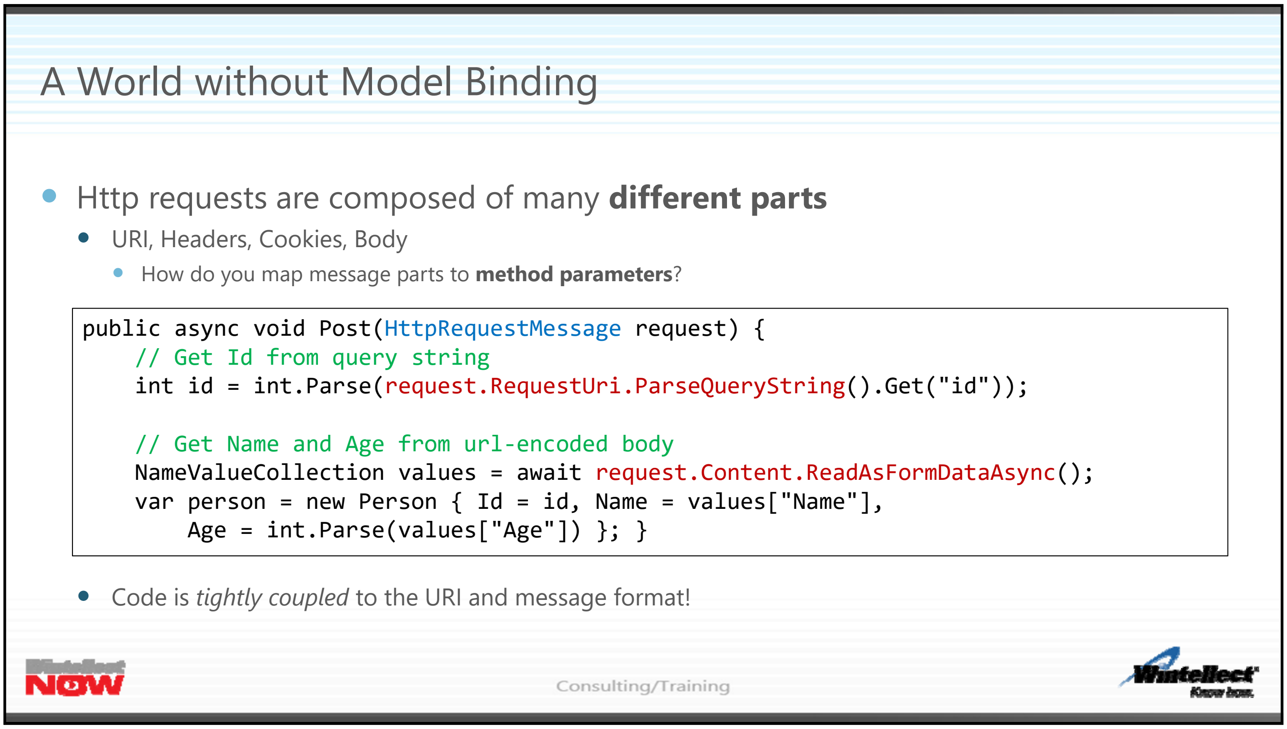
Serialization

Consulting/Training



Model Binding

Consulting/Training

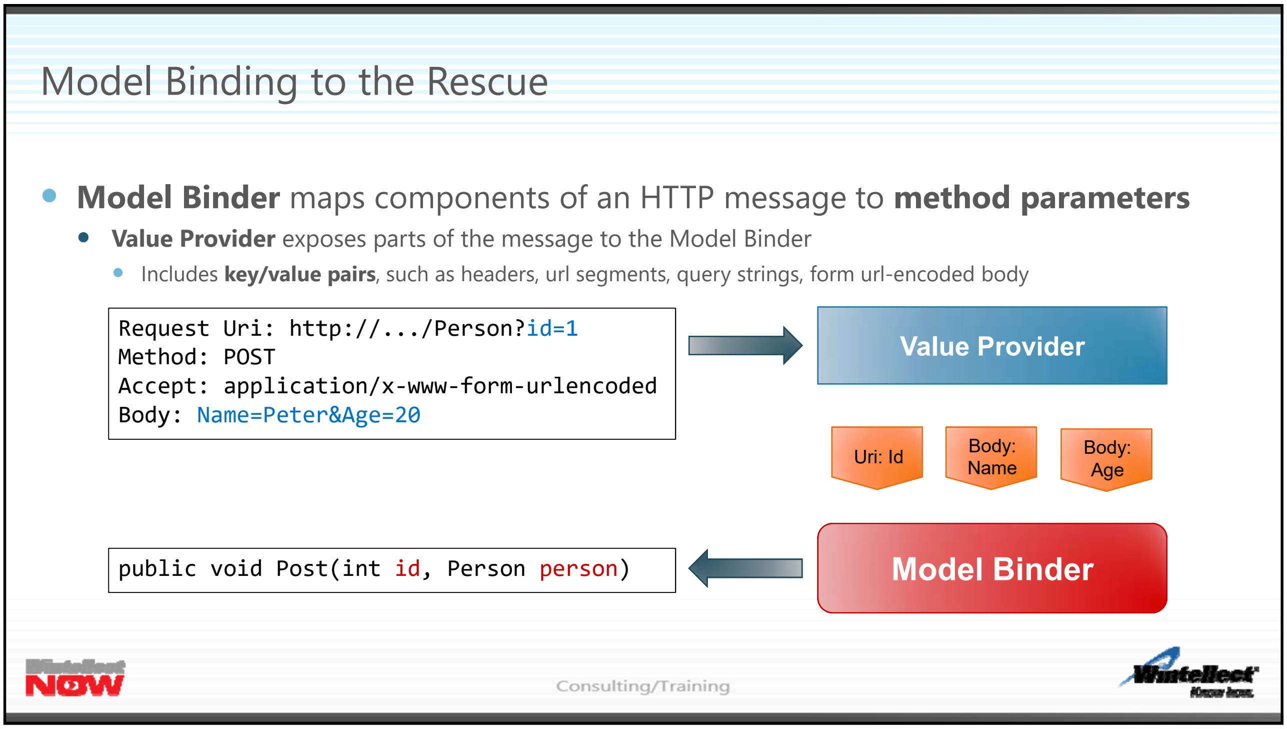


A World without Model Binding

- Http requests are composed of many **different parts**
  - URI, Headers, Cookies, Body
    - How do you map message parts to **method parameters**?

```
public async void Post(HttpRequestMessage request) {
    // Get Id from query string
    int id = int.Parse(request.RequestUri.ParseQueryString().Get("id"));

    // Get Name and Age from url-encoded body
    NameValueCollection values = await request.Content.ReadAsFormDataAsync();
    var person = new Person { Id = id, Name = values["Name"],
        Age = int.Parse(values["Age"]) }; }
```

  - Code is *tightly coupled* to the URI and message format!

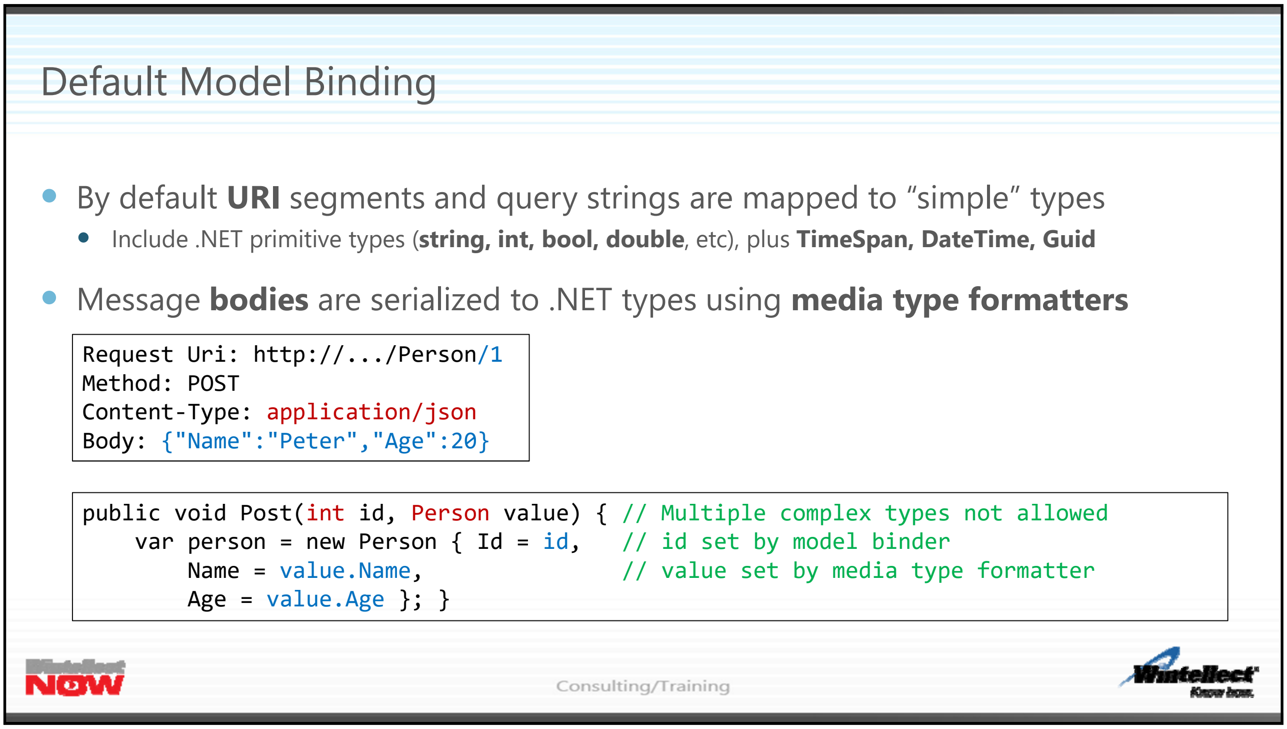


Model Binding to the Rescue

- **Model Binder** maps components of an HTTP message to **method parameters**
  - **Value Provider** exposes parts of the message to the Model Binder
    - Includes **key/value pairs**, such as headers, url segments, query strings, form url-encoded body

```
Request Uri: http://.../Person?id=1
Method: POST
Accept: application/x-www-form-urlencoded
Body: Name=Peter&Age=20
```

```
public void Post(int id, Person person)
```



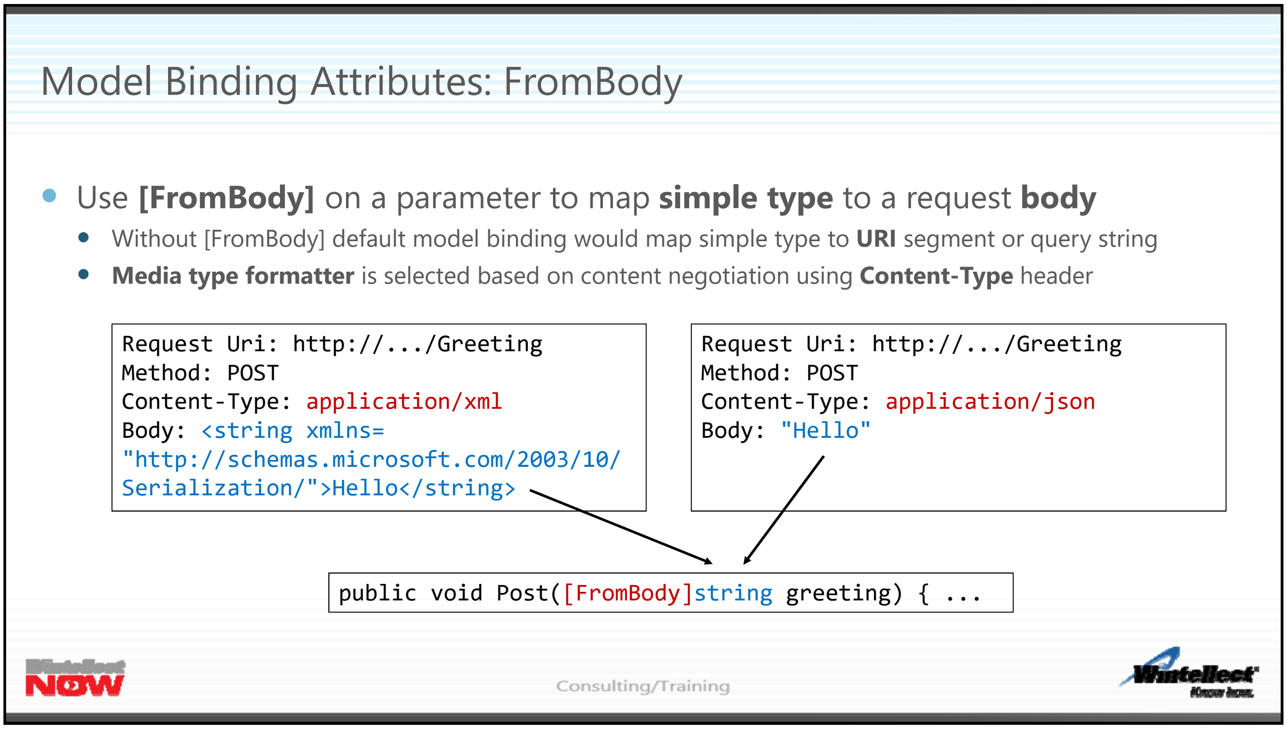
Default Model Binding

- By default **URI** segments and query strings are mapped to “simple” types
  - Include .NET primitive types (**string, int, bool, double**, etc), plus **TimeSpan, DateTime, Guid**
- Message **bodies** are serialized to .NET types using **media type formatters**

```
Request Uri: http://.../Person/1
Method: POST
Content-Type: application/json
Body: {"Name":"Peter","Age":20}
```

```
public void Post(int id, Person value) { // Multiple complex types not allowed
    var person = new Person { Id = id,   // id set by model binder
        Name = value.Name,               // value set by media type formatter
        Age = value.Age }; }
```

Consulting/Training



Model Binding Attributes: FromBody

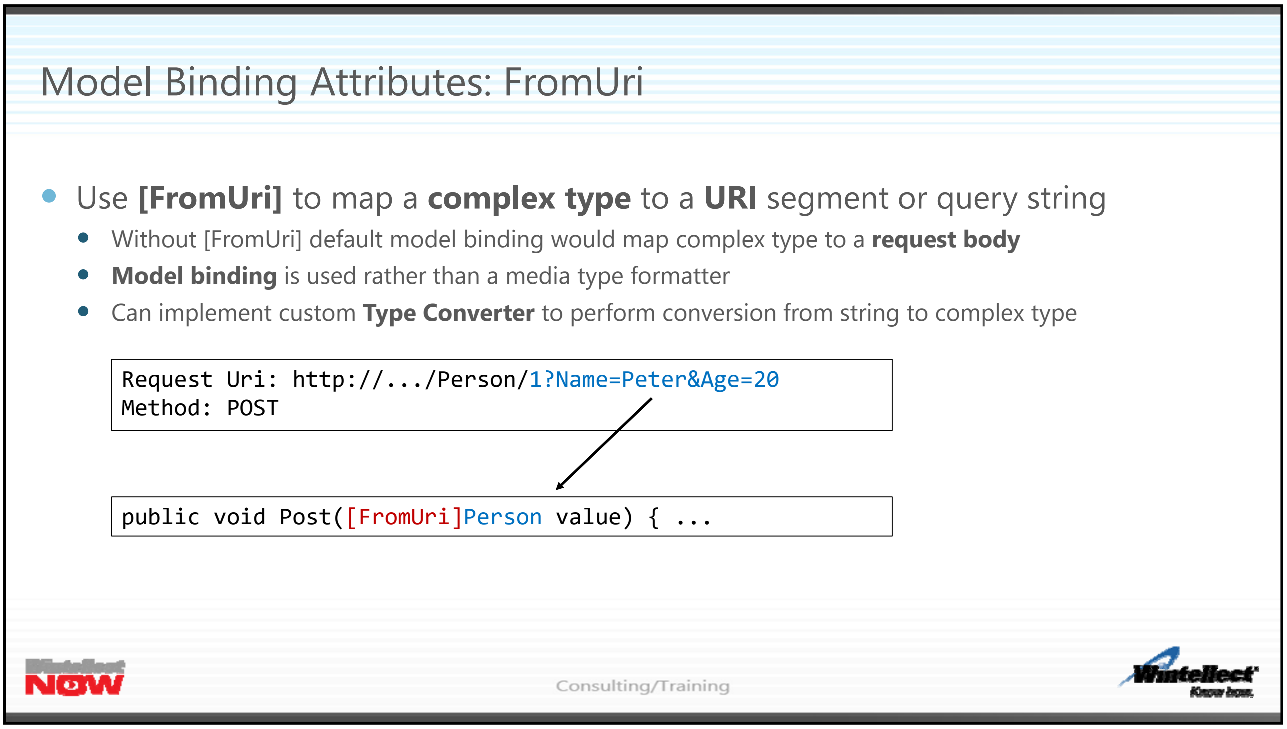
- Use **[FromBody]** on a parameter to map **simple type** to a request **body**
  - Without [FromBody] default model binding would map simple type to **URI** segment or query string
  - **Media type formatter** is selected based on content negotiation using **Content-Type** header

```
Request Uri: http://.../Greeting
Method: POST
Content-Type: application/xml
Body: <string xmlns=
"http://schemas.microsoft.com/2003/10/
Serialization/">Hello</string>
```

```
Request Uri: http://.../Greeting
Method: POST
Content-Type: application/json
Body: "Hello"
```

```
public void Post([FromBody]string greeting) { ...
```

Consulting/Training



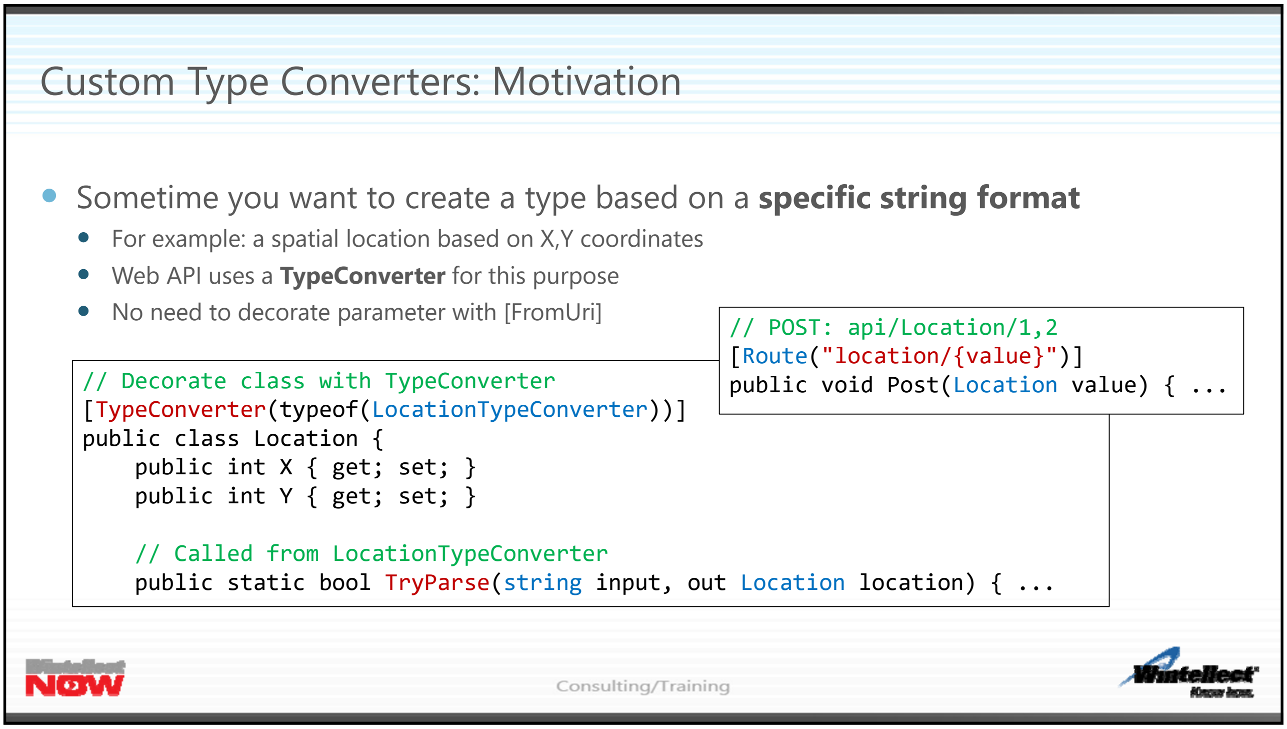
Model Binding Attributes: FromUri

- Use **[FromUri]** to map a **complex type** to a **URI** segment or query string
  - Without [FromUri] default model binding would map complex type to a **request body**
  - **Model binding** is used rather than a media type formatter
  - Can implement custom **Type Converter** to perform conversion from string to complex type

```
Request Uri: http://.../Person/1?Name=Peter&Age=20
Method: POST
```

```
public void Post([FromUri]Person value) { ...
```

Consulting/Training



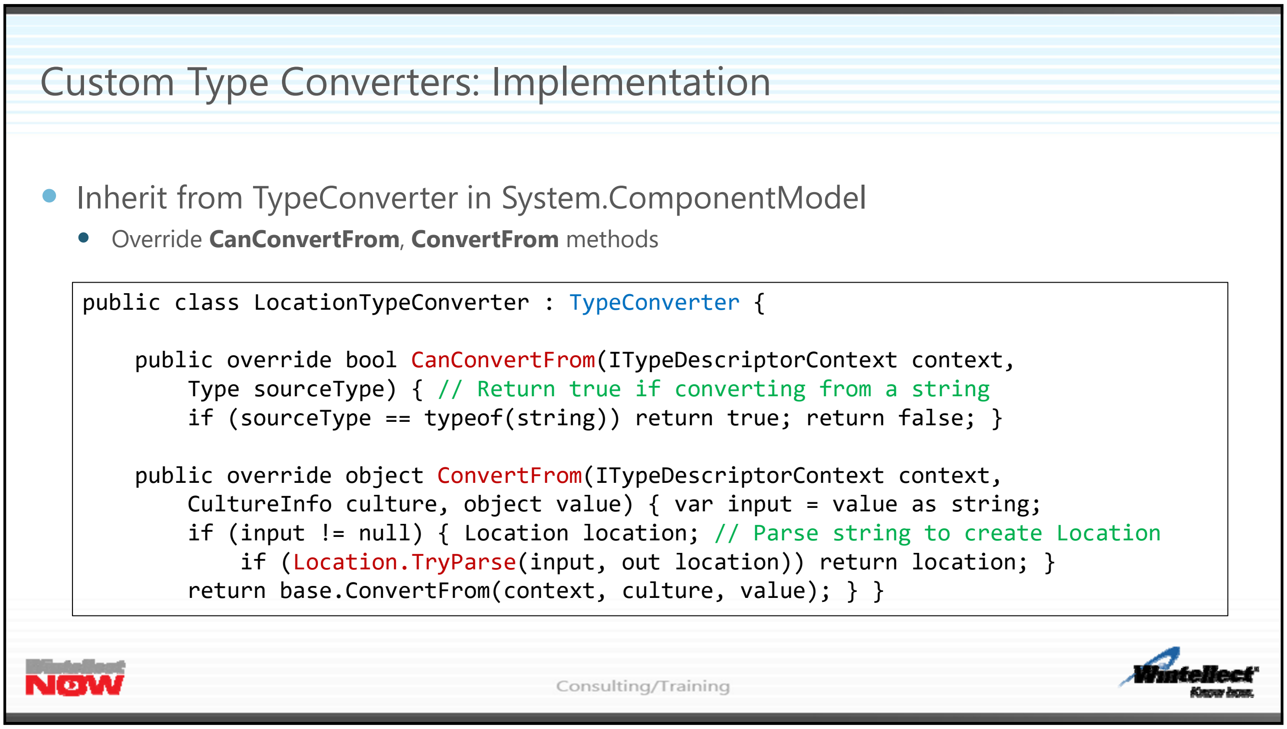
Custom Type Converters: Motivation

- Sometime you want to create a type based on a **specific string format**
  - For example: a spatial location based on X,Y coordinates
  - Web API uses a **TypeConverter** for this purpose
  - No need to decorate parameter with [FromUri]

```
// POST: api/Location/1,2
[Route("location/{value}")]
public void Post(Location value) { ...
```

```
// Decorate class with TypeConverter
[TypeConverter(typeof(LocationTypeConverter))]
public class Location {
    public int X { get; set; }
    public int Y { get; set; }

    // Called from LocationTypeConverter
    public static bool TryParse(string input, out Location location) { ...
```

Consulting/Training

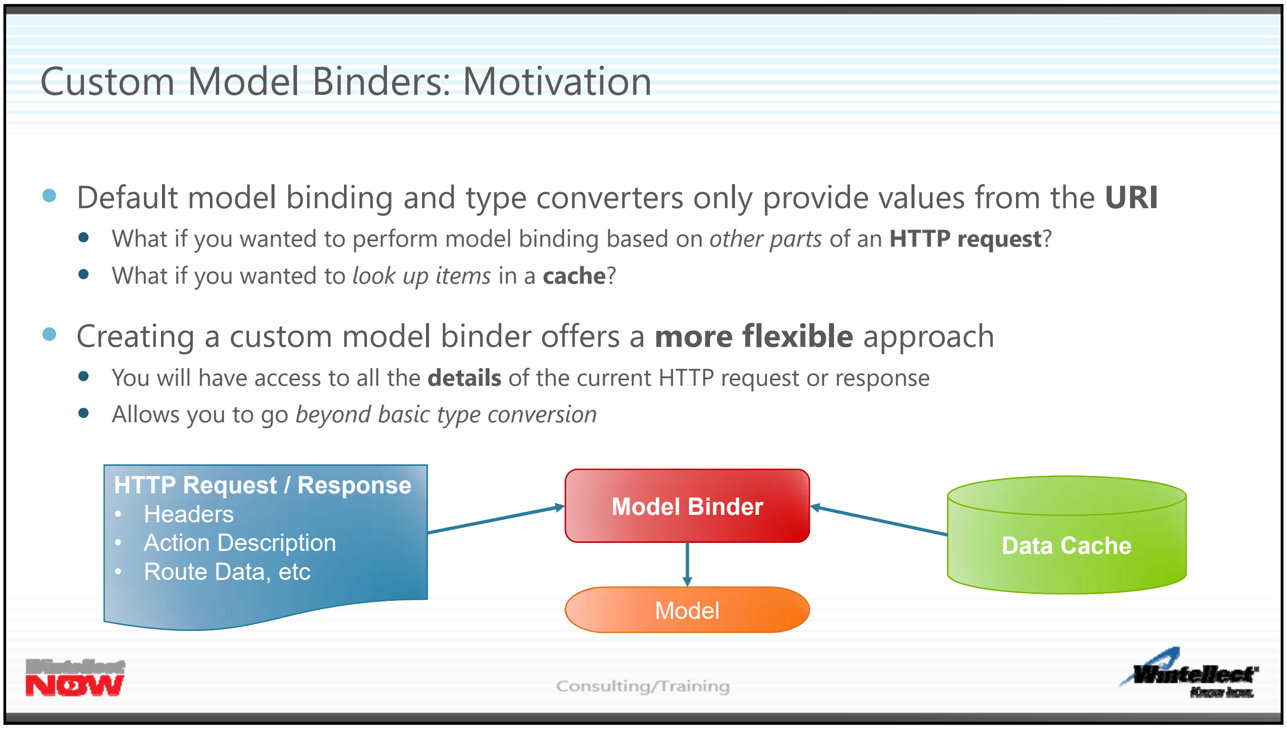


Custom Type Converters: Implementation

- Inherit from TypeConverter in System.ComponentModel
  - Override **CanConvertFrom**, **ConvertFrom** methods

```
public class LocationTypeConverter : TypeConverter {

    public override bool CanConvertFrom(ITypeDescriptorContext context,
        Type sourceType) { // Return true if converting from a string
        if (sourceType == typeof(string)) return true; return false; }

    public override object ConvertFrom(ITypeDescriptorContext context,
        CultureInfo culture, object value) { var input = value as string;
        if (input != null) { Location location; // Parse string to create Location
            if (Location.TryParse(input, out location)) return location; }
        return base.ConvertFrom(context, culture, value); } }
```

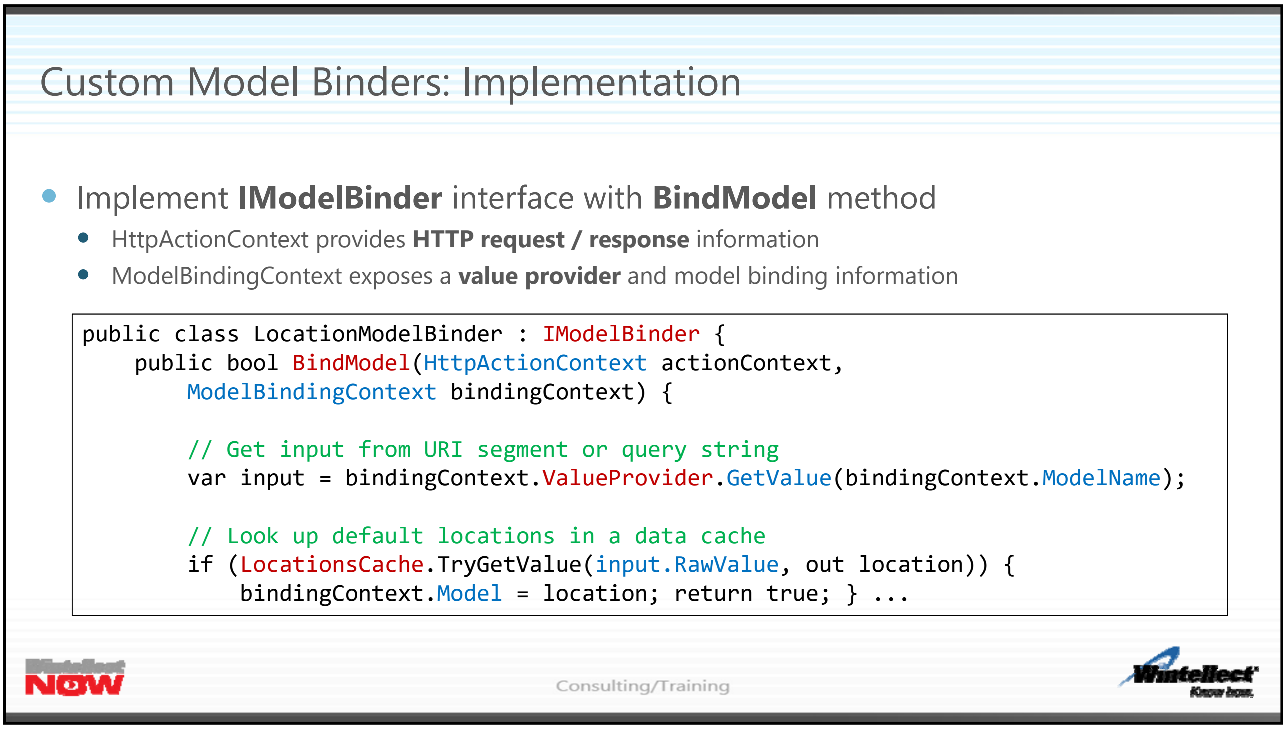
Consulting/Training



Custom Model Binders: Motivation

- Default model binding and type converters only provide values from the **URI**
  - What if you wanted to perform model binding based on *other parts* of an **HTTP request**?
  - What if you wanted to *look up items* in a **cache**?
- Creating a custom model binder offers a **more flexible** approach
  - You will have access to all the **details** of the current HTTP request or response
  - Allows you to go *beyond basic type conversion*

Consulting/Training

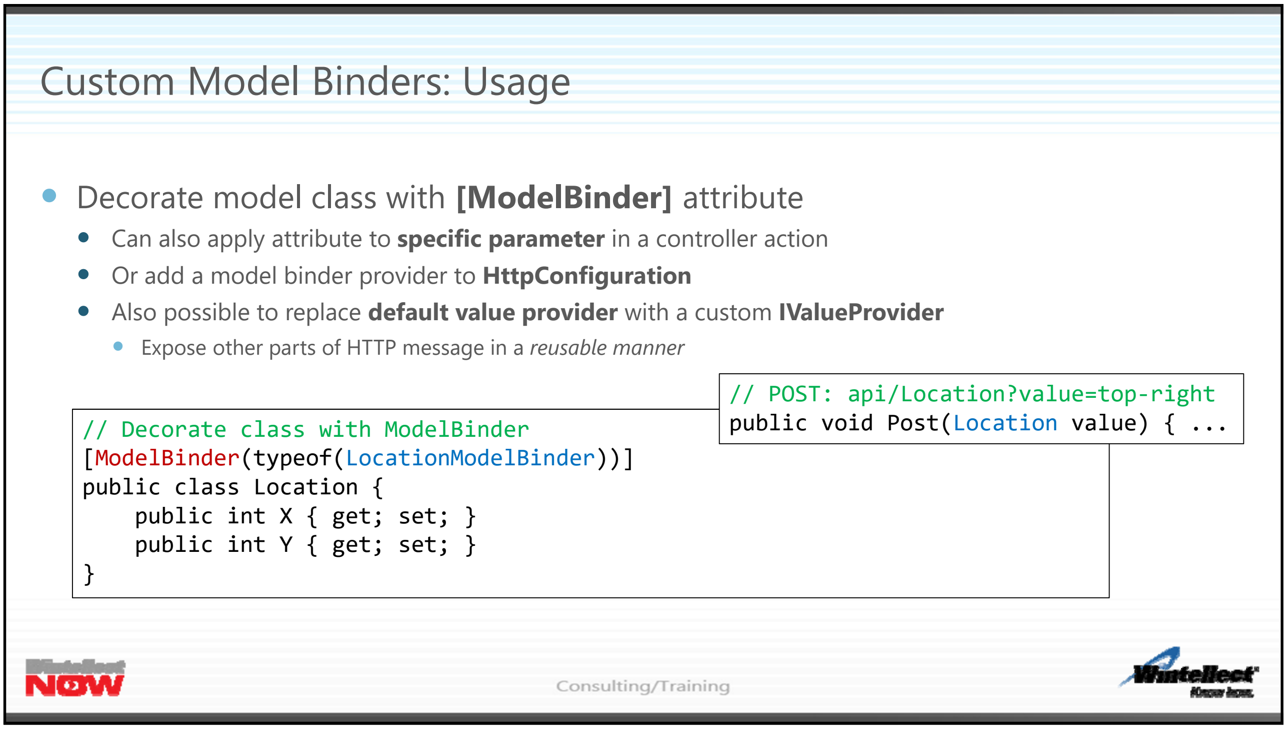


Custom Model Binders: Implementation

- Implement **IModelBinder** interface with **BindModel** method
  - HttpActionContext provides **HTTP request / response** information
  - ModelBindingContext exposes a **value provider** and model binding information

```
public class LocationModelBinder : IModelBinder {
    public bool BindModel(HttpActionContext actionContext,
        ModelBindingContext bindingContext) {

        // Get input from URI segment or query string
        var input = bindingContext.ValueProvider.GetValue(bindingContext.ModelName);

        // Look up default locations in a data cache
        if (LocationsCache.TryGetValue(input.RawValue, out location)) {
            bindingContext.Model = location; return true; } ...
```

Consulting/Training



Custom Model Binders: Usage

- Decorate model class with **[ModelBinder]** attribute
  - Can also apply attribute to **specific parameter** in a controller action
  - Or add a model binder provider to **HttpConfiguration**
  - Also possible to replace **default value provider** with a custom **IValueProvider**
    - Expose other parts of HTTP message in a *reusable manner*

```
// POST: api/Location?value=top-right
public void Post(Location value) { ...
```

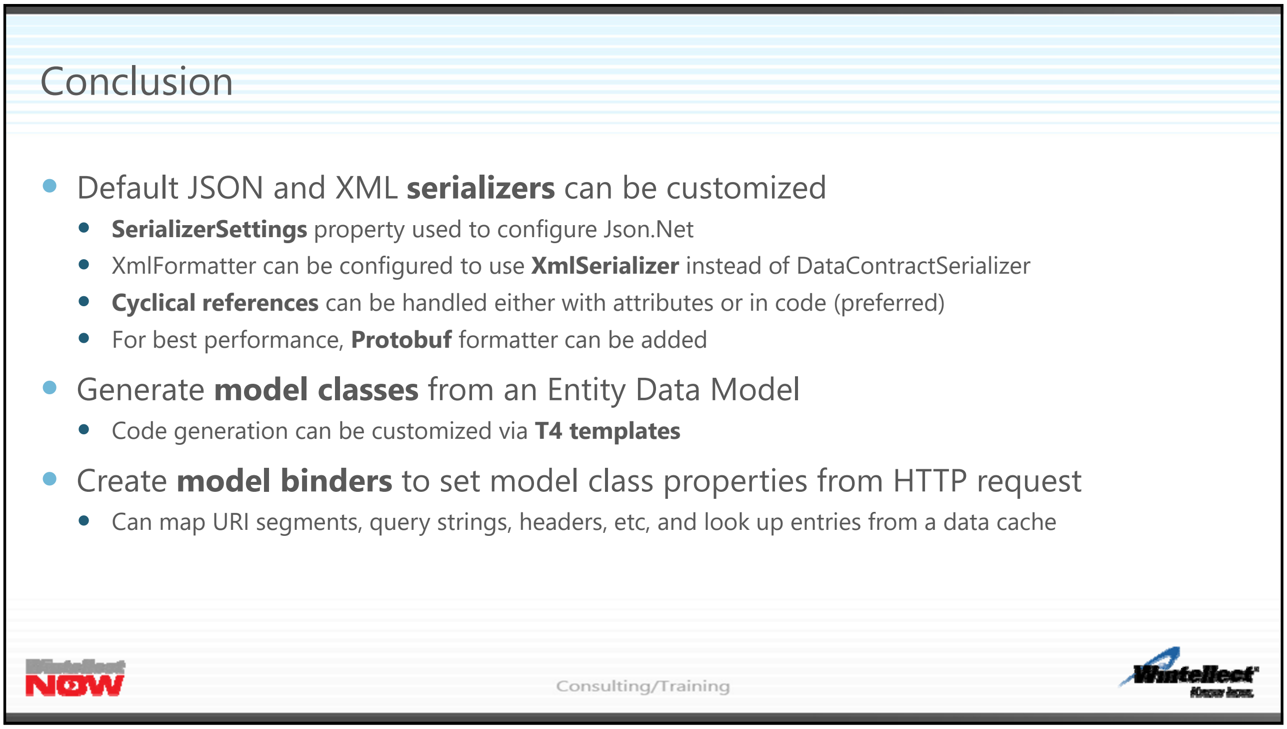
```
// Decorate class with ModelBinder
[ModelBinder(typeof(LocationModelBinder))]
public class Location {
    public int X { get; set; }
    public int Y { get; set; }
}
```

Consulting/Training

DEMO

Model Binding

Consulting/Training



Conclusion

- Default JSON and XML **serializers** can be customized
  - **SerializerSettings** property used to configure Json.Net
  - XmlFormatter can be configured to use **XmlSerializer** instead of DataContractSerializer
  - **Cyclical references** can be handled either with attributes or in code (preferred)
  - For best performance, **Protobuf** formatter can be added
- Generate **model classes** from an Entity Data Model
  - Code generation can be customized via **T4 templates**
- Create **model binders** to set model class properties from HTTP request
  - Can map URI segments, query strings, headers, etc, and look up entries from a data cache

Consulting/Training

Serialization and Model Binding

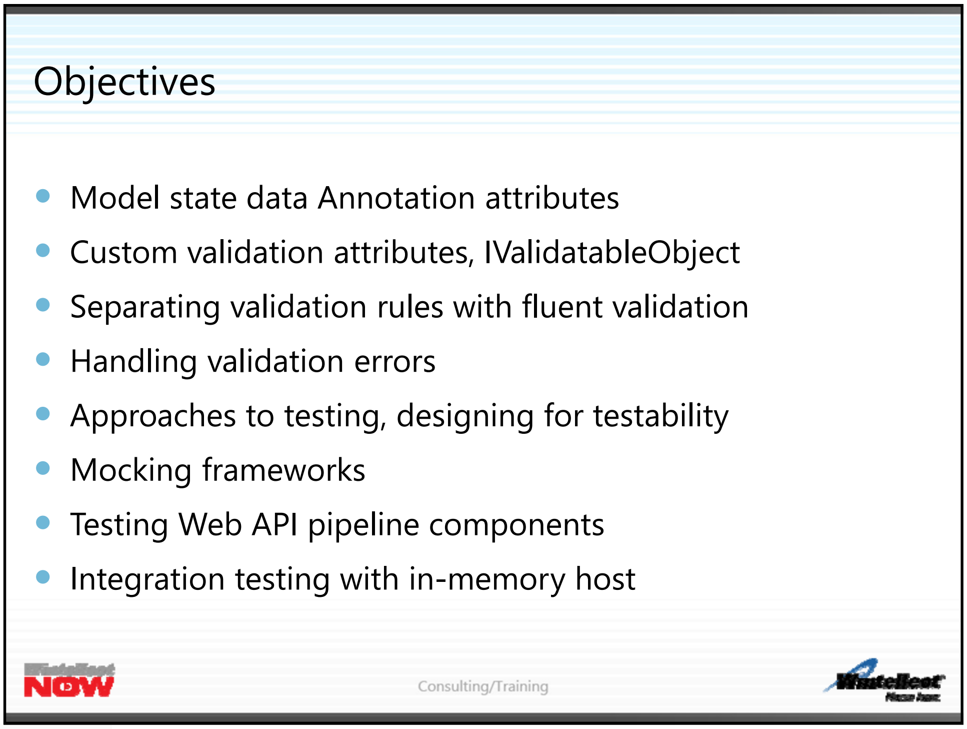
Tony Sneed
tsneed@wintellect.com
http://blog.tonysneed.com

Consulting/Training

Validation and Testing with ASP.NET Web API

Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

Consulting/Training



Objectives

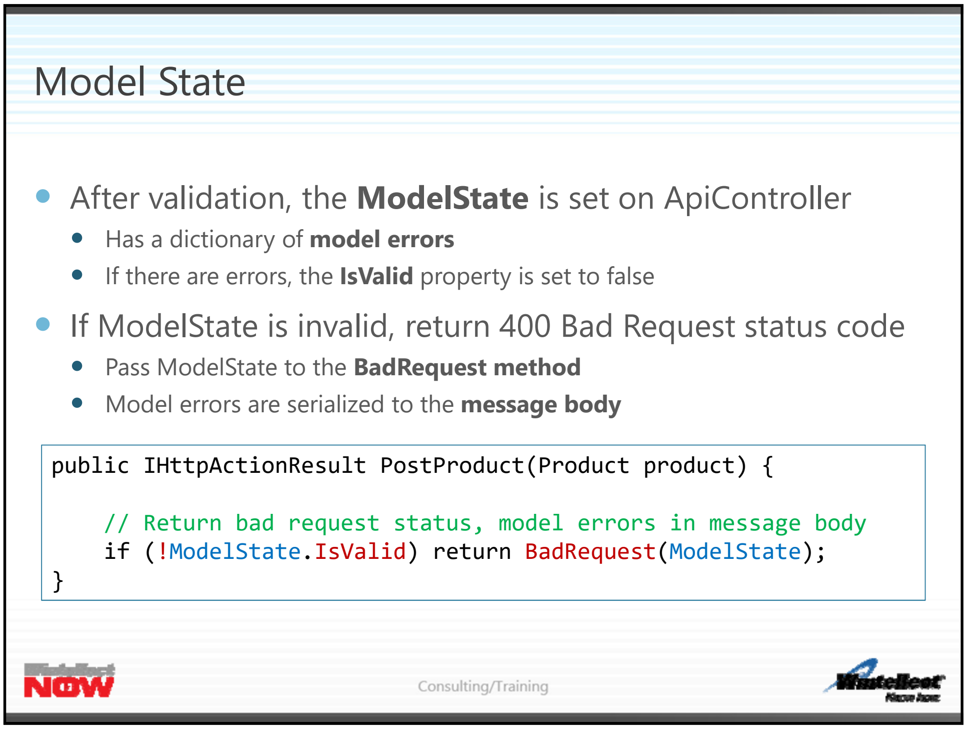
- Model state data Annotation attributes
- Custom validation attributes, IValidatableObject
- Separating validation rules with fluent validation
- Handling validation errors
- Approaches to testing, designing for testability
- Mocking frameworks
- Testing Web API pipeline components
- Integration testing with in-memory host

Consulting/Training

Data Validation

Validating Input

- Always **validate** client input
  - Invalid input can pose a **security risk**
- Validation is performed using **Data Annotations**
  - Rules defined as **attributes** placed on properties
- Custom validation performed by implementing **IValidatableObject**
  - Supports entity-level validation
- Can **separate** validation rules from model classes
  - FluentValidation open-source library enables this

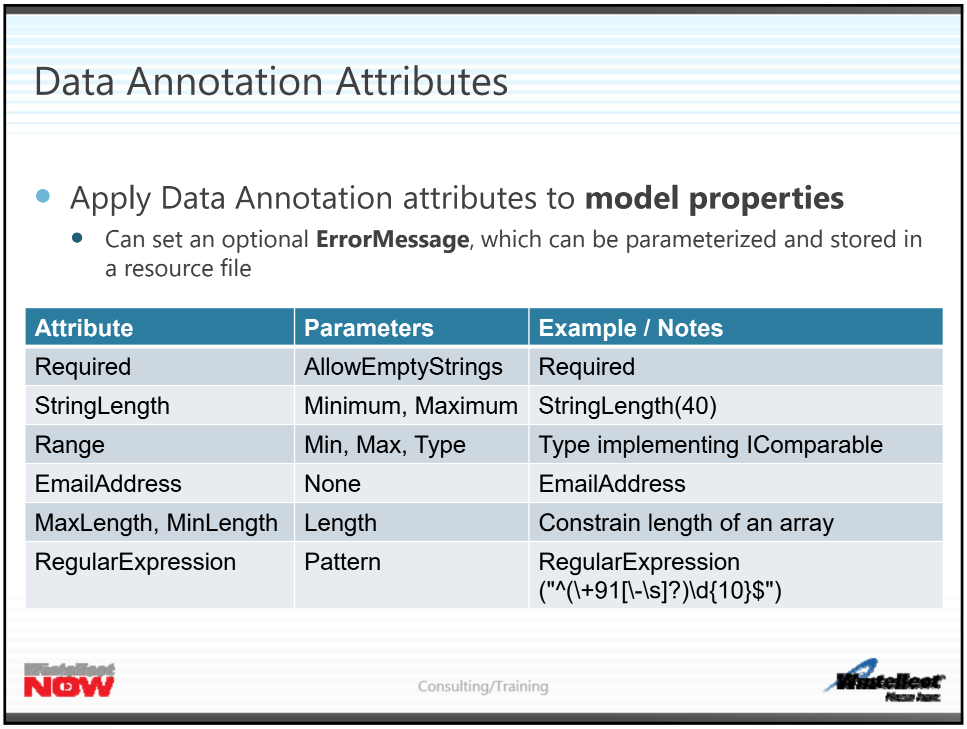


Model State

- After validation, the **ModelState** is set on ApiController
  - Has a dictionary of **model errors**
  - If there are errors, the **IsValid** property is set to false
- If ModelState is invalid, return 400 Bad Request status code
  - Pass ModelState to the **BadRequest method**
  - Model errors are serialized to the **message body**

```
public IHttpActionResult PostProduct(Product product) {

    // Return bad request status, model errors in message body
    if (!ModelState.IsValid) return BadRequest(ModelState);
}
```

Consulting/Training

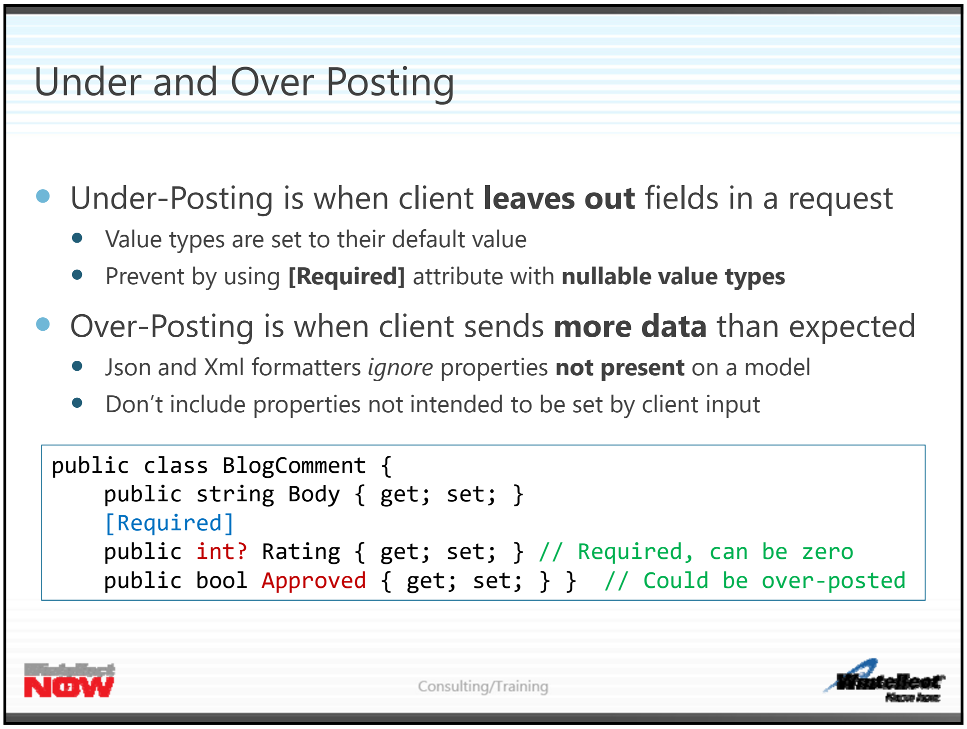


Data Annotation Attributes

- Apply Data Annotation attributes to **model properties**
  - Can set an optional **ErrorMessage**, which can be parameterized and stored in a resource file

Attribute	Parameters	Example / Notes
Required	AllowEmptyStrings	Required
StringLength	Minimum, Maximum	StringLength(40)
Range	Min, Max, Type	Type implementing IComparable
EmailAddress	None	EmailAddress
MaxLength, MinLength	Length	Constrain length of an array
RegularExpression	Pattern	RegularExpression ("^(\+91[\-\s]?)\d{10}$")

Consulting/Training

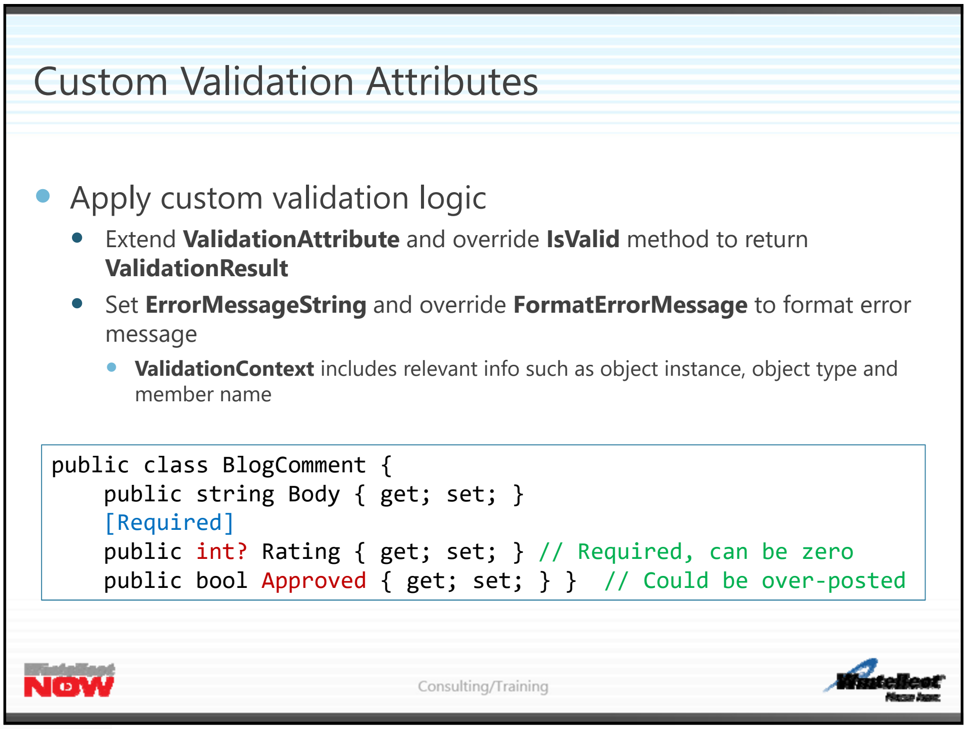


Under and Over Posting

- Under-Posting is when client **leaves out** fields in a request
  - Value types are set to their default value
  - Prevent by using **[Required]** attribute with **nullable value types**
- Over-Posting is when client sends **more data** than expected
  - Json and Xml formatters *ignore* properties **not present** on a model
  - Don’t include properties not intended to be set by client input

```
public class BlogComment {
    public string Body { get; set; }
    [Required]
    public int? Rating { get; set; } // Required, can be zero
    public bool Approved { get; set; } }  // Could be over-posted
```

Consulting/Training

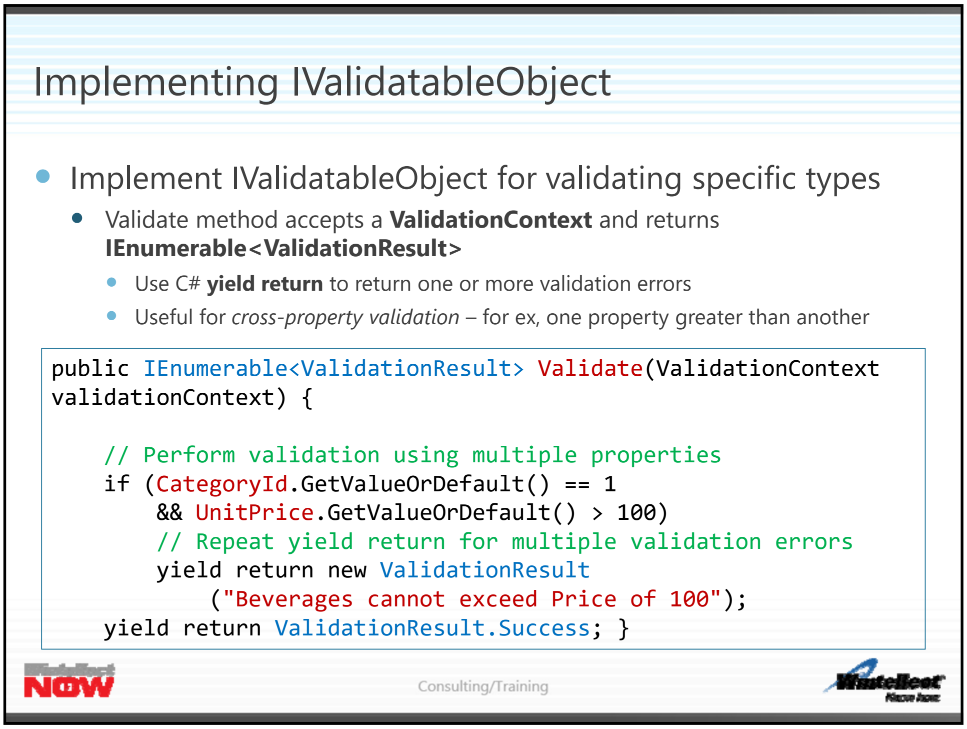


Custom Validation Attributes

- Apply custom validation logic
  - Extend **ValidationAttribute** and override **IsValid** method to return **ValidationResult**
  - Set **ErrorMessageString** and override **FormatErrorMessage** to format error message
    - **ValidationContext** includes relevant info such as object instance, object type and member name

```
public class BlogComment {
    public string Body { get; set; }
    [Required]
    public int? Rating { get; set; } // Required, can be zero
    public bool Approved { get; set; } }  // Could be over-posted
```

Consulting/Training

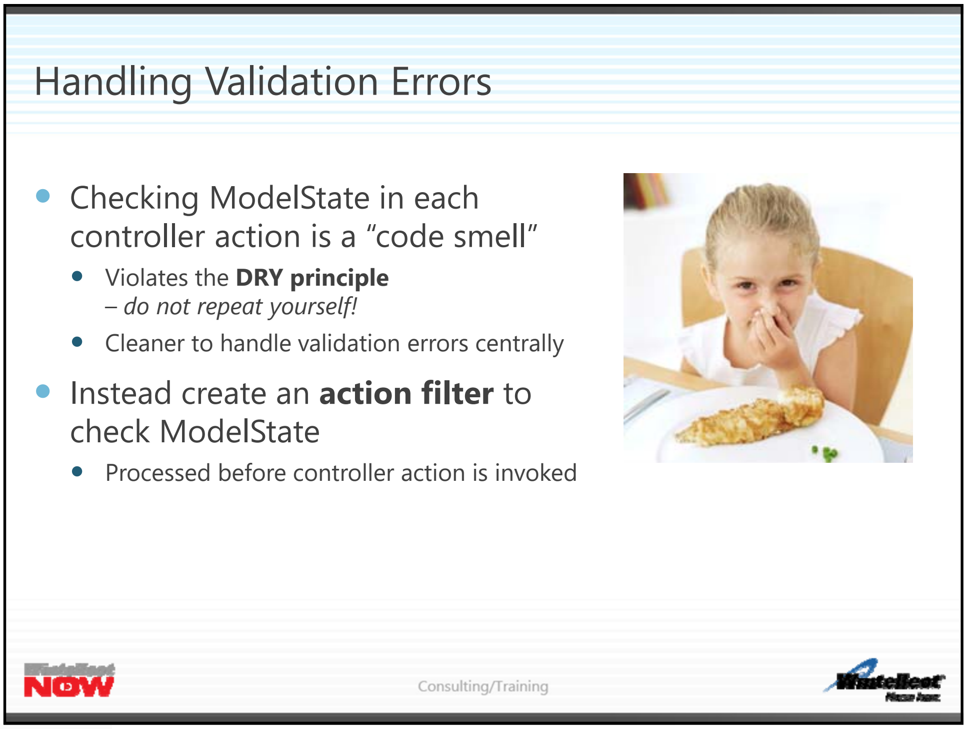


Implementing IValidatableObject

- Implement IValidatableObject for validating specific types
  - Validate method accepts a **ValidationContext** and returns **IEnumerable<ValidationResult>**
    - Use C# **yield return** to return one or more validation errors
    - Useful for *cross-property validation* – for ex, one property greater than another

```
public IEnumerable<ValidationResult> Validate(ValidationContext
validationContext) {

    // Perform validation using multiple properties
    if (CategoryId.GetValueOrDefault() == 1
        && UnitPrice.GetValueOrDefault() > 100)
        // Repeat yield return for multiple validation errors
        yield return new ValidationResult
            ("Beverages cannot exceed Price of 100");
    yield return ValidationResult.Success; }
```

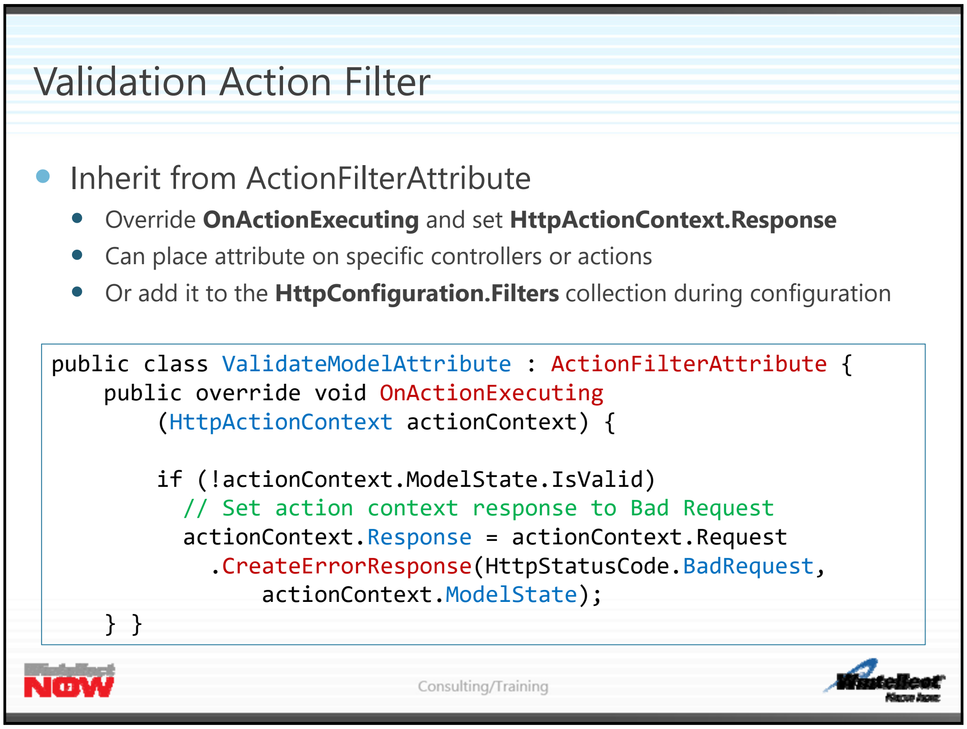
Consulting/Training



Handling Validation Errors

- Checking ModelState in each controller action is a “code smell”
  - Violates the **DRY principle**
    – do not repeat yourself!
  - Cleaner to handle validation errors centrally
- Instead create an **action filter** to check ModelState
  - Processed before controller action is invoked

Consulting/Training

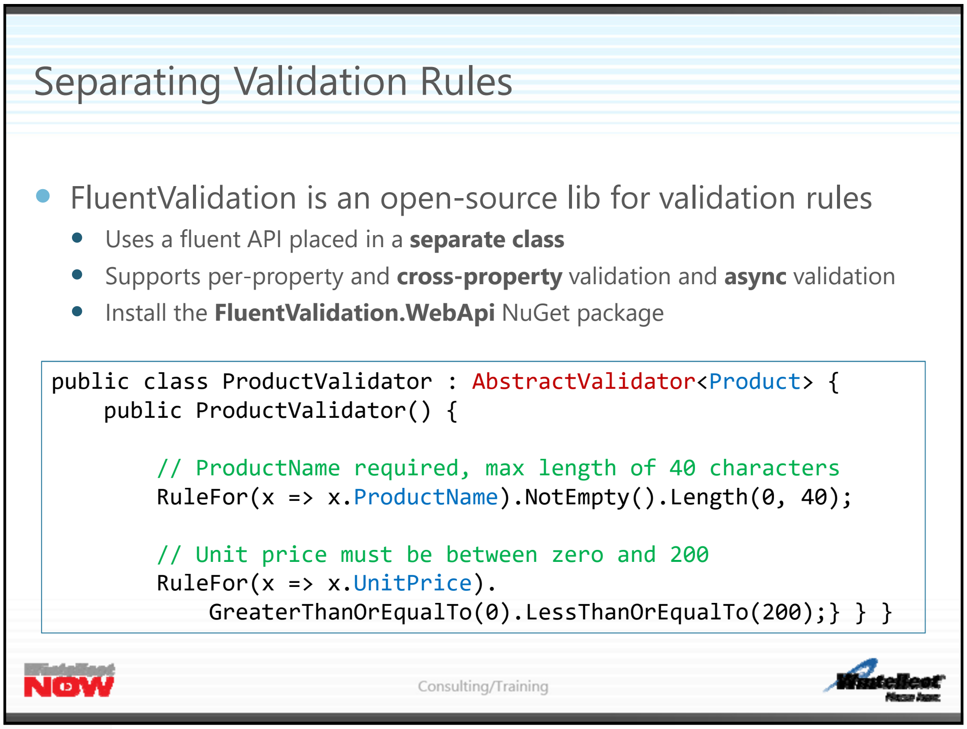


Validation Action Filter

- Inherit from ActionFilterAttribute
  - Override **OnActionExecuting** and set **HttpActionContext.Response**
  - Can place attribute on specific controllers or actions
  - Or add it to the **HttpConfiguration.Filters** collection during configuration

```
public class ValidateModelAttribute : ActionFilterAttribute {
    public override void OnActionExecuting
        (HttpActionContext actionContext) {

        if (!actionContext.ModelState.IsValid)
          // Set action context response to Bad Request
          actionContext.Response = actionContext.Request
            .CreateErrorResponse(HttpStatusCode.BadRequest,
                actionContext.ModelState);
    } }
```

Consulting/Training

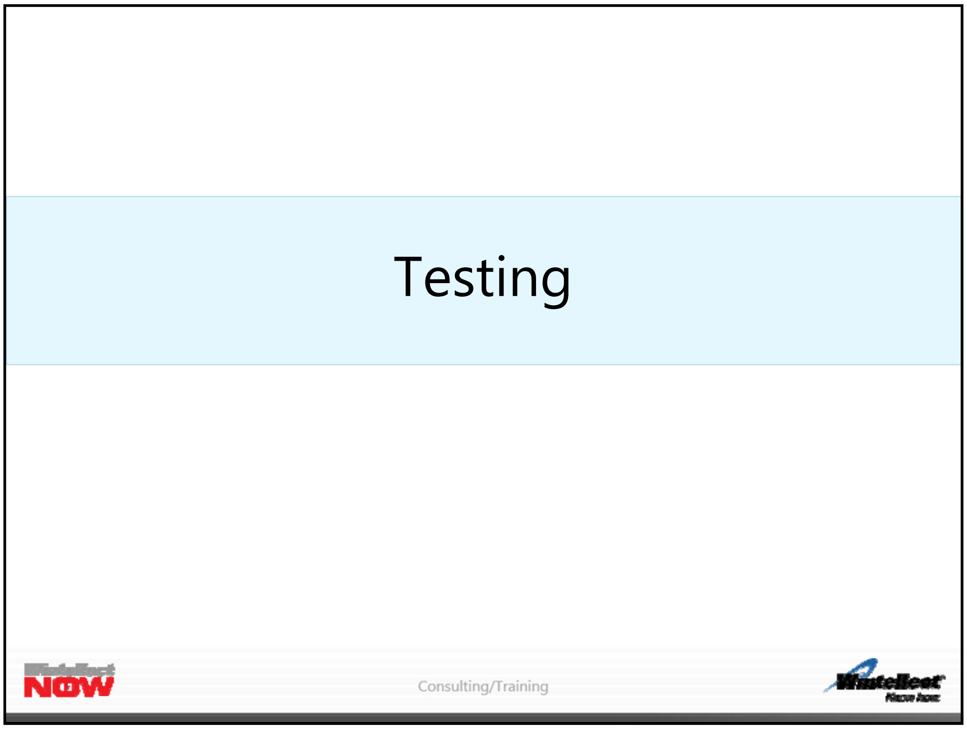


Separating Validation Rules

- FluentValidation is an open-source lib for validation rules
  - Uses a fluent API placed in a **separate class**
  - Supports per-property and **cross-property** validation and **async** validation
  - Install the **FluentValidation.WebApi** NuGet package

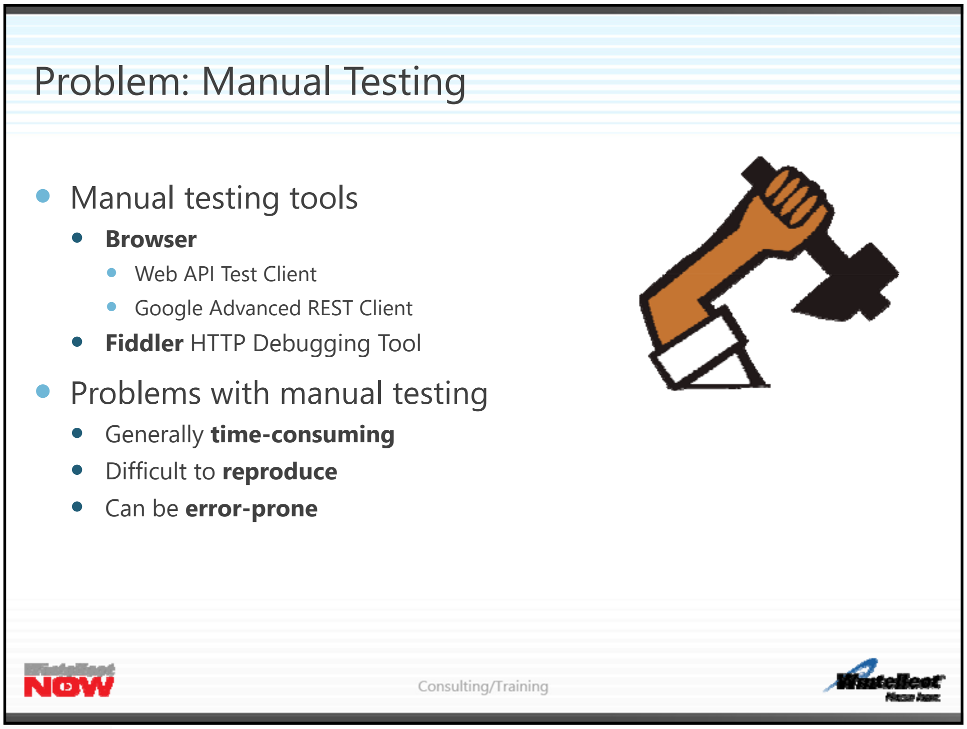
```
public class ProductValidator : AbstractValidator<Product> {
    public ProductValidator() {

        // ProductName required, max length of 40 characters
        RuleFor(x => x.ProductName).NotEmpty().Length(0, 40);

        // Unit price must be between zero and 200
        RuleFor(x => x.UnitPrice).
            GreaterThanOrEqualTo(0).LessThanOrEqualTo(200);} } }
```

Consulting/Training



Testing

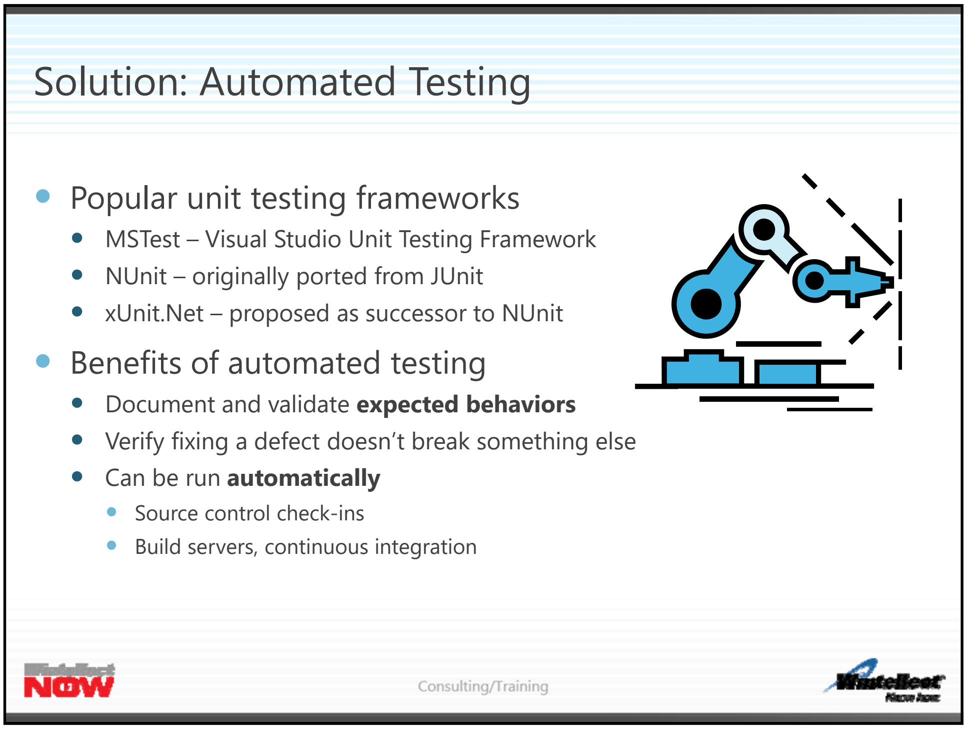
Consulting/Training



Problem: Manual Testing

- Manual testing tools
  - **Browser**
    - Web API Test Client
    - Google Advanced REST Client
  - **Fiddler** HTTP Debugging Tool
- Problems with manual testing
  - Generally **time-consuming**
  - Difficult to **reproduce**
  - Can be **error-prone**

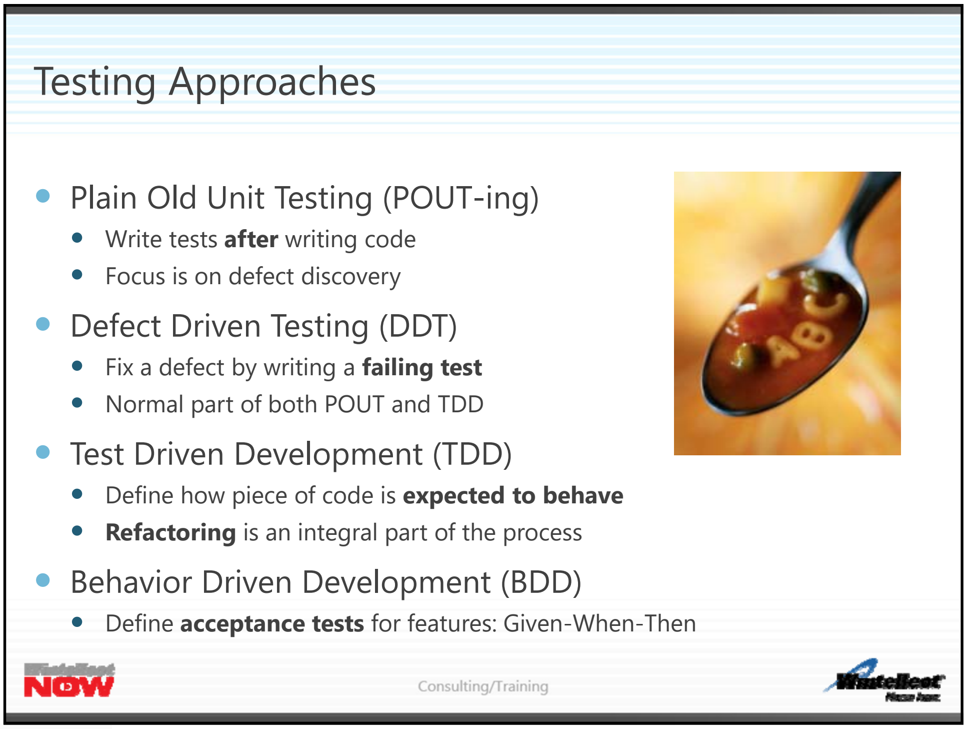
Consulting/Training



Solution: Automated Testing

- Popular unit testing frameworks
  - MSTest – Visual Studio Unit Testing Framework
  - NUnit – originally ported from JUnit
  - xUnit.Net – proposed as successor to NUnit
- Benefits of automated testing
  - Document and validate **expected behaviors**
  - Verify fixing a defect doesn't break something else
  - Can be run **automatically**
    - Source control check-ins
    - Build servers, continuous integration

Consulting/Training



Testing Approaches

- Plain Old Unit Testing (POUT-ing)
  - Write tests **after** writing code
  - Focus is on defect discovery
- Defect Driven Testing (DDT)
  - Fix a defect by writing a **failing test**
  - Normal part of both POUT and TDD
- Test Driven Development (TDD)
  - Define how piece of code is **expected to behave**
  - **Refactoring** is an integral part of the process
- Behavior Driven Development (BDD)
  - Define **acceptance tests** for features: Given-When-Then

Consulting/Training

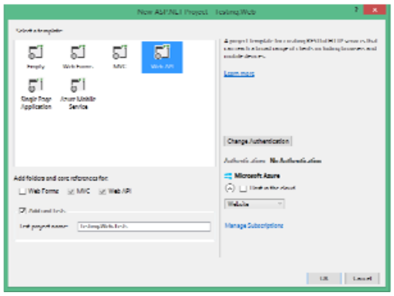
Unit Tests versus Integration Tests

- Unit Tests
  - Serve as **specification** of required functionality
  - Aim to demonstrate just **one behavior**
  - Decoupled from **external dependencies**
  - No **interdependencies** with other tests
  - Can be run **in parallel**
- Integration Tests
  - **End-to-end** from client to server
  - Includes all **components** in the stack
    - Controllers, filters, message handlers, etc
  - Can include **external dependencies**

Consulting/Training

Visual Studio Unit Test Template

- Add tests to a Web API project
  - Just a **checkbox** in New ASP.NET Project dialog
  - Uses **Visual Studio Unit Testing Framework**
  - Provides unit tests for MVC and API controllers



Authentication: No Authentication

Add folders and core references for:

□ Web Forms ☑ MVC ☑ Web API

☑ Add unit tests

Consulting/Training

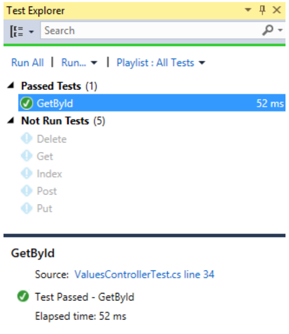
Hello World Unit Tests

- Create the **controller** and invoke the **action**
  - Write **Asserts** to verify expected results

```
[TestClass] // MSTest attributes
public class ValuesControllerTest {

    [TestMethod]
    public void GetById() {

        var controller = new ValuesController();  // Arrange
        string result = controller.Get(5);        // Act
        Assert.AreEqual("value", result);         // Assert
    } }
```

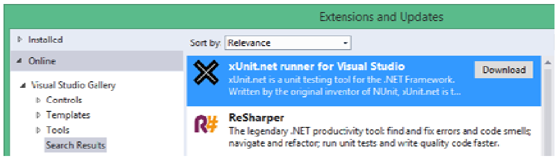
Visual Studio Test Runner

- Also works with xUnit.net, NUnit
  - Install extensions: Tools, Extensions and Updates
- Other test runners also available
  - ReSharper, TestDriven.net, etc



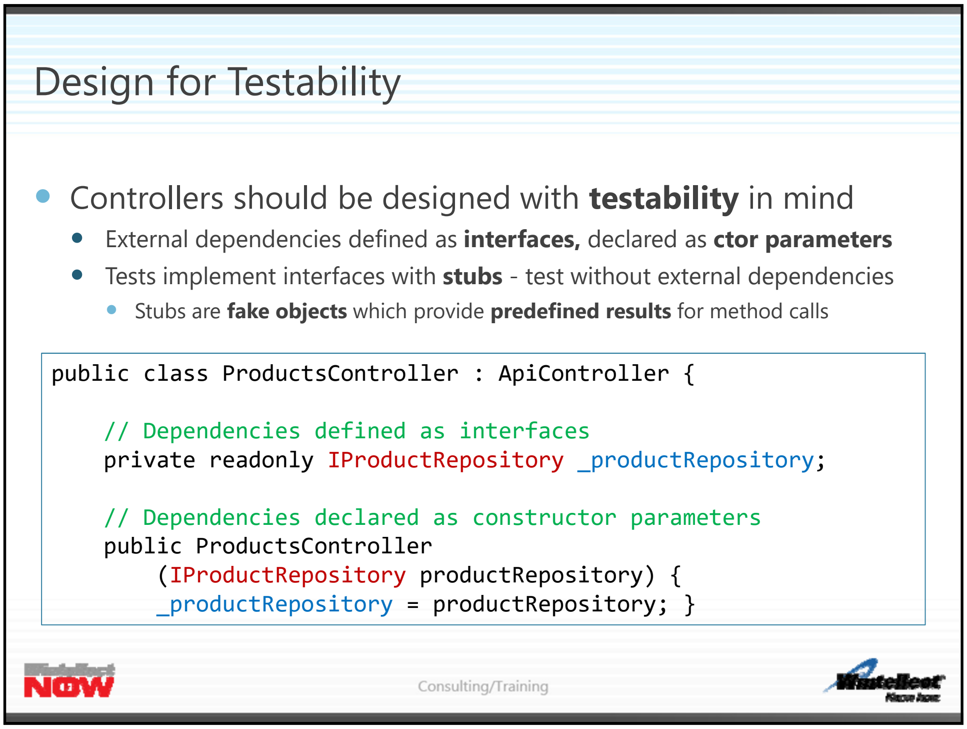
Passed Tests (1)
GetById 52 ms
Not Run Tests (5)
Delete
Get
Index
Post
Put

GetById
Source: ValuesControllerTest.cs line 34
Test Passed - GetById
Elapsed time: 52 ms



xUnit.net runner for Visual Studio
xUnit.net is a unit testing tool for the .NET Framework. Written by the original inventor of NUnit, xUnit.net is t...

ReSharper
The legendary .NET productivity tool: find and fix errors and code smells; navigate and refactor; run unit tests and write quality code faster.

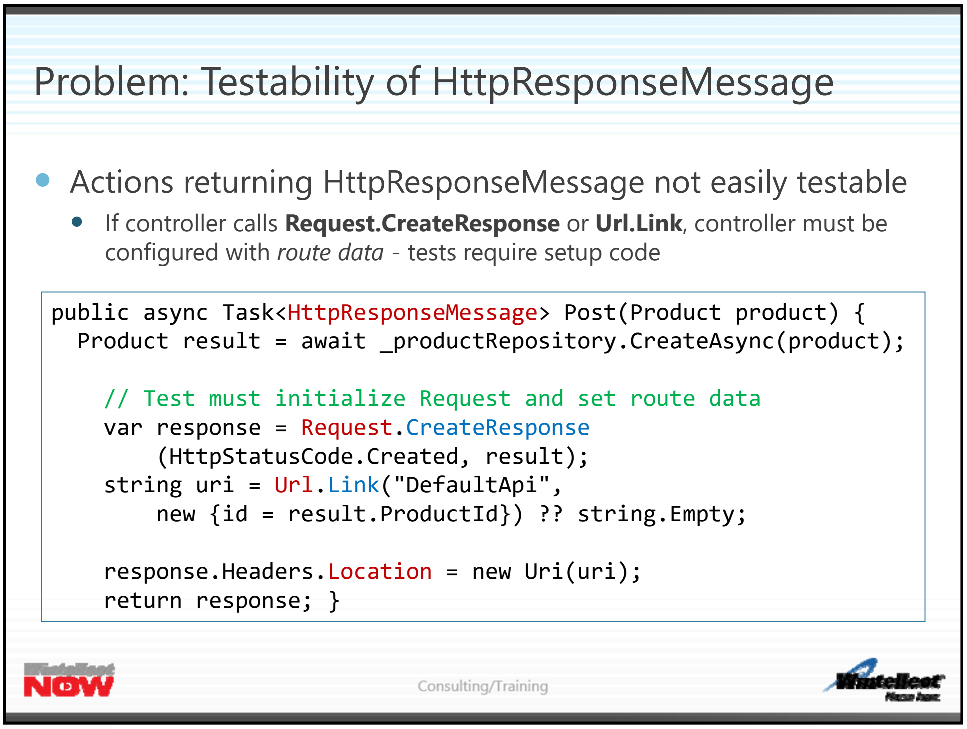


Design for Testability

- Controllers should be designed with **testability** in mind
  - External dependencies defined as **interfaces,** declared as **ctor parameters**
  - Tests implement interfaces with **stubs** - test without external dependencies
    - Stubs are **fake objects** which provide **predefined results** for method calls

```
public class ProductsController : ApiController {

    // Dependencies defined as interfaces
    private readonly IProductRepository _productRepository;

    // Dependencies declared as constructor parameters
    public ProductsController
        (IProductRepository productRepository) {
        _productRepository = productRepository; }
```

Consulting/Training

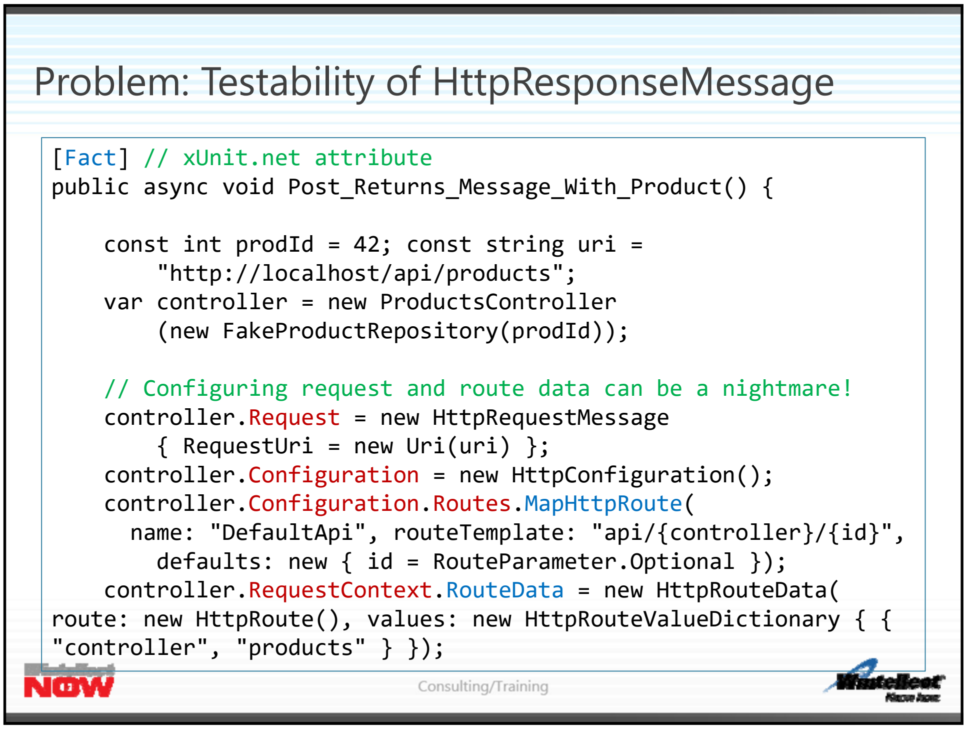


Problem: Testability of HttpResponseMessage

- Actions returning HttpResponseMessage not easily testable
  - If controller calls **Request.CreateResponse** or **Url.Link**, controller must be configured with *route data* - tests require setup code

```
public async Task<HttpResponseMessage> Post(Product product) {
  Product result = await _productRepository.CreateAsync(product);

    // Test must initialize Request and set route data
    var response = Request.CreateResponse
        (HttpStatusCode.Created, result);
    string uri = Url.Link("DefaultApi",
        new {id = result.ProductId}) ?? string.Empty;

    response.Headers.Location = new Uri(uri);
    return response; }
```

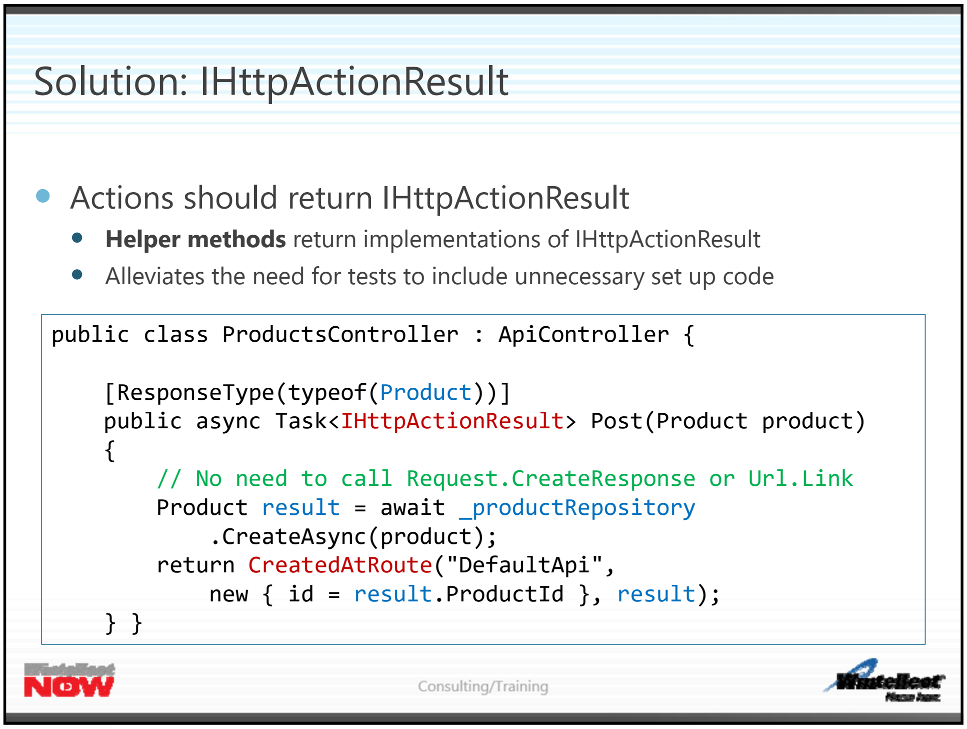
Consulting/Training



Problem: Testability of HttpResponseMessage

```
[Fact] // xUnit.net attribute
public async void Post_Returns_Message_With_Product() {

    const int prodId = 42; const string uri =
        "http://localhost/api/products";
    var controller = new ProductsController
        (new FakeProductRepository(prodId));

    // Configuring request and route data can be a nightmare!
    controller.Request = new HttpRequestMessage
        { RequestUri = new Uri(uri) };
    controller.Configuration = new HttpConfiguration();
    controller.Configuration.Routes.MapHttpRoute(
      name: "DefaultApi", routeTemplate: "api/{controller}/{id}",
        defaults: new { id = RouteParameter.Optional });
    controller.RequestContext.RouteData = new HttpRouteData(
route: new HttpRoute(), values: new HttpRouteValueDictionary { {
"controller", "products" } });
```

Consulting/Training

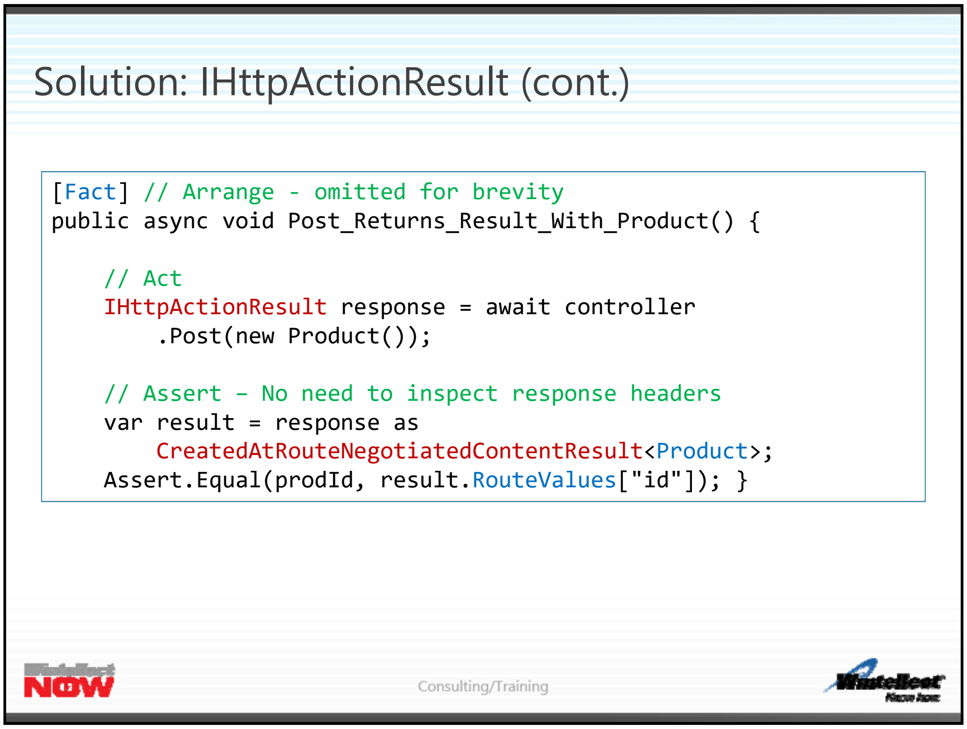


Solution: IHttpActionResult

- Actions should return IHttpActionResult
  - **Helper methods** return implementations of IHttpActionResult
  - Alleviates the need for tests to include unnecessary set up code

```
public class ProductsController : ApiController {

    [ResponseType(typeof(Product))]
    public async Task<IHttpActionResult> Post(Product product)
    {
        // No need to call Request.CreateResponse or Url.Link
        Product result = await _productRepository
            .CreateAsync(product);
        return CreatedAtRoute("DefaultApi",
            new { id = result.ProductId }, result);
    } }
```

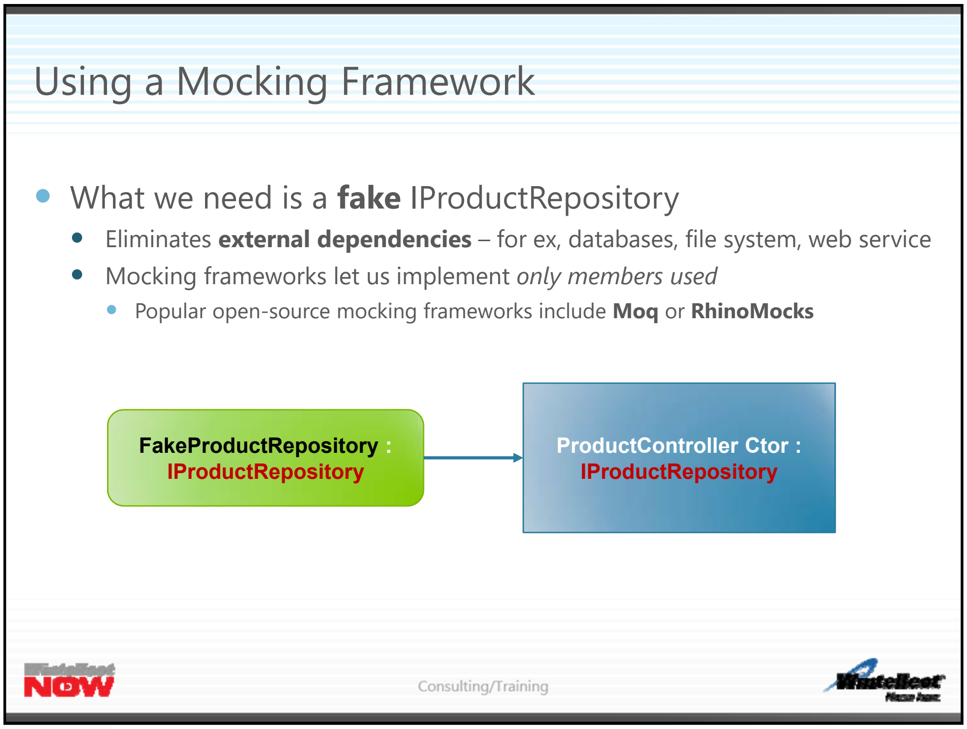
Consulting/Training



Solution: IHttpActionResult (cont.)

```
[Fact] // Arrange - omitted for brevity
public async void Post_Returns_Result_With_Product() {

    // Act
    IHttpActionResult response = await controller
        .Post(new Product());

    // Assert – No need to inspect response headers
    var result = response as
        CreatedAtRouteNegotiatedContentResult<Product>;
    Assert.Equal(prodId, result.RouteValues["id"]); }
```

Consulting/Training

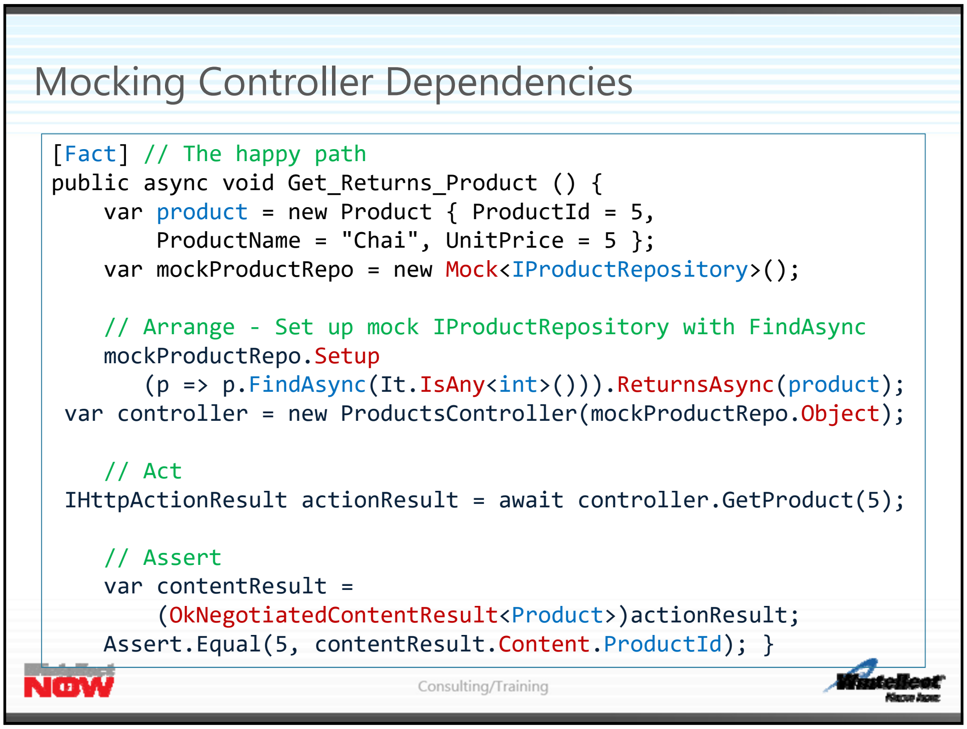


Using a Mocking Framework

- What we need is a **fake** IProductRepository
  - Eliminates **external dependencies** – for ex, databases, file system, web service
  - Mocking frameworks let us implement *only members used*
    - Popular open-source mocking frameworks include **Moq** or **RhinoMocks**

FakeProductRepository :
IProductRepository
ProductController Ctor :
IProductRepository

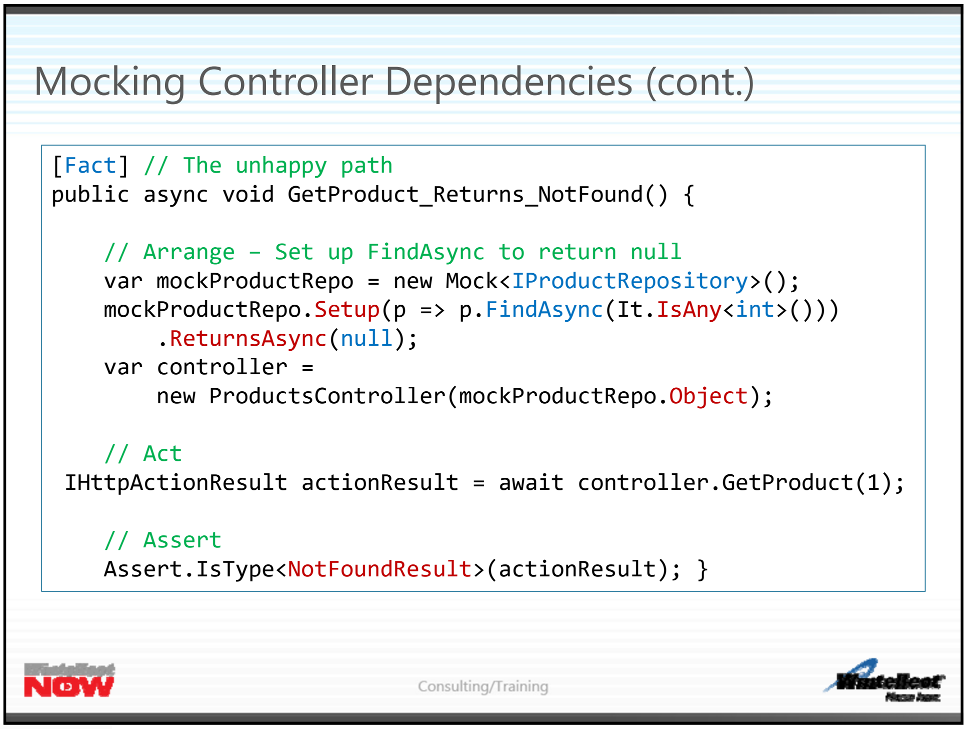
Consulting/Training



Mocking Controller Dependencies

```
[Fact] // The happy path
public async void Get_Returns_Product () {
    var product = new Product { ProductId = 5,
        ProductName = "Chai", UnitPrice = 5 };
    var mockProductRepo = new Mock<IProductRepository>();

    // Arrange - Set up mock IProductRepository with FindAsync
    mockProductRepo.Setup
       (p => p.FindAsync(It.IsAny<int>())).ReturnsAsync(product);
 var controller = new ProductsController(mockProductRepo.Object);

    // Act
 IHttpActionResult actionResult = await controller.GetProduct(5);

    // Assert
    var contentResult =
        (OkNegotiatedContentResult<Product>)actionResult;
    Assert.Equal(5, contentResult.Content.ProductId); }
```

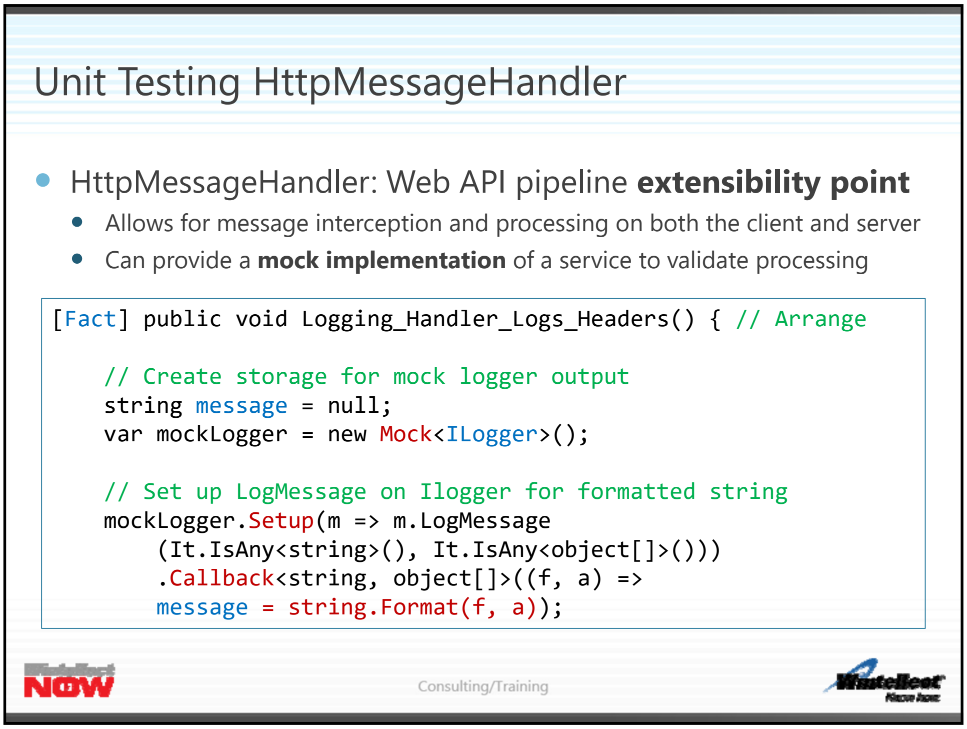
Consulting/Training



Mocking Controller Dependencies (cont.)

```
[Fact] // The unhappy path
public async void GetProduct_Returns_NotFound() {

    // Arrange – Set up FindAsync to return null
    var mockProductRepo = new Mock<IProductRepository>();
    mockProductRepo.Setup(p => p.FindAsync(It.IsAny<int>()))
        .ReturnsAsync(null);
    var controller =
        new ProductsController(mockProductRepo.Object);

    // Act
 IHttpActionResult actionResult = await controller.GetProduct(1);

    // Assert
    Assert.IsType<NotFoundResult>(actionResult); }
```

Consulting/Training

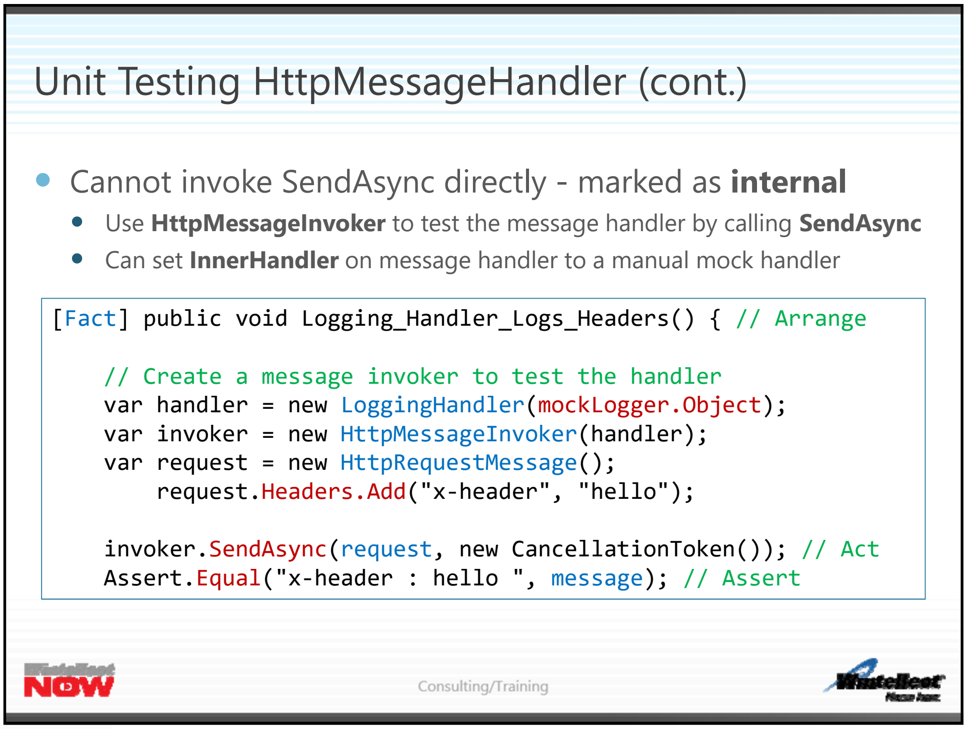


Unit Testing HttpMessageHandler

- HttpMessageHandler: Web API pipeline **extensibility point**
  - Allows for message interception and processing on both the client and server
  - Can provide a **mock implementation** of a service to validate processing

```
[Fact] public void Logging_Handler_Logs_Headers() { // Arrange

    // Create storage for mock logger output
    string message = null;
    var mockLogger = new Mock<ILogger>();

    // Set up LogMessage on Ilogger for formatted string
    mockLogger.Setup(m => m.LogMessage
        (It.IsAny<string>(), It.IsAny<object[]>()))
        .Callback<string, object[]>((f, a) =>
        message = string.Format(f, a));
```

Consulting/Training

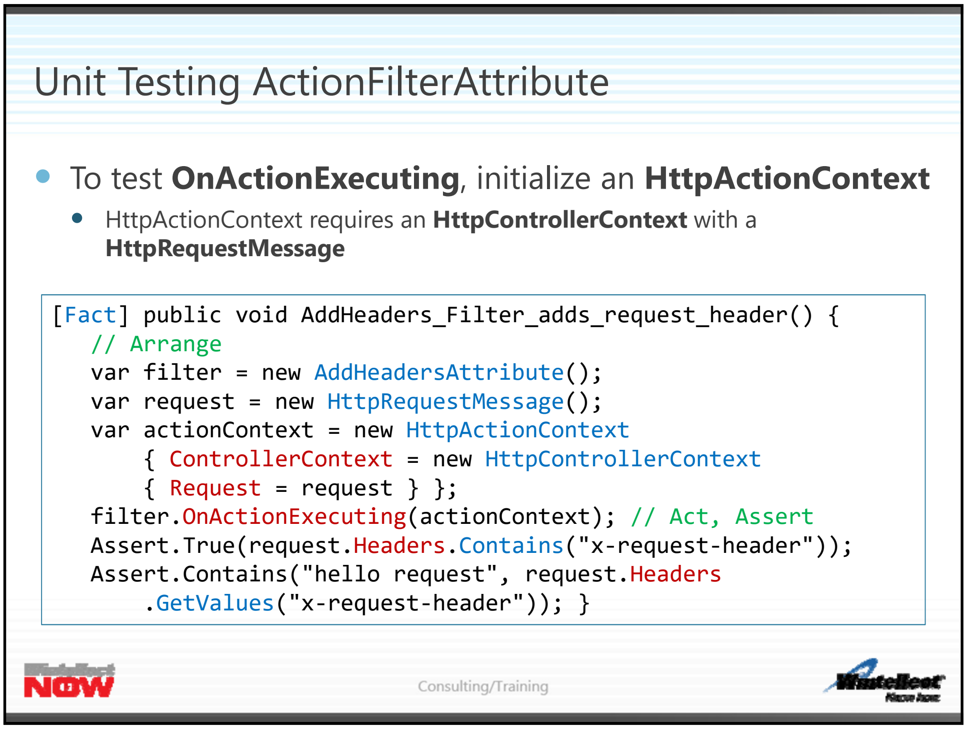


Unit Testing HttpMessageHandler (cont.)

- Cannot invoke SendAsync directly - marked as **internal**
  - Use **HttpMessageInvoker** to test the message handler by calling **SendAsync**
  - Can set **InnerHandler** on message handler to a manual mock handler

```
[Fact] public void Logging_Handler_Logs_Headers() { // Arrange

    // Create a message invoker to test the handler
    var handler = new LoggingHandler(mockLogger.Object);
    var invoker = new HttpMessageInvoker(handler);
    var request = new HttpRequestMessage();
        request.Headers.Add("x-header", "hello");

    invoker.SendAsync(request, new CancellationToken()); // Act
    Assert.Equal("x-header : hello ", message); // Assert
```

Consulting/Training

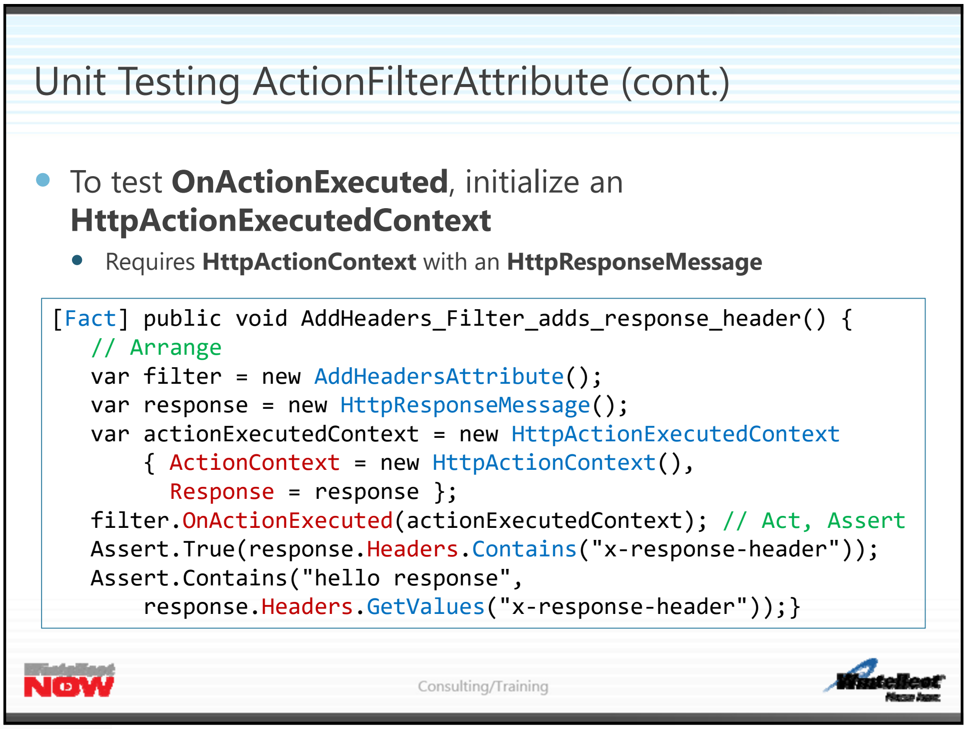


Unit Testing ActionFilterAttribute

- To test **OnActionExecuting**, initialize an **HttpActionContext**
  - HttpActionContext requires an **HttpControllerContext** with a **HttpRequestMessage**

```
[Fact] public void AddHeaders_Filter_adds_request_header() {
   // Arrange
   var filter = new AddHeadersAttribute();
   var request = new HttpRequestMessage();
   var actionContext = new HttpActionContext
       { ControllerContext = new HttpControllerContext
       { Request = request } };
   filter.OnActionExecuting(actionContext); // Act, Assert
   Assert.True(request.Headers.Contains("x-request-header"));
   Assert.Contains("hello request", request.Headers
       .GetValues("x-request-header")); }
```

Consulting/Training

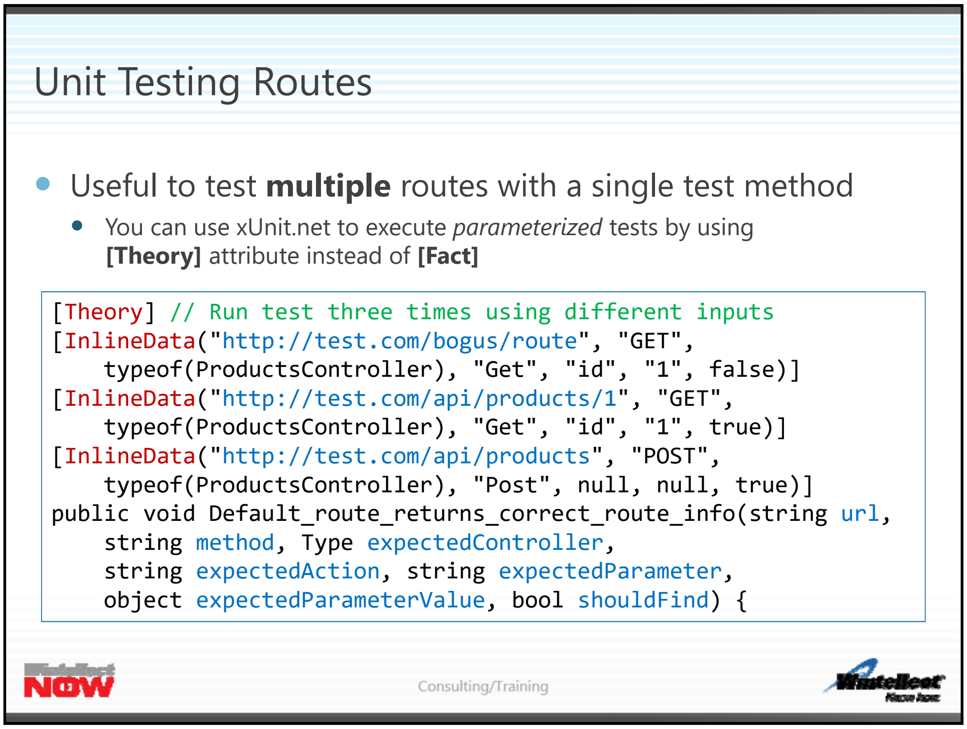


Unit Testing ActionFilterAttribute (cont.)

- To test **OnActionExecuted**, initialize an **HttpActionExecutedContext**
  - Requires **HttpActionContext** with an **HttpResponseMessage**

```
[Fact] public void AddHeaders_Filter_adds_response_header() {
   // Arrange
   var filter = new AddHeadersAttribute();
   var response = new HttpResponseMessage();
   var actionExecutedContext = new HttpActionExecutedContext
       { ActionContext = new HttpActionContext(),
         Response = response };
   filter.OnActionExecuted(actionExecutedContext); // Act, Assert
   Assert.True(response.Headers.Contains("x-response-header"));
   Assert.Contains("hello response",
       response.Headers.GetValues("x-response-header"));}
```

Consulting/Training

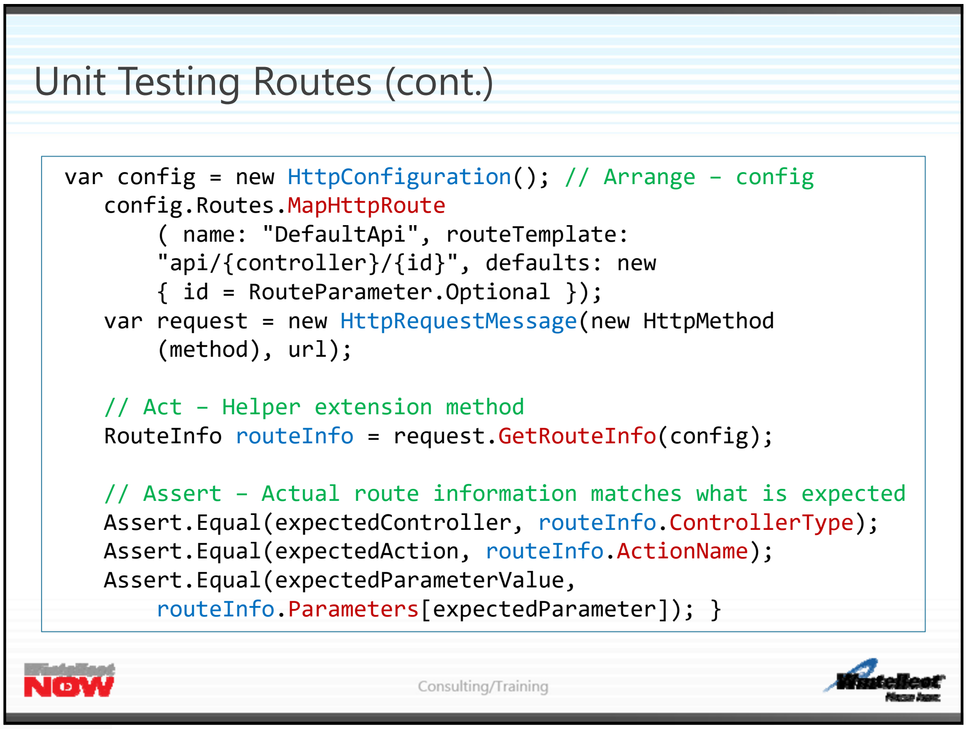


Unit Testing Routes

- Useful to test **multiple** routes with a single test method
  - You can use xUnit.net to execute *parameterized* tests by using **[Theory]** attribute instead of **[Fact]**

```
[Theory] // Run test three times using different inputs
[InlineData("http://test.com/bogus/route", "GET",
    typeof(ProductsController), "Get", "id", "1", false)]
[InlineData("http://test.com/api/products/1", "GET",
    typeof(ProductsController), "Get", "id", "1", true)]
[InlineData("http://test.com/api/products", "POST",
    typeof(ProductsController), "Post", null, null, true)]
public void Default_route_returns_correct_route_info(string url,
    string method, Type expectedController,
    string expectedAction, string expectedParameter,
    object expectedParameterValue, bool shouldFind) {
```

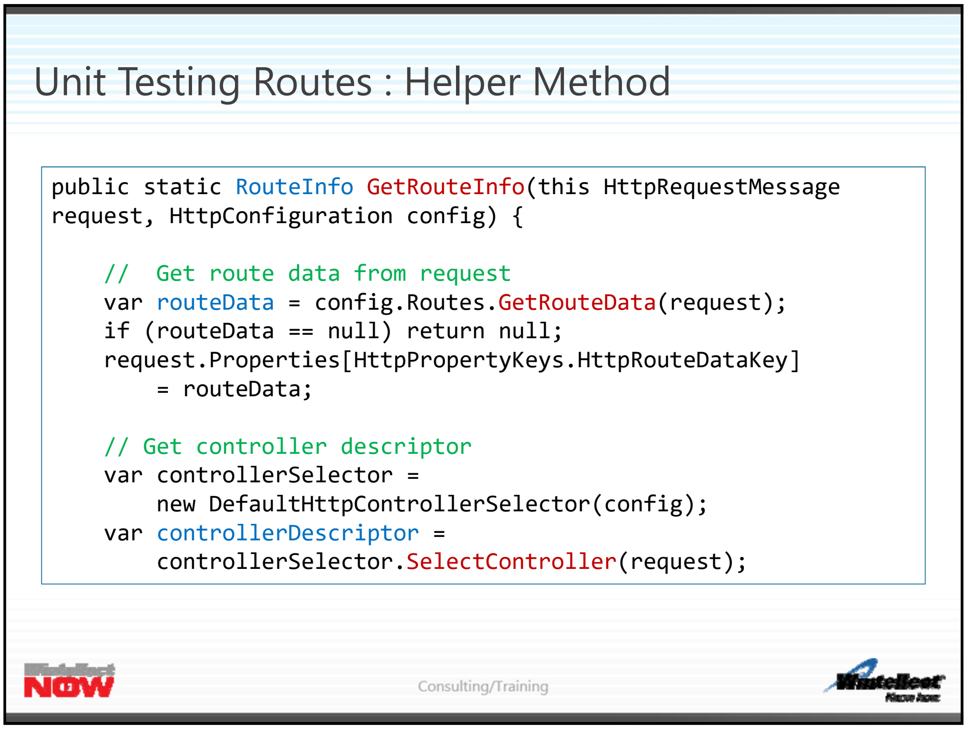
Consulting/Training



Unit Testing Routes (cont.)

```
 var config = new HttpConfiguration(); // Arrange - config
    config.Routes.MapHttpRoute
        ( name: "DefaultApi", routeTemplate:
        "api/{controller}/{id}", defaults: new
        { id = RouteParameter.Optional });
    var request = new HttpRequestMessage(new HttpMethod
        (method), url);

    // Act - Helper extension method
    RouteInfo routeInfo = request.GetRouteInfo(config);

    // Assert - Actual route information matches what is expected
    Assert.Equal(expectedController, routeInfo.ControllerType);
    Assert.Equal(expectedAction, routeInfo.ActionName);
    Assert.Equal(expectedParameterValue,
        routeInfo.Parameters[expectedParameter]); }
```

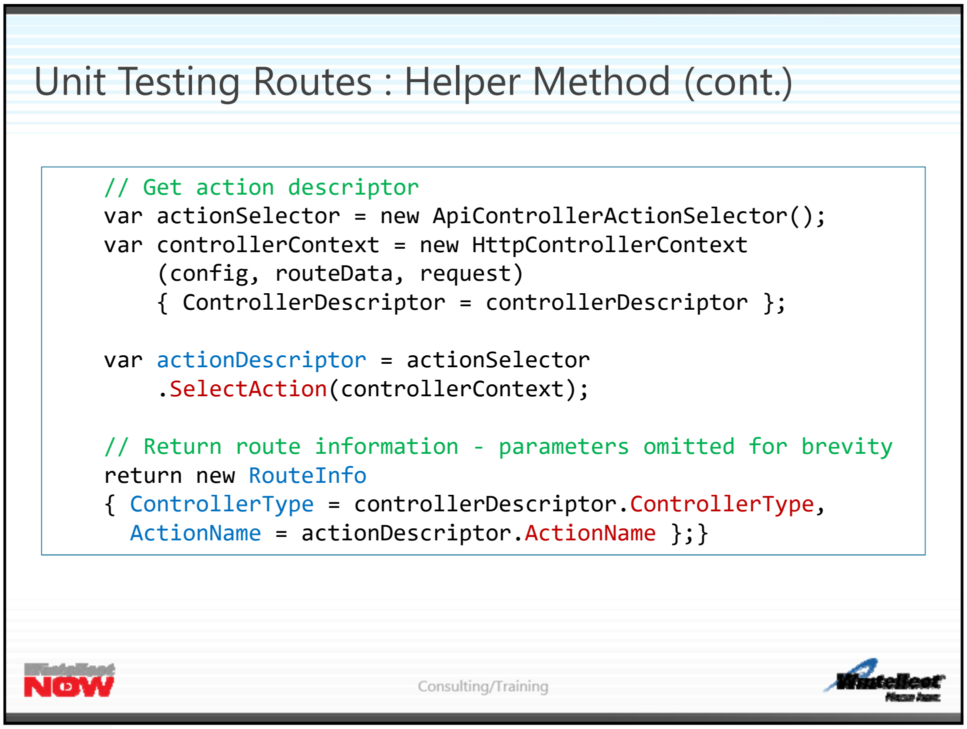
Consulting/Training



Unit Testing Routes : Helper Method

```
public static RouteInfo GetRouteInfo(this HttpRequestMessage
request, HttpConfiguration config) {

    //  Get route data from request
    var routeData = config.Routes.GetRouteData(request);
    if (routeData == null) return null;
    request.Properties[HttpPropertyKeys.HttpRouteDataKey]
        = routeData;

    // Get controller descriptor
    var controllerSelector =
        new DefaultHttpControllerSelector(config);
    var controllerDescriptor =
        controllerSelector.SelectController(request);
```

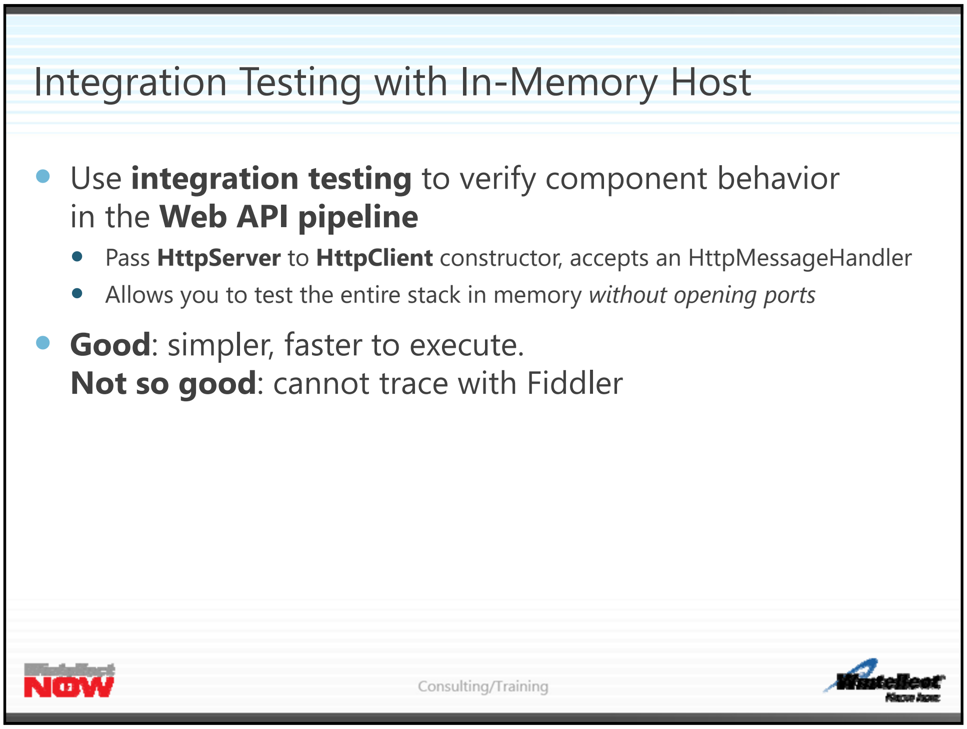
Consulting/Training



Unit Testing Routes : Helper Method (cont.)

```
    // Get action descriptor
    var actionSelector = new ApiControllerActionSelector();
    var controllerContext = new HttpControllerContext
        (config, routeData, request)
        { ControllerDescriptor = controllerDescriptor };

    var actionDescriptor = actionSelector
        .SelectAction(controllerContext);

    // Return route information - parameters omitted for brevity
    return new RouteInfo
    { ControllerType = controllerDescriptor.ControllerType,
      ActionName = actionDescriptor.ActionName };}
```

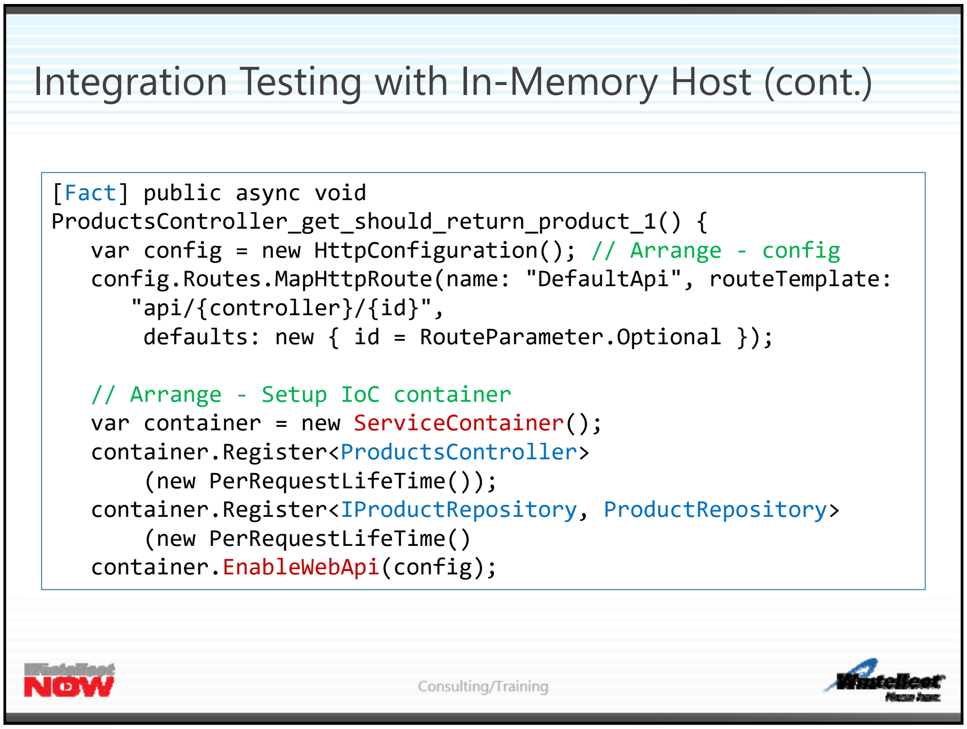
Consulting/Training



Integration Testing with In-Memory Host

- Use **integration testing** to verify component behavior in the **Web API pipeline**
  - Pass **HttpServer** to **HttpClient** constructor, accepts an HttpMessageHandler
  - Allows you to test the entire stack in memory *without opening ports*
- **Good**: simpler, faster to execute.
  Not so good: cannot trace with Fiddler

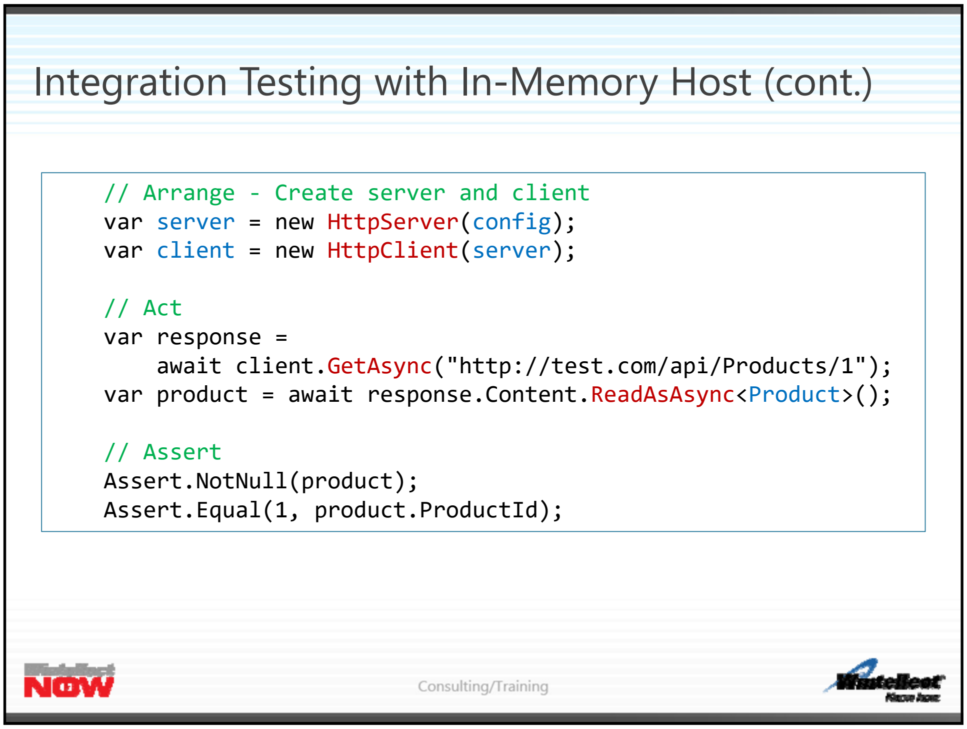
Consulting/Training



Integration Testing with In-Memory Host (cont.)

```
[Fact] public async void
ProductsController_get_should_return_product_1() {
   var config = new HttpConfiguration(); // Arrange - config
   config.Routes.MapHttpRoute(name: "DefaultApi", routeTemplate:
      "api/{controller}/{id}",
       defaults: new { id = RouteParameter.Optional });

   // Arrange - Setup IoC container
   var container = new ServiceContainer();
   container.Register<ProductsController>
       (new PerRequestLifeTime());
   container.Register<IProductRepository, ProductRepository>
       (new PerRequestLifeTime()
   container.EnableWebApi(config);
```

Consulting/Training



Integration Testing with In-Memory Host (cont.)

```
    // Arrange - Create server and client
    var server = new HttpServer(config);
    var client = new HttpClient(server);

    // Act
    var response =
        await client.GetAsync("http://test.com/api/Products/1");
    var product = await response.Content.ReadAsAsync<Product>();

    // Assert
    Assert.NotNull(product);
    Assert.Equal(1, product.ProductId);
```

Consulting/Training

QUESTIONS

Consulting/Training

Validation and Testing with ASP.NET Web API

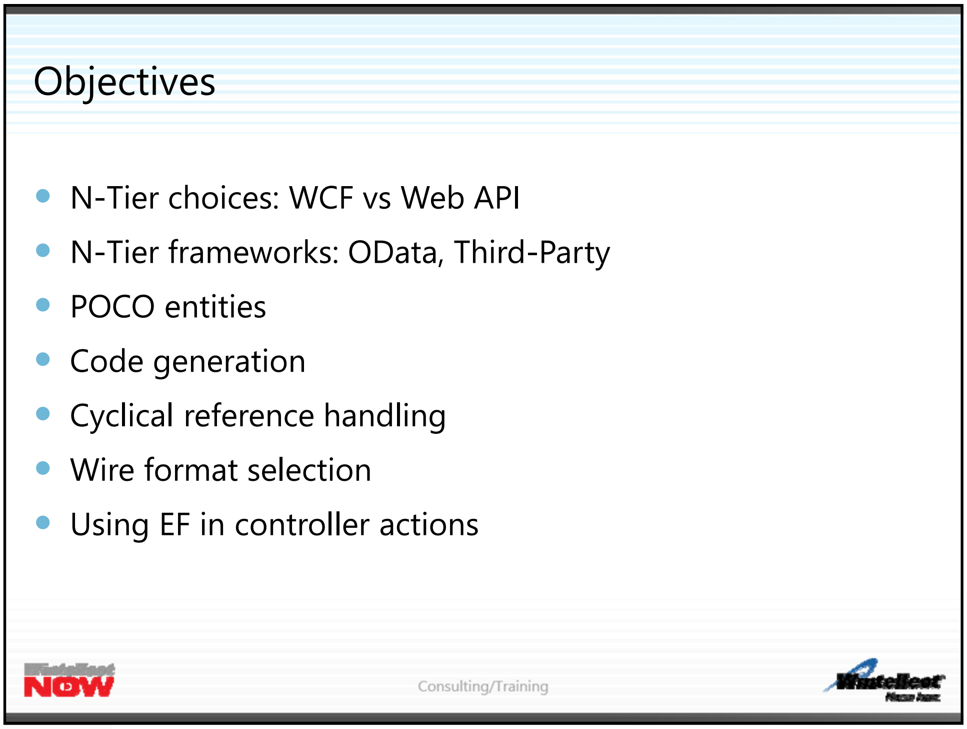
Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

Consulting/Training

Entity Framework in N-Tier Applications

Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

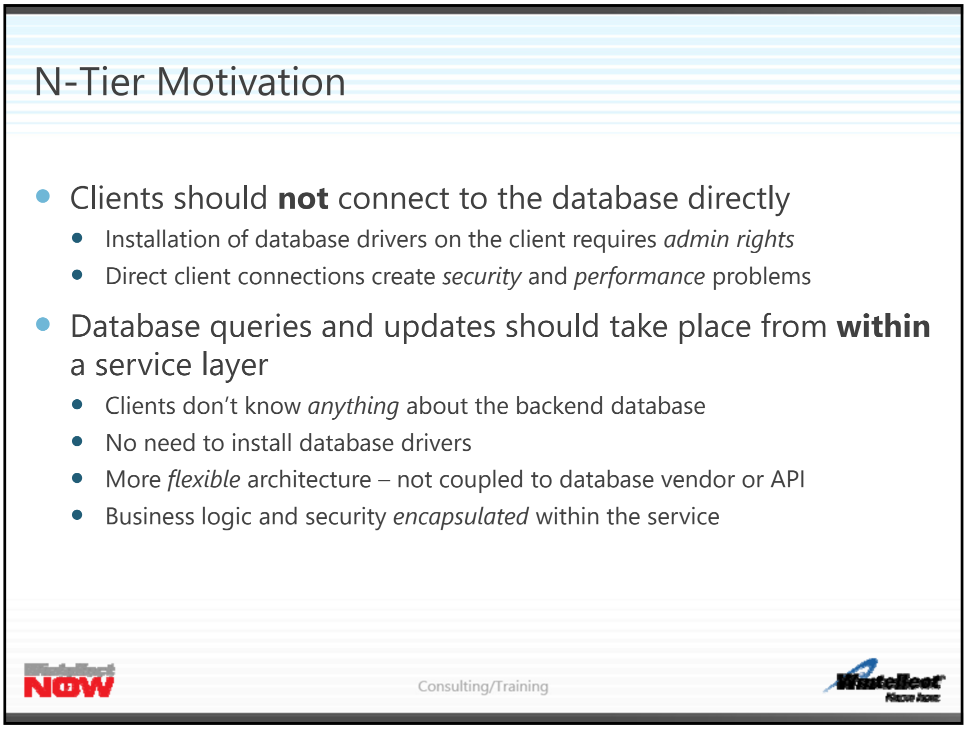
Consulting/Training



Objectives

- N-Tier choices: WCF vs Web API
- N-Tier frameworks: OData, Third-Party
- POCO entities
- Code generation
- Cyclical reference handling
- Wire format selection
- Using EF in controller actions

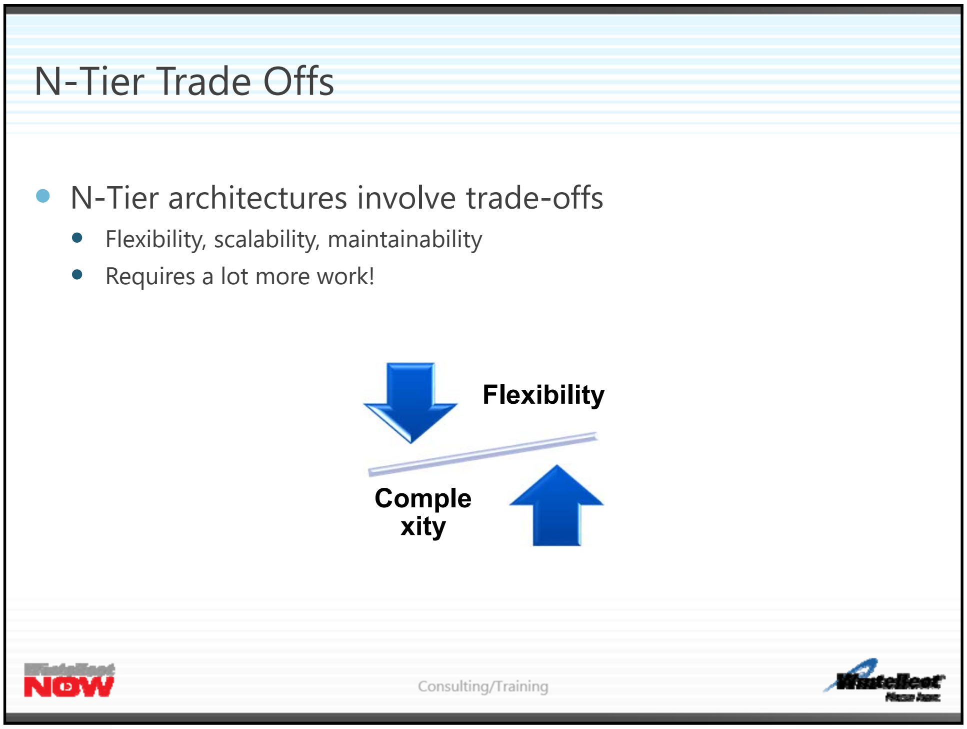
Consulting/Training



N-Tier Motivation

- Clients should **not** connect to the database directly
  - Installation of database drivers on the client requires *admin rights*
  - Direct client connections create *security* and *performance* problems
- Database queries and updates should take place from **within** a service layer
  - Clients don't know *anything* about the backend database
  - No need to install database drivers
  - More *flexible* architecture – not coupled to database vendor or API
  - Business logic and security *encapsulated* within the service

Consulting/Training

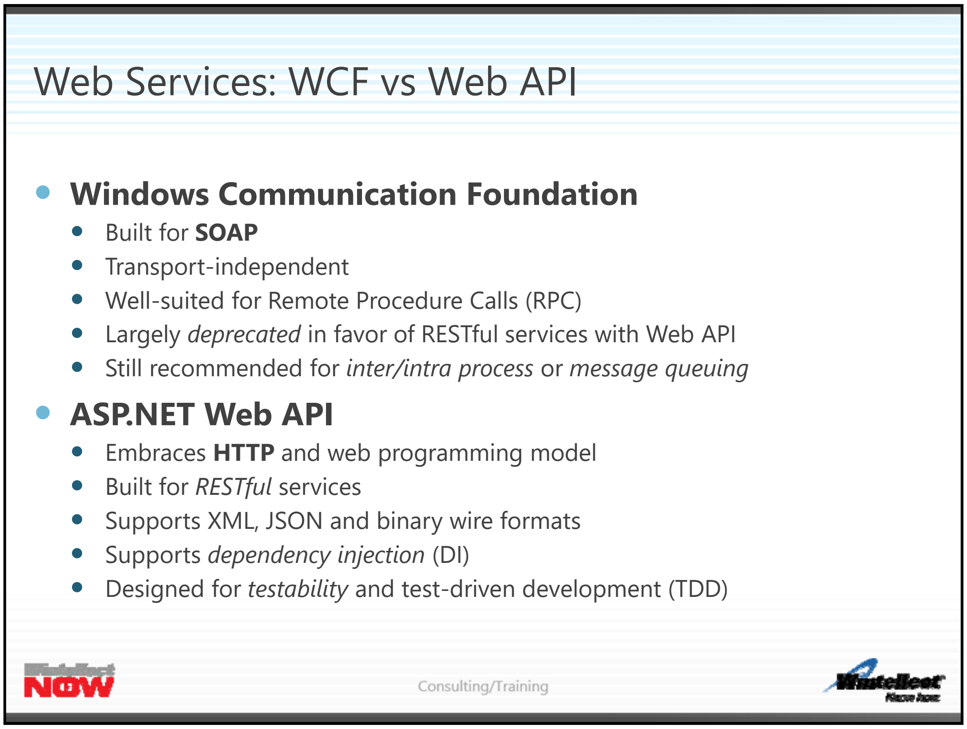


N-Tier Trade Offs

- N-Tier architectures involve trade-offs
  - Flexibility, scalability, maintainability
  - Requires a lot more work!

Flexibility
Comple xity

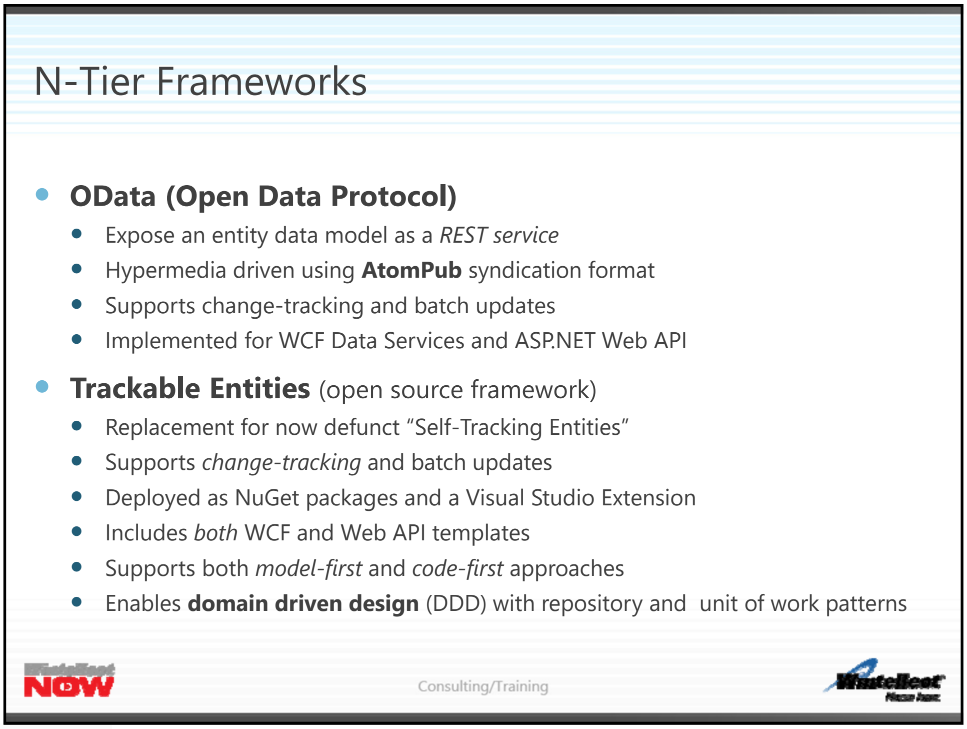
Consulting/Training



Web Services: WCF vs Web API

- **Windows Communication Foundation**
  - Built for **SOAP**
  - Transport-independent
  - Well-suited for Remote Procedure Calls (RPC)
  - Largely *deprecated* in favor of RESTful services with Web API
  - Still recommended for *inter/intra process* or *message queuing*
- **ASP.NET Web API**
  - Embraces **HTTP** and web programming model
  - Built for *RESTful* services
  - Supports XML, JSON and binary wire formats
  - Supports *dependency injection* (DI)
  - Designed for *testability* and test-driven development (TDD)

Consulting/Training



N-Tier Frameworks

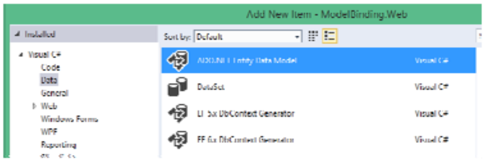
- **OData (Open Data Protocol)**
  - Expose an entity data model as a *REST service*
  - Hypermedia driven using **AtomPub** syndication format
  - Supports change-tracking and batch updates
  - Implemented for WCF Data Services and ASP.NET Web API
- **Trackable Entities** (open source framework)
  - Replacement for now defunct "Self-Tracking Entities"
  - Supports *change-tracking* and batch updates
  - Deployed as NuGet packages and a Visual Studio Extension
  - Includes *both* WCF and Web API templates
  - Supports both *model-first* and *code-first* approaches
  - Enables **domain driven design** (DDD) with repository and unit of work patterns

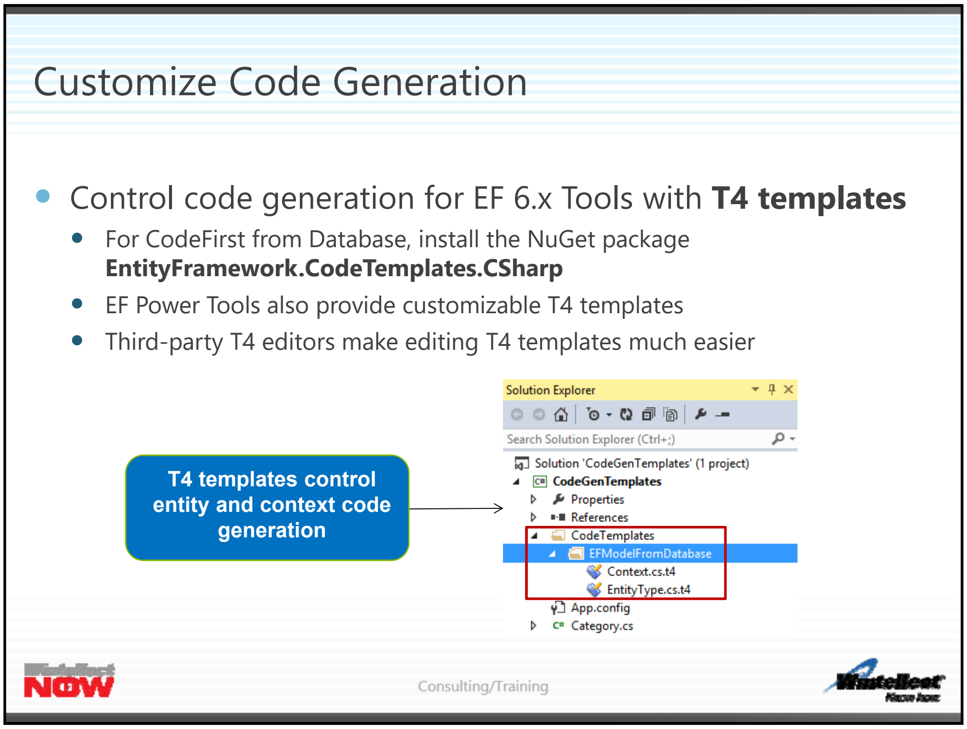
Consulting/Training

Entity Framework and Web API

Separate Entities from EF-Specific Code

- Entities should be **ignorant** of persistence concerns
  - Place entities in a separate (portable) class library project
- May *reverse engineer* entities from an existing database
  - EF 6.x VS Tools: Add **ADO.NET Entity Data Model** to **.Data** project
    - EF Designer from Database
    - Code First from Database
  - Copy or "Add As Link" from **.Data** to **.Entities** project
  - Or use *EF Power Tools* to reverse engineer Code First classes





Customize Code Generation

- Control code generation for EF 6.x Tools with **T4 templates**
  - For CodeFirst from Database, install the NuGet package **EntityFramework.CodeTemplates.CSharp**
  - EF Power Tools also provide customizable T4 templates
  - Third-party T4 editors make editing T4 templates much easier

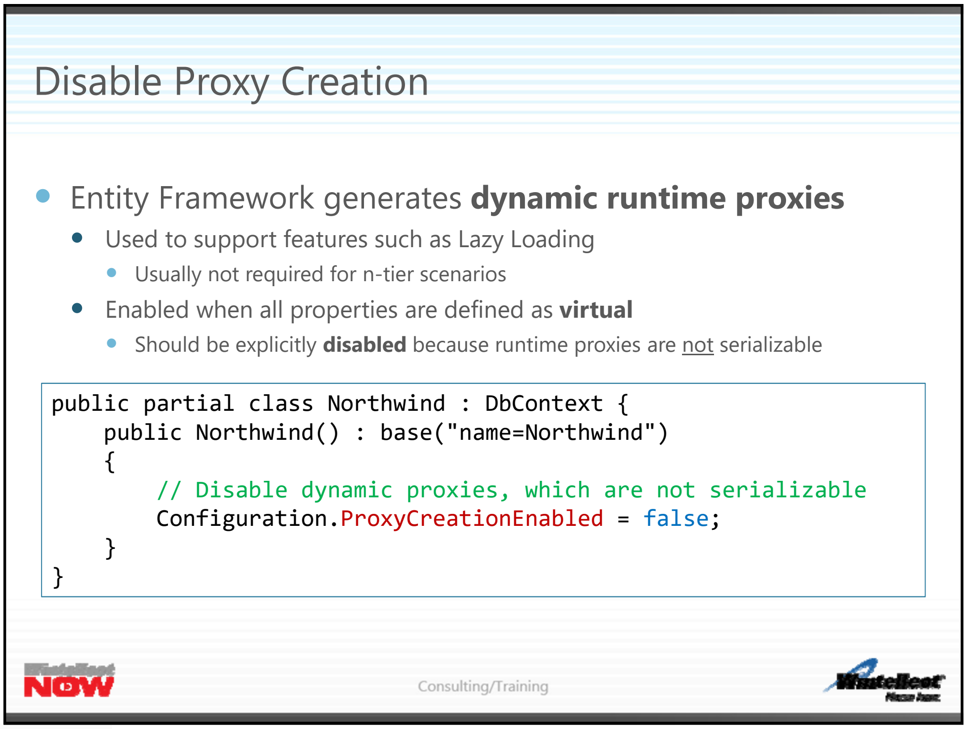
T4 templates control entity and context code generation

Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'CodeGenTemplates' (1 project)
  CodeGenTemplates
    Properties
    References
    CodeTemplates
      EFModelFromDatabase
        Context.cs.t4
        EntityType.cs.t4
    App.config
    Category.cs
```

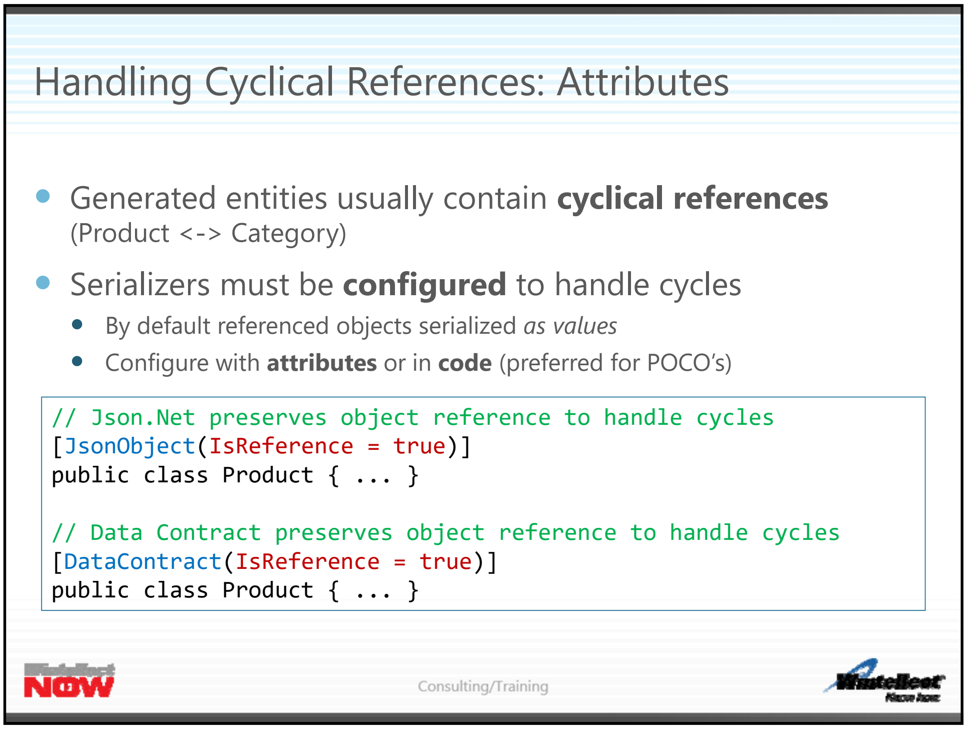
Consulting/Training



Disable Proxy Creation

- Entity Framework generates **dynamic runtime proxies**
  - Used to support features such as Lazy Loading
    - Usually not required for n-tier scenarios
  - Enabled when all properties are defined as **virtual**
    - Should be explicitly **disabled** because runtime proxies are not serializable

```
public partial class Northwind : DbContext {
    public Northwind() : base("name=Northwind")
    {
        // Disable dynamic proxies, which are not serializable
        Configuration.ProxyCreationEnabled = false;
    }
}
```

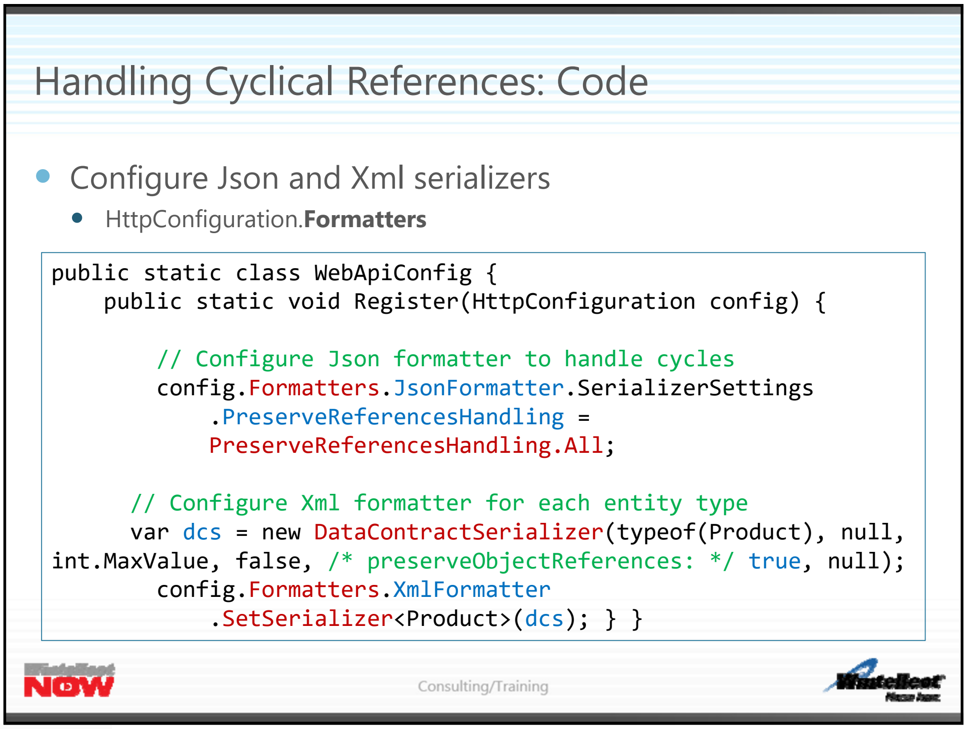


Handling Cyclical References: Attributes

- Generated entities usually contain **cyclical references** (Product <-> Category)
- Serializers must be **configured** to handle cycles
  - By default referenced objects serialized *as values*
  - Configure with **attributes** or in **code** (preferred for POCO's)

```
// Json.Net preserves object reference to handle cycles
[JsonObject(IsReference = true)]
public class Product { ... }

// Data Contract preserves object reference to handle cycles
[DataContract(IsReference = true)]
public class Product { ... }
```

Consulting/Training

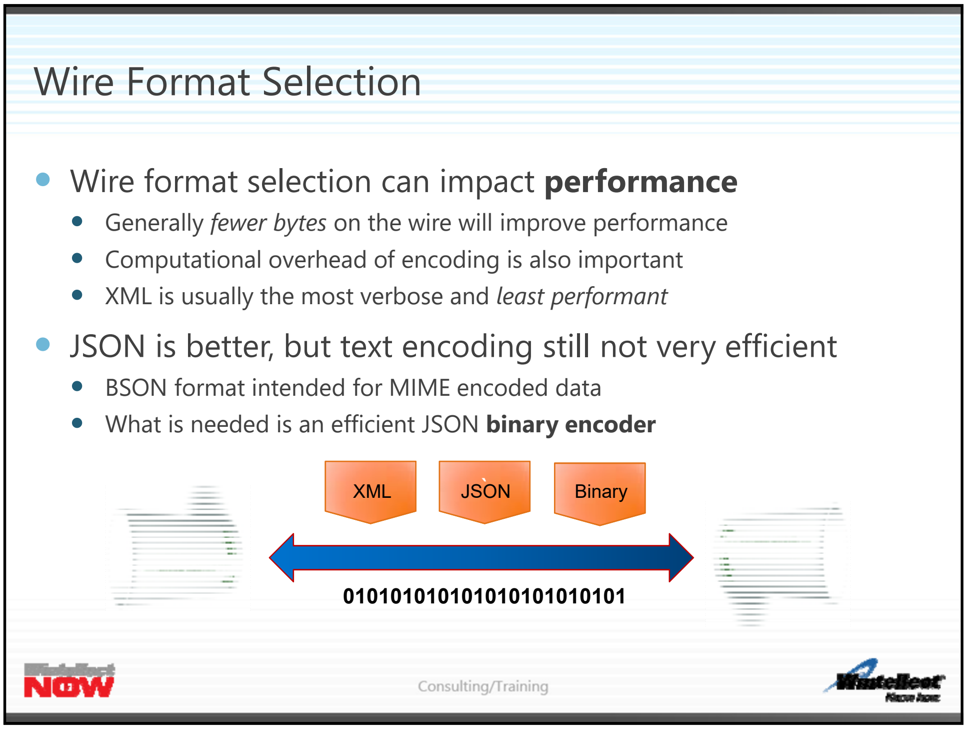


Handling Cyclical References: Code

- Configure Json and Xml serializers
  - HttpConfiguration.**Formatters**

```
public static class WebApiConfig {
    public static void Register(HttpConfiguration config) {

        // Configure Json formatter to handle cycles
        config.Formatters.JsonFormatter.SerializerSettings
            .PreserveReferencesHandling =
            PreserveReferencesHandling.All;

      // Configure Xml formatter for each entity type
      var dcs = new DataContractSerializer(typeof(Product), null,
int.MaxValue, false, /* preserveObjectReferences: */ true, null);
        config.Formatters.XmlFormatter
            .SetSerializer<Product>(dcs); } }
```

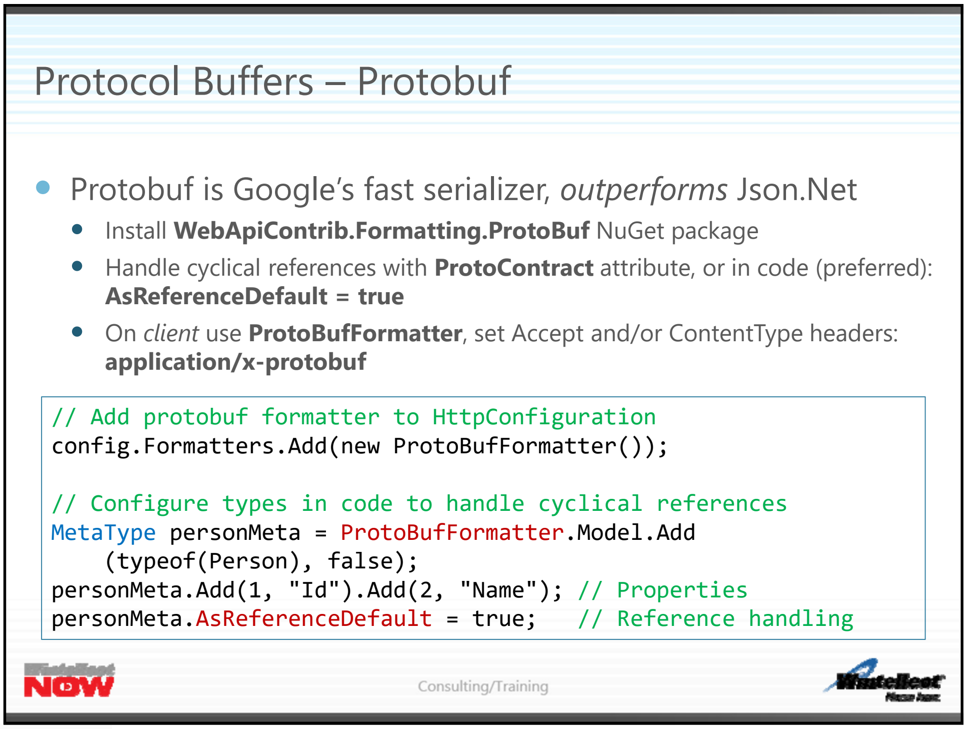
Consulting/Training



Wire Format Selection

- Wire format selection can impact **performance**
  - Generally *fewer bytes* on the wire will improve performance
  - Computational overhead of encoding is also important
  - XML is usually the most verbose and *least performant*
- JSON is better, but text encoding still not very efficient
  - BSON format intended for MIME encoded data
  - What is needed is an efficient JSON **binary encoder**

Consulting/Training



Protocol Buffers – Protobuf

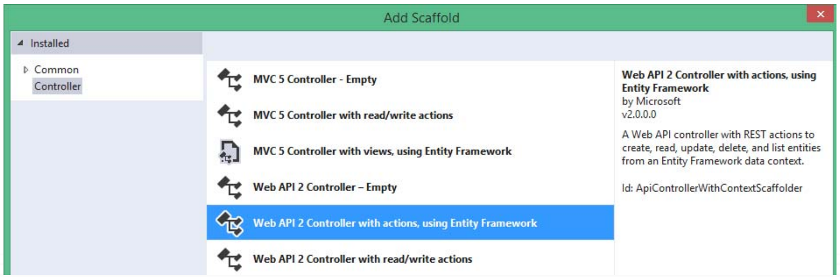
- Protobuf is Google's fast serializer, *outperforms* Json.Net
  - Install **WebApiContrib.Formatting.ProtoBuf** NuGet package
  - Handle cyclical references with **ProtoContract** attribute, or in code (preferred): **AsReferenceDefault = true**
  - On *client* use **ProtoBufFormatter**, set Accept and/or ContentType headers: **application/x-protobuf**

```
// Add protobuf formatter to HttpConfiguration
config.Formatters.Add(new ProtoBufFormatter());

// Configure types in code to handle cyclical references
MetaType personMeta = ProtoBufFormatter.Model.Add
    (typeof(Person), false);
personMeta.Add(1, "Id").Add(2, "Name"); // Properties
personMeta.AsReferenceDefault = true;   // Reference handling
```

Consulting/Training

Add Controller with Actions using EF

- There is **tooling support** in VS for using EF with Web API
  - Don’t take as gospel but use as a starting point
  - For example, first Get method is not async, returns IQueryable
  - Modify queries to *eager load* related entities via **Include** operator


Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions

Web API 2 Controller with actions, using Entity Framework
by Microsoft
v2.0.0.0

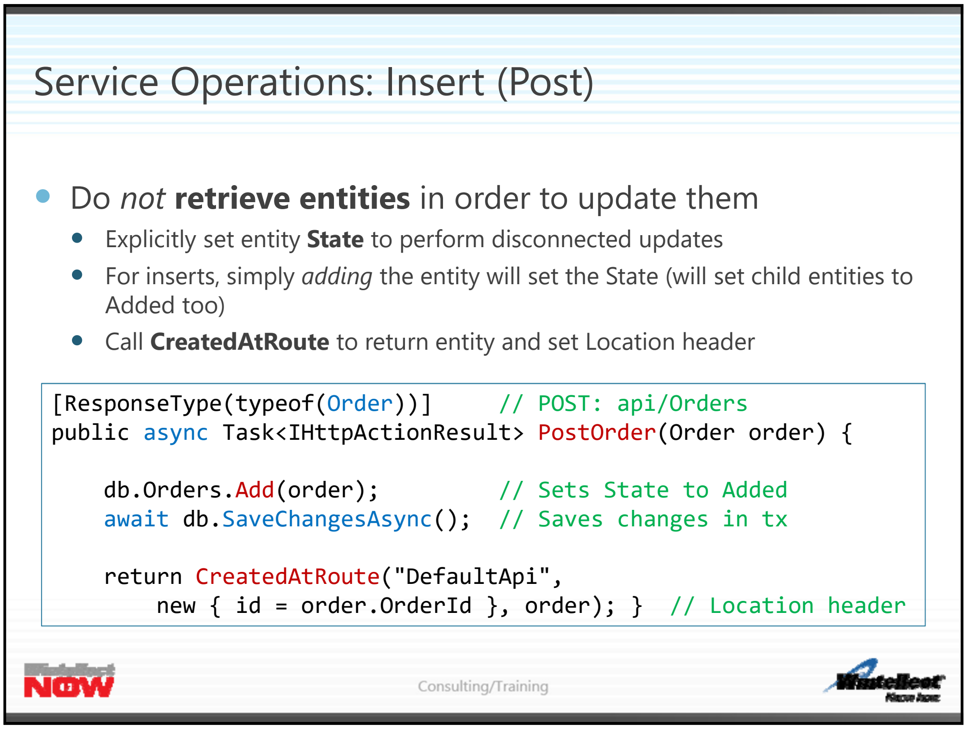
A Web API controller with REST actions to create, read, update, delete, and list entities from an Entity Framework data context.

Id: ApiControllerWithContextScaffolder

Service Operations: Retrieve (Get)

- Get everything you need in **one trip**
  - Prefer *chunky* over *chatty* database calls
  - Use **async** API for greater scalability
  - **Include** operator has overloads accepting lambda expression (single property) and string (nested properties)

```
[ResponseType(typeof(IEnumerable<Order>))]  // GET: api/Orders
public async Task<IHttpActionResult> GetOrders() {

    var orders = await (from o in db.Orders
       .Include(o => o.Customer)         // Load Customer property
       .Include("OrderDetails.Product") // Details, Products
       select o).ToListAsync();
    return Ok(orders); }
```

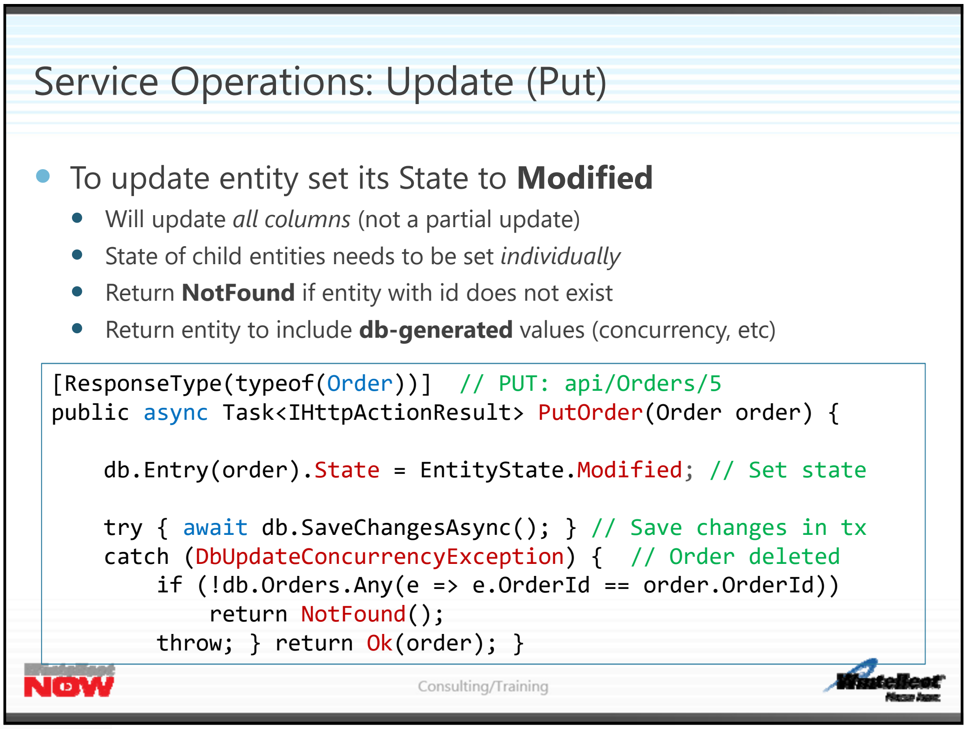


Service Operations: Insert (Post)

- Do *not* **retrieve entities** in order to update them
  - Explicitly set entity **State** to perform disconnected updates
  - For inserts, simply *adding* the entity will set the State (will set child entities to Added too)
  - Call **CreatedAtRoute** to return entity and set Location header

```
[ResponseType(typeof(Order))]      // POST: api/Orders
public async Task<IHttpActionResult> PostOrder(Order order) {

    db.Orders.Add(order);          // Sets State to Added
    await db.SaveChangesAsync();   // Saves changes in tx

    return CreatedAtRoute("DefaultApi",
        new { id = order.OrderId }, order); }  // Location header
```

Consulting/Training

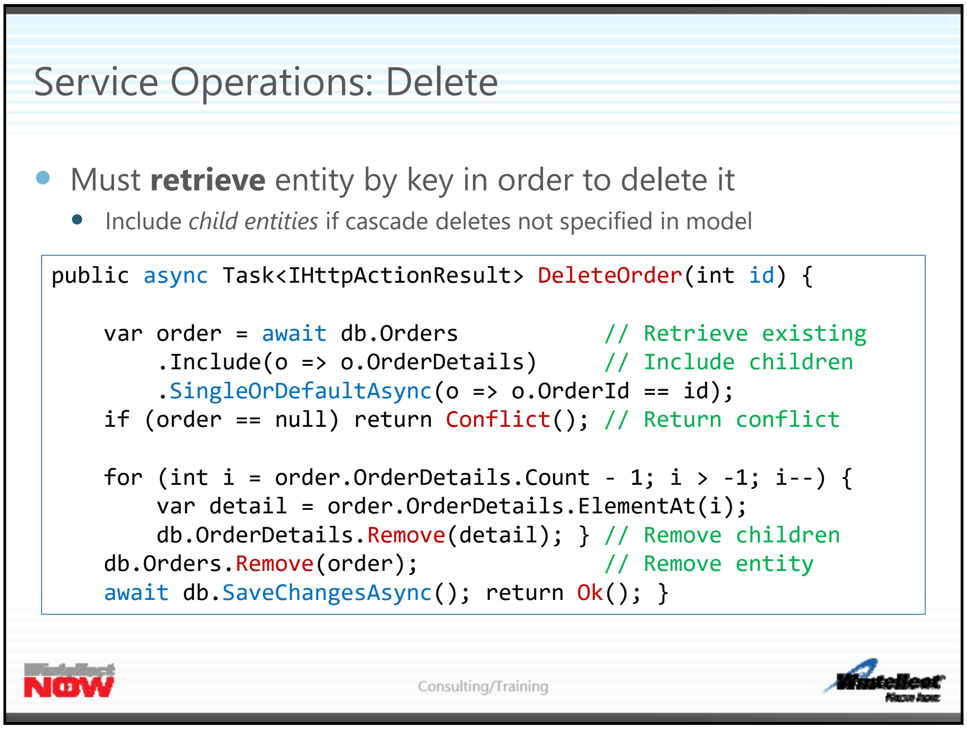


Service Operations: Update (Put)

- To update entity set its State to **Modified**
  - Will update *all columns* (not a partial update)
  - State of child entities needs to be set *individually*
  - Return **NotFound** if entity with id does not exist
  - Return entity to include **db-generated** values (concurrency, etc)

```
[ResponseType(typeof(Order))]  // PUT: api/Orders/5
public async Task<IHttpActionResult> PutOrder(Order order) {

    db.Entry(order).State = EntityState.Modified; // Set state

    try { await db.SaveChangesAsync(); } // Save changes in tx
    catch (DbUpdateConcurrencyException) {  // Order deleted
        if (!db.Orders.Any(e => e.OrderId == order.OrderId))
            return NotFound();
        throw; } return Ok(order); }
```

Consulting/Training



Service Operations: Delete

- Must **retrieve** entity by key in order to delete it
  - Include *child entities* if cascade deletes not specified in model

```
public async Task<IHttpActionResult> DeleteOrder(int id) {

    var order = await db.Orders           // Retrieve existing
        .Include(o => o.OrderDetails)     // Include children
        .SingleOrDefaultAsync(o => o.OrderId == id);
    if (order == null) return Conflict(); // Return conflict

    for (int i = order.OrderDetails.Count - 1; i > -1; i--) {
        var detail = order.OrderDetails.ElementAt(i);
        db.OrderDetails.Remove(detail); } // Remove children
    db.Orders.Remove(order);              // Remove entity
    await db.SaveChangesAsync(); return Ok(); }
```

Consulting/Training

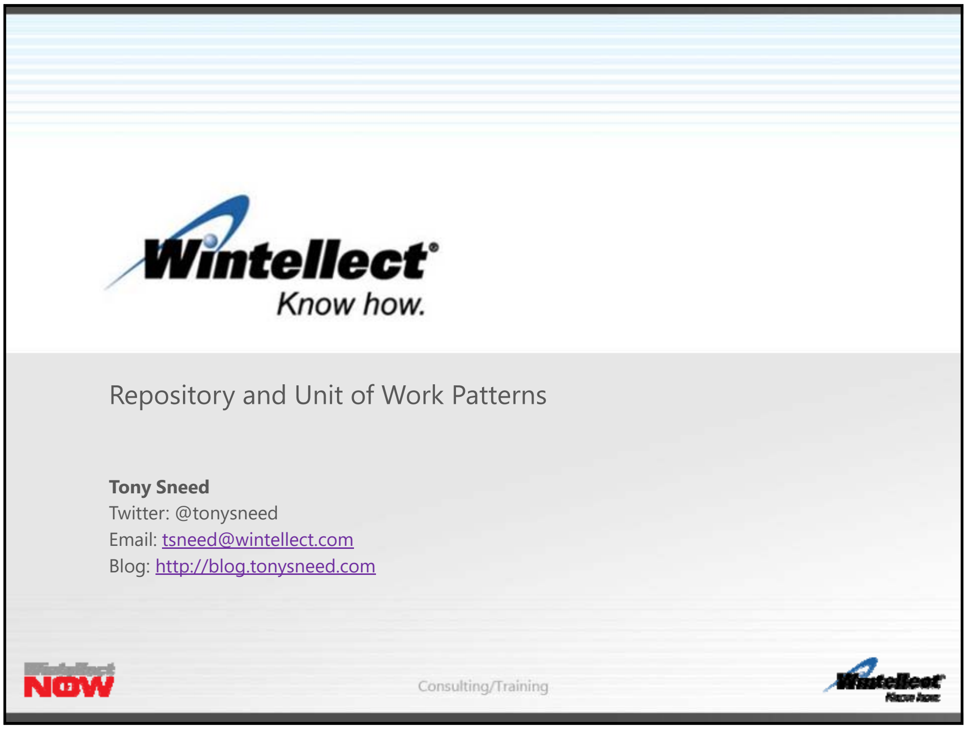
QUESTIONS

Consulting/Training

Entity Framework in N-Tier Applications

Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

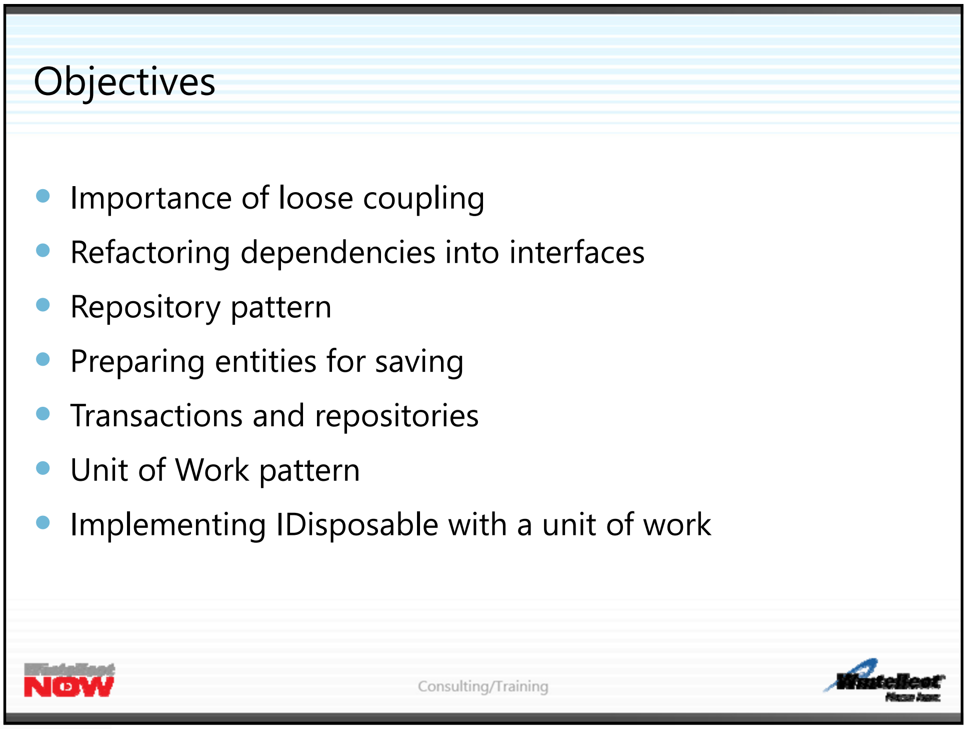
Consulting/Training



Repository and Unit of Work Patterns

Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

Consulting/Training



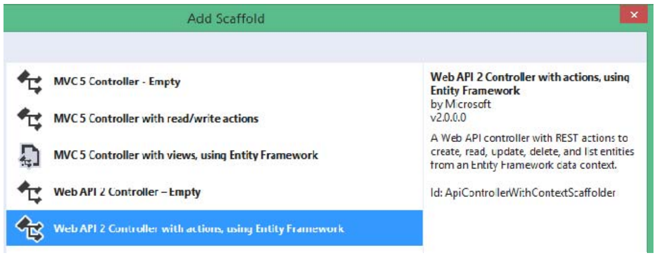
Objectives

- Importance of loose coupling
- Refactoring dependencies into interfaces
- Repository pattern
- Preparing entities for saving
- Transactions and repositories
- Unit of Work pattern
- Implementing IDisposable with a unit of work

Consulting/Training

Problem: Data Access Coupling

- Default Web API scaffolding creates **direct dependency** on Entity Framework

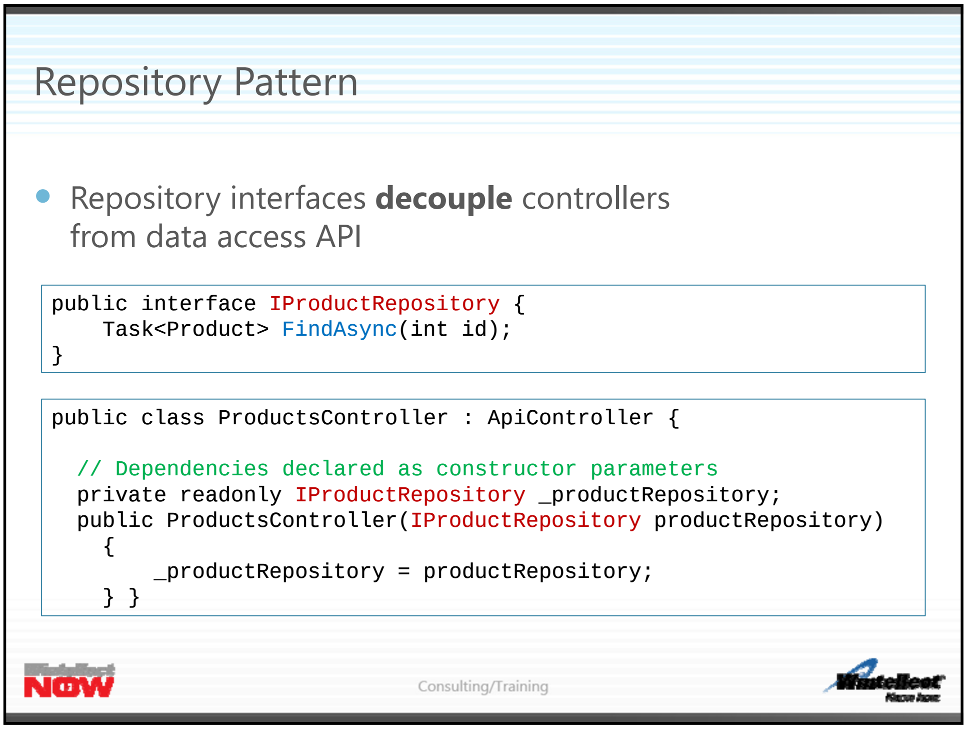

Add Scaffold
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework

Web API 2 Controller with actions, using Entity Framework
by Microsoft
v2.0.0.0

A Web API controller with REST actions to create, read, update, delete, and list entities from an Entity Framework data context.

Id: ApiControllerWithContextScaffolder

Repository Pattern



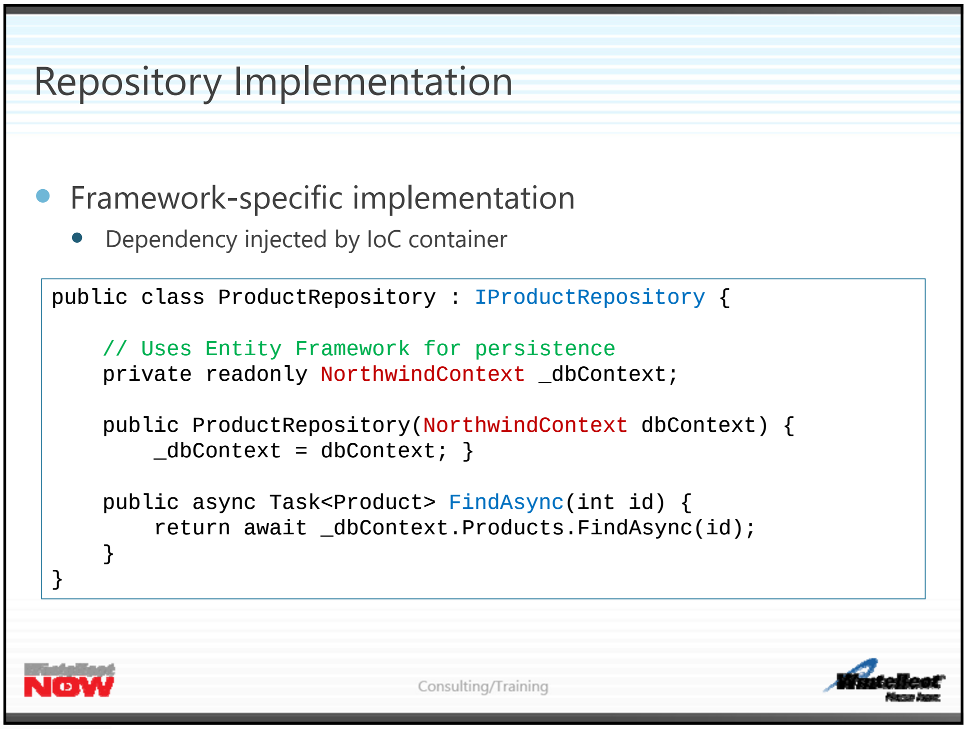
Repository Pattern

- Repository interfaces **decouple** controllers from data access API

```
public interface IProductRepository {
    Task<Product> FindAsync(int id);
}
```

```
public class ProductsController : ApiController {

  // Dependencies declared as constructor parameters
  private readonly IProductRepository _productRepository;
  public ProductsController(IProductRepository productRepository)
    {
        _productRepository = productRepository;
    } }
```

Consulting/Training

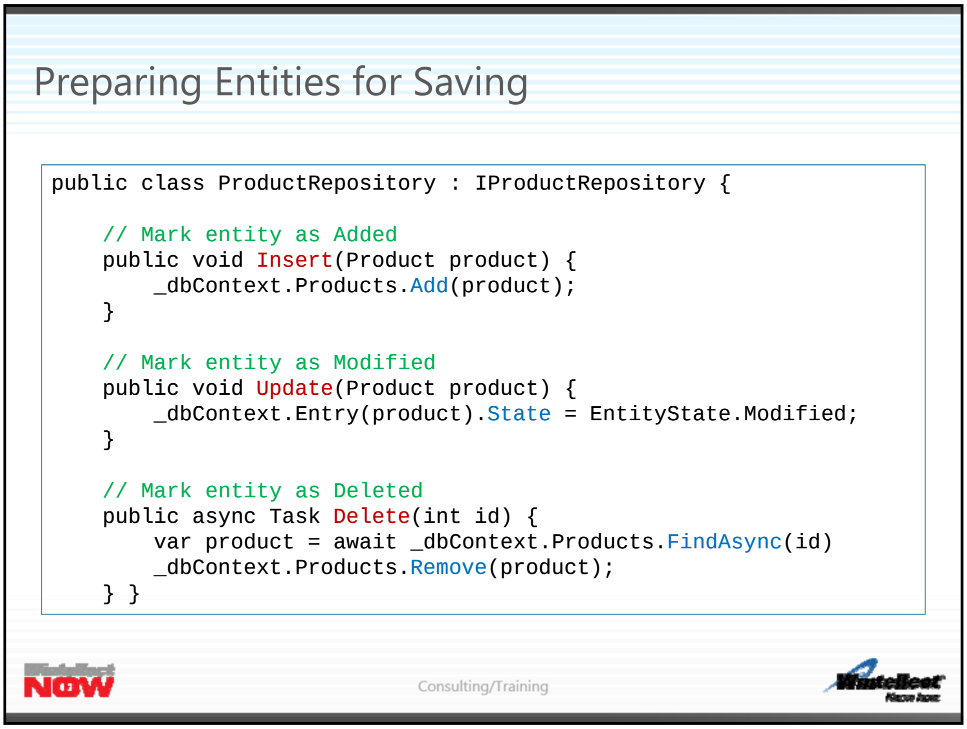


Repository Implementation

- Framework-specific implementation
  - Dependency injected by IoC container

```
public class ProductRepository : IProductRepository {

    // Uses Entity Framework for persistence
    private readonly NorthwindContext _dbContext;

    public ProductRepository(NorthwindContext dbContext) {
        _dbContext = dbContext; }

    public async Task<Product> FindAsync(int id) {
        return await _dbContext.Products.FindAsync(id);
    }
}
```

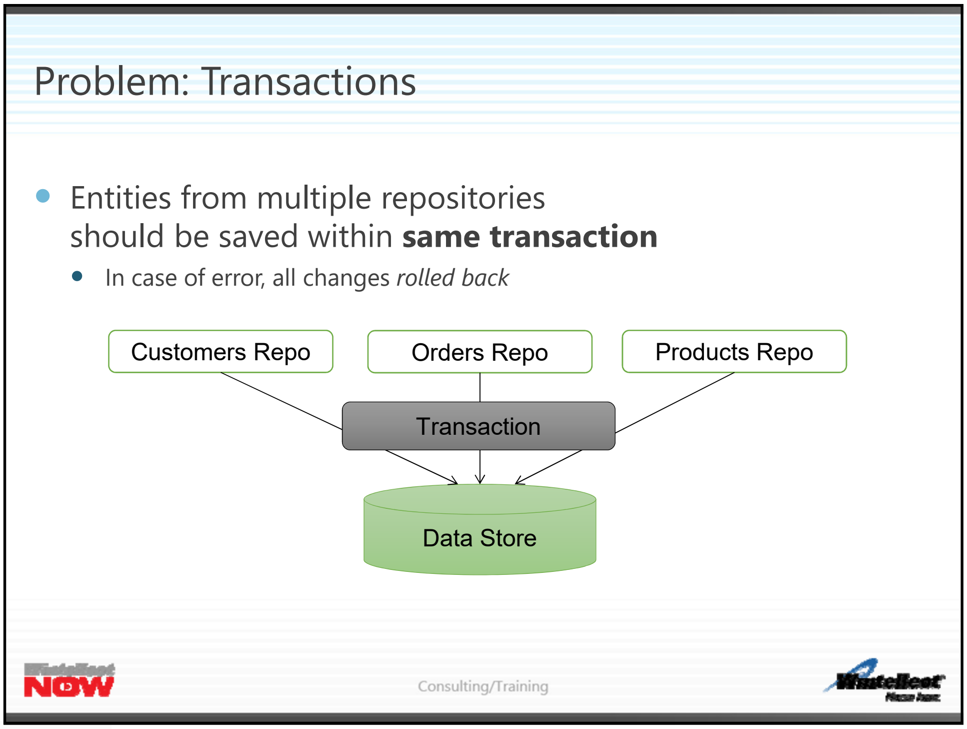
Consulting/Training



Preparing Entities for Saving

```
public class ProductRepository : IProductRepository {

    // Mark entity as Added
    public void Insert(Product product) {
        _dbContext.Products.Add(product);
    }

    // Mark entity as Modified
    public void Update(Product product) {
        _dbContext.Entry(product).State = EntityState.Modified;
    }

    // Mark entity as Deleted
    public async Task Delete(int id) {
        var product = await _dbContext.Products.FindAsync(id)
        _dbContext.Products.Remove(product);
    } }
```

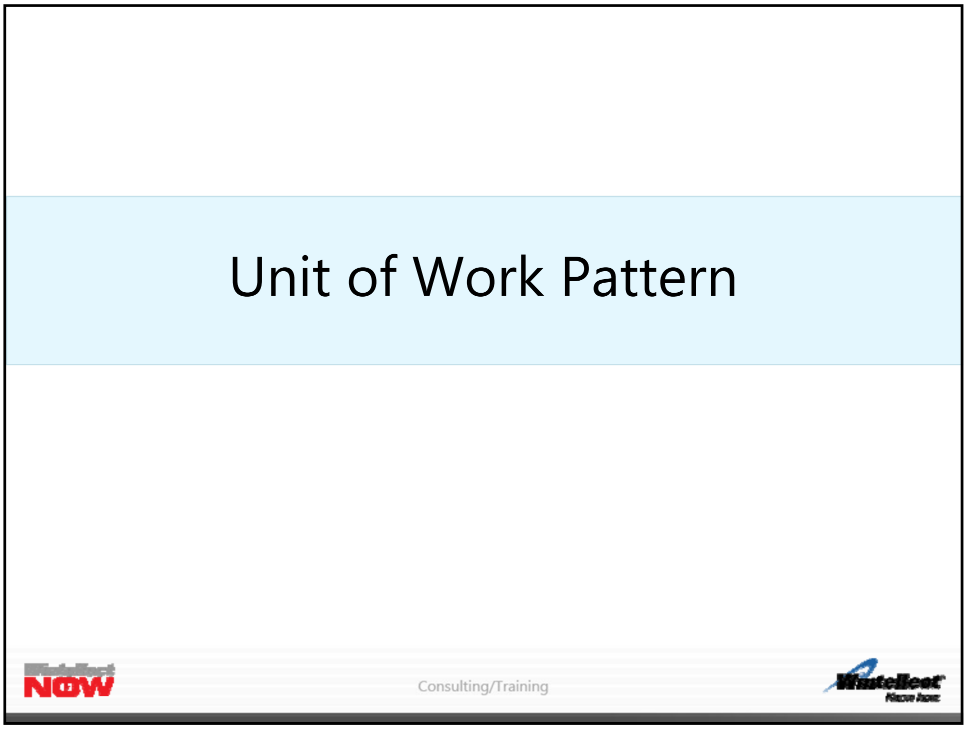
Consulting/Training



Problem: Transactions

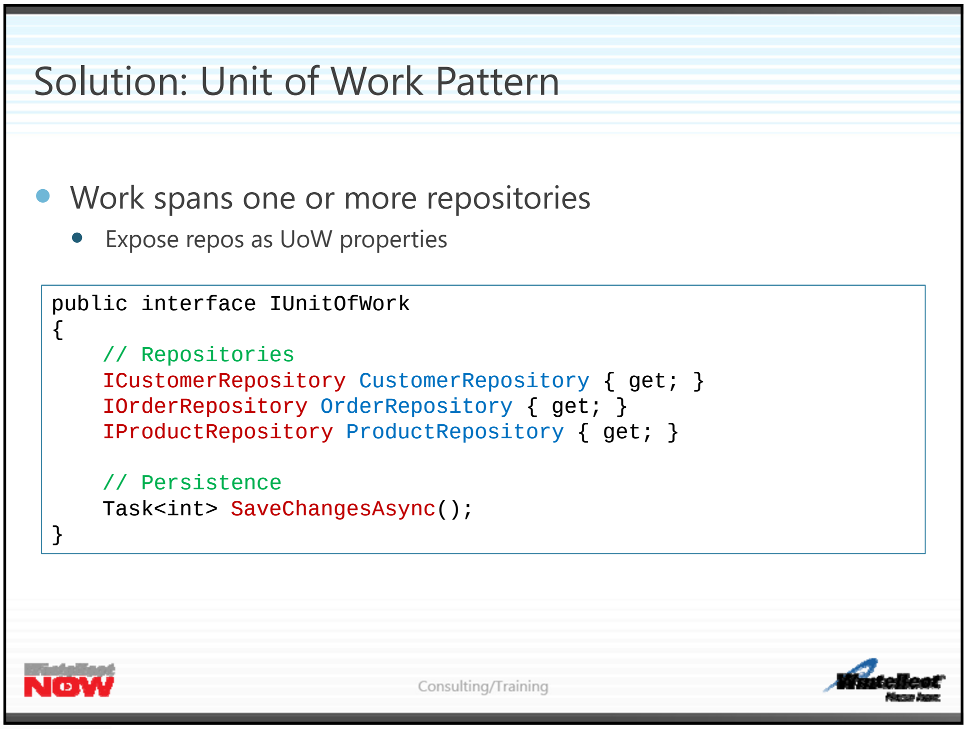
- Entities from multiple repositories should be saved within **same transaction**
  - In case of error, all changes *rolled back*

Consulting/Training



Unit of Work Pattern

Consulting/Training

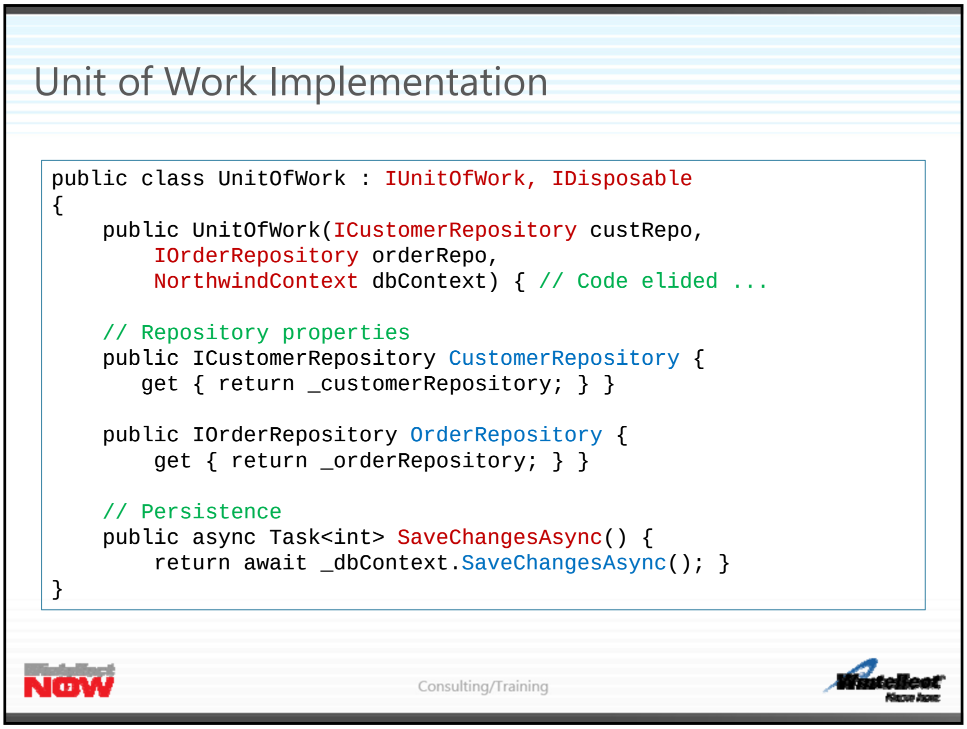


Solution: Unit of Work Pattern

- Work spans one or more repositories
  - Expose repos as UoW properties

```
public interface IUnitOfWork
{
    // Repositories
    ICustomerRepository CustomerRepository { get; }
    IOrderRepository OrderRepository { get; }
    IProductRepository ProductRepository { get; }

    // Persistence
    Task<int> SaveChangesAsync();
}
```

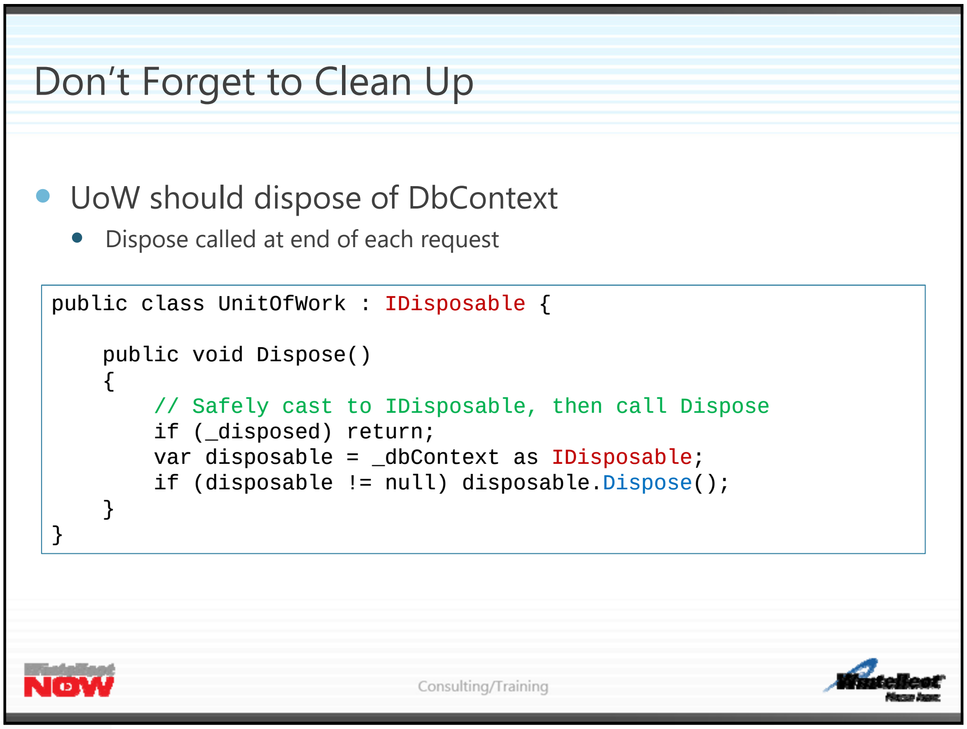
Consulting/Training



Unit of Work Implementation

```
public class UnitOfWork : IUnitOfWork, IDisposable
{
    public UnitOfWork(ICustomerRepository custRepo,
        IOrderRepository orderRepo,
        NorthwindContext dbContext) { // Code elided ...

    // Repository properties
    public ICustomerRepository CustomerRepository {
       get { return _customerRepository; } }

    public IOrderRepository OrderRepository {
        get { return _orderRepository; } }

    // Persistence
    public async Task<int> SaveChangesAsync() {
        return await _dbContext.SaveChangesAsync(); }
}
```

Consulting/Training

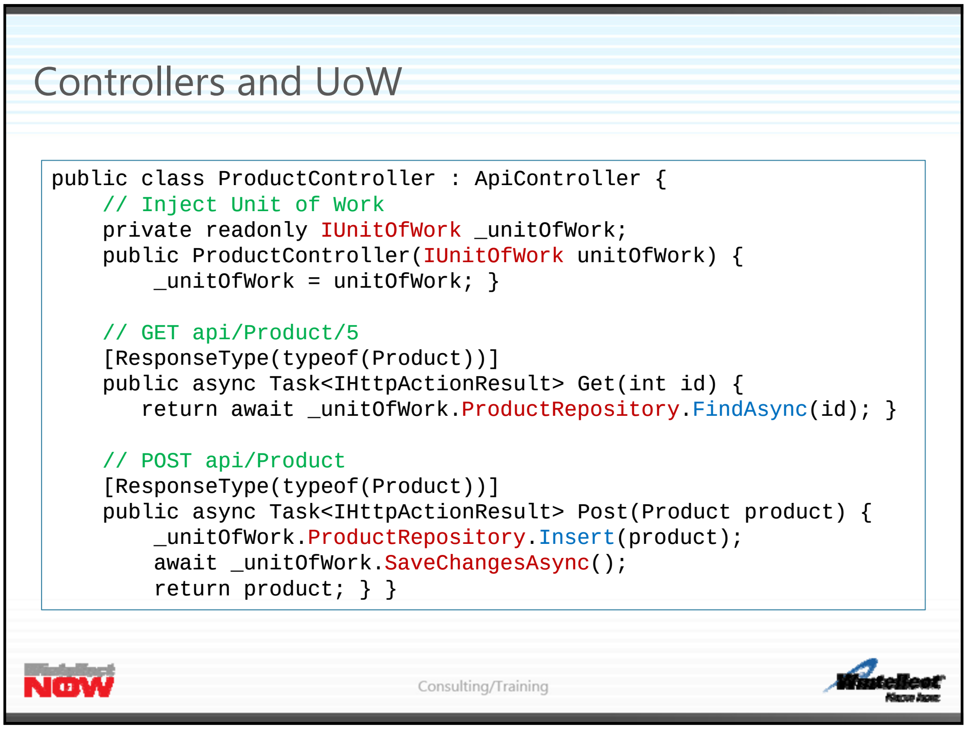


Don’t Forget to Clean Up

- UoW should dispose of DbContext
  - Dispose called at end of each request

```
public class UnitOfWork : IDisposable {

    public void Dispose()
    {
        // Safely cast to IDisposable, then call Dispose
        if (_disposed) return;
        var disposable = _dbContext as IDisposable;
        if (disposable != null) disposable.Dispose();
    }
}
```

Consulting/Training



Controllers and UoW

```
public class ProductController : ApiController {
    // Inject Unit of Work
    private readonly IUnitOfWork _unitOfWork;
    public ProductController(IUnitOfWork unitOfWork) {
        _unitOfWork = unitOfWork; }

    // GET api/Product/5
    [ResponseType(typeof(Product))]
    public async Task<IHttpActionResult> Get(int id) {
       return await _unitOfWork.ProductRepository.FindAsync(id); }

    // POST api/Product
    [ResponseType(typeof(Product))]
    public async Task<IHttpActionResult> Post(Product product) {
        _unitOfWork.ProductRepository.Insert(product);
        await _unitOfWork.SaveChangesAsync();
        return product; } }
```

Consulting/Training

QUESTIONS

Consulting/Training

Repository and Unit of Work Patterns

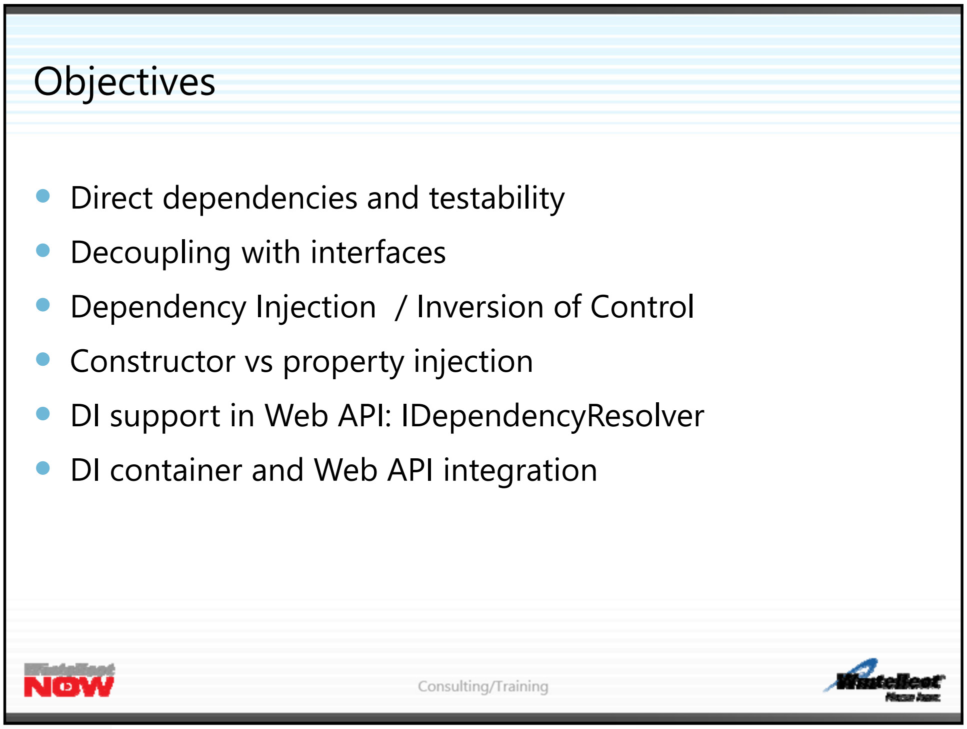
Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

Consulting/Training

Dependency Injection with ASP.NET Web API

Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

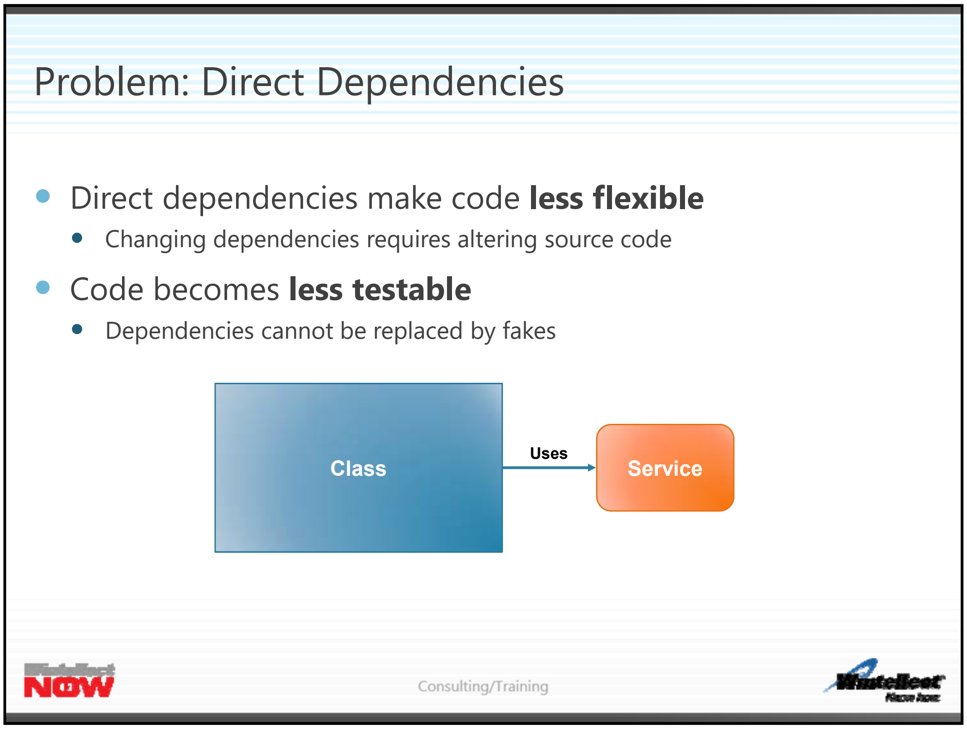
Consulting/Training



Objectives

- Direct dependencies and testability
- Decoupling with interfaces
- Dependency Injection / Inversion of Control
- Constructor vs property injection
- DI support in Web API: IDependencyResolver
- DI container and Web API integration

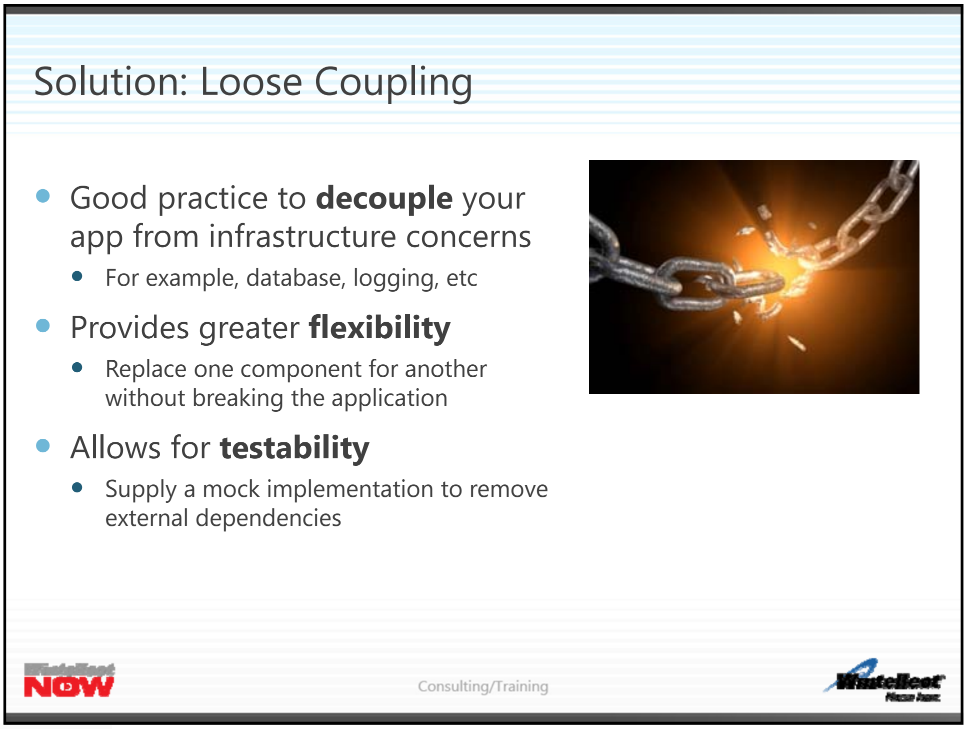
Consulting/Training



Problem: Direct Dependencies

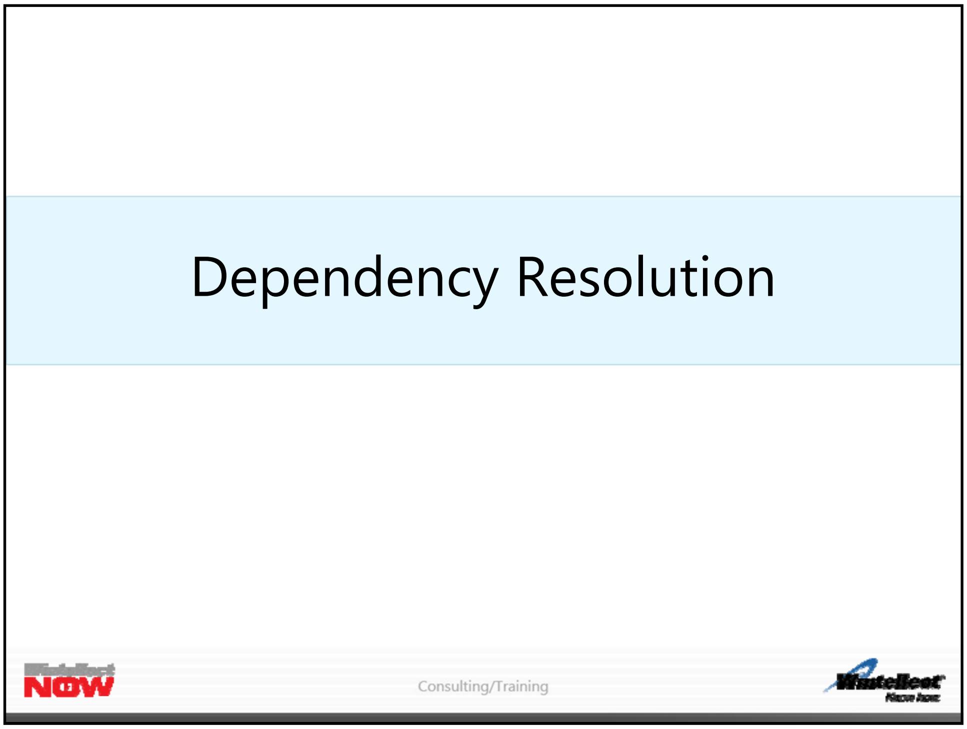
- Direct dependencies make code **less flexible**
  - Changing dependencies requires altering source code
- Code becomes **less testable**
  - Dependencies cannot be replaced by fakes

Class
Uses
Service



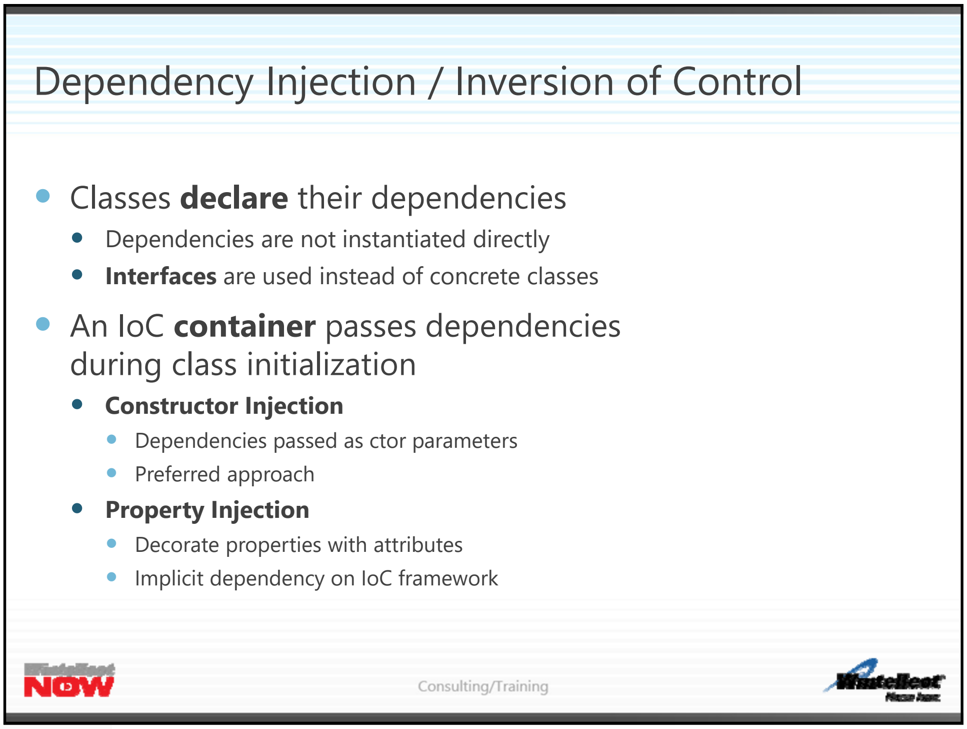
Solution: Loose Coupling

- Good practice to **decouple** your app from infrastructure concerns
  - For example, database, logging, etc
- Provides greater **flexibility**
  - Replace one component for another without breaking the application
- Allows for **testability**
  - Supply a mock implementation to remove external dependencies



Dependency Resolution

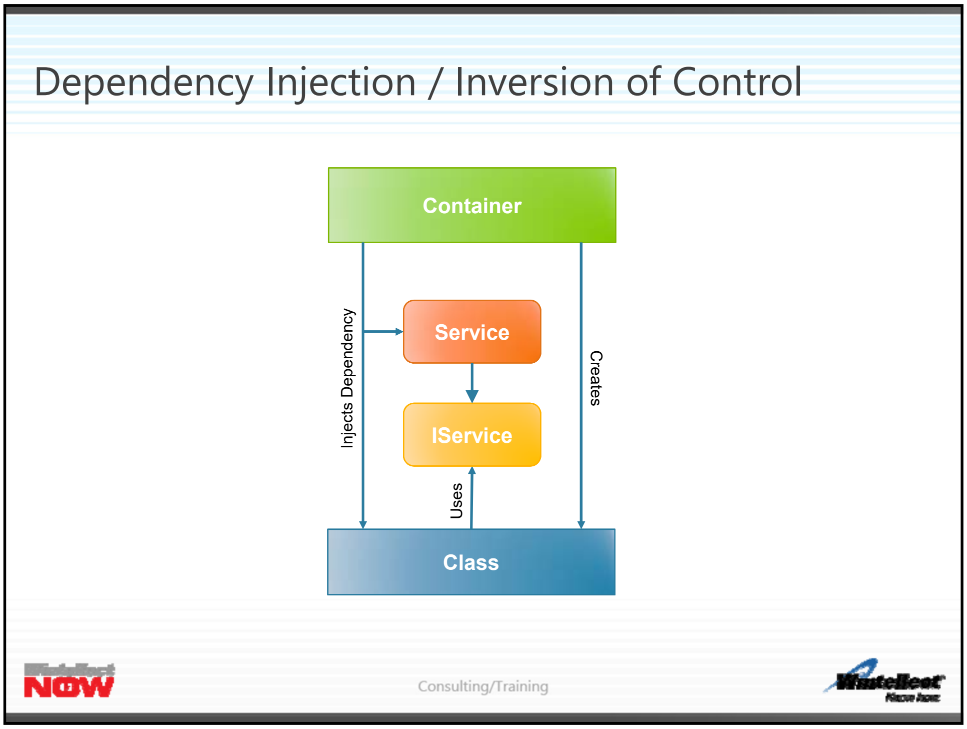
Consulting/Training



Dependency Injection / Inversion of Control

- Classes **declare** their dependencies
  - Dependencies are not instantiated directly
  - **Interfaces** are used instead of concrete classes
- An IoC **container** passes dependencies during class initialization
  - **Constructor Injection**
    - Dependencies passed as ctor parameters
    - Preferred approach
  - **Property Injection**
    - Decorate properties with attributes
    - Implicit dependency on IoC framework

Consulting/Training



Dependency Injection / Inversion of Control

Container

Injects Dependency

Service

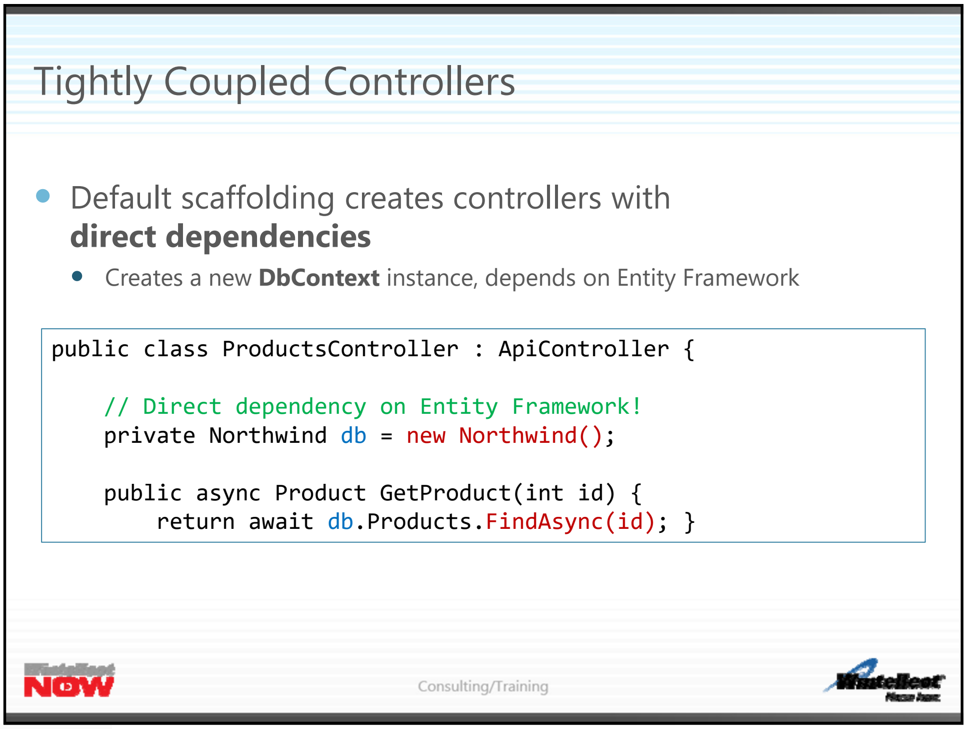
IService

Uses

Creates

Class

Consulting/Training

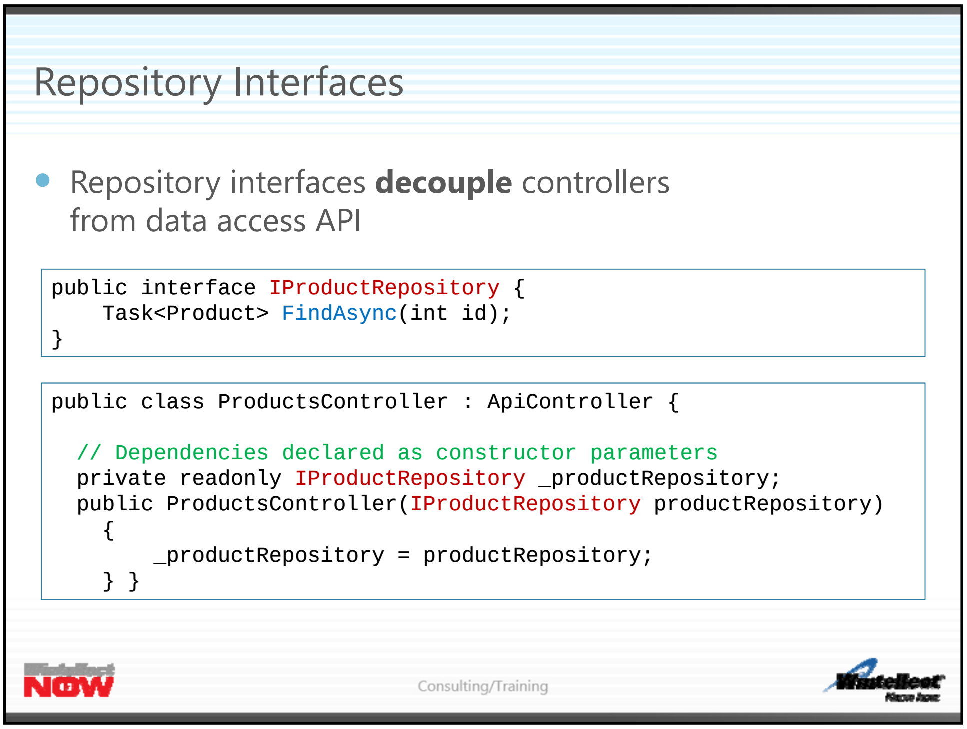


Tightly Coupled Controllers

- Default scaffolding creates controllers with **direct dependencies**
  - Creates a new **DbContext** instance, depends on Entity Framework

```
public class ProductsController : ApiController {

    // Direct dependency on Entity Framework!
    private Northwind db = new Northwind();

    public async Product GetProduct(int id) {
        return await db.Products.FindAsync(id); }
```

Consulting/Training



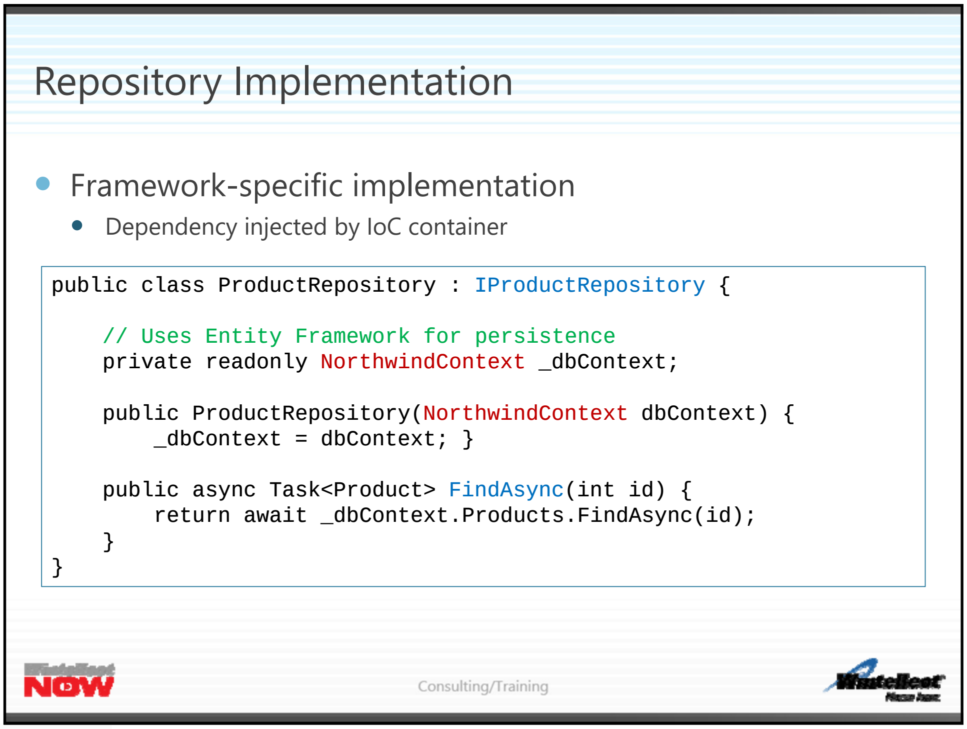
Repository Interfaces

- Repository interfaces **decouple** controllers from data access API

```
public interface IProductRepository {
    Task<Product> FindAsync(int id);
}
```

```
public class ProductsController : ApiController {

  // Dependencies declared as constructor parameters
  private readonly IProductRepository _productRepository;
  public ProductsController(IProductRepository productRepository)
    {
        _productRepository = productRepository;
    } }
```

Consulting/Training

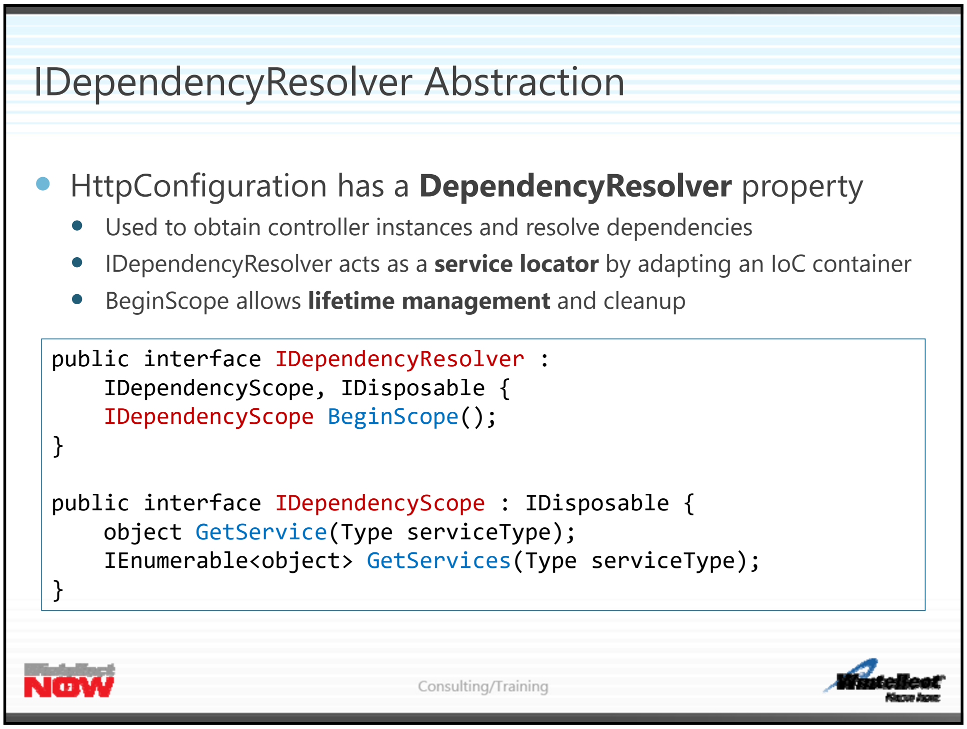


Repository Implementation

- Framework-specific implementation
  - Dependency injected by IoC container

```
public class ProductRepository : IProductRepository {

    // Uses Entity Framework for persistence
    private readonly NorthwindContext _dbContext;

    public ProductRepository(NorthwindContext dbContext) {
        _dbContext = dbContext; }

    public async Task<Product> FindAsync(int id) {
        return await _dbContext.Products.FindAsync(id);
    }
}
```

Consulting/Training

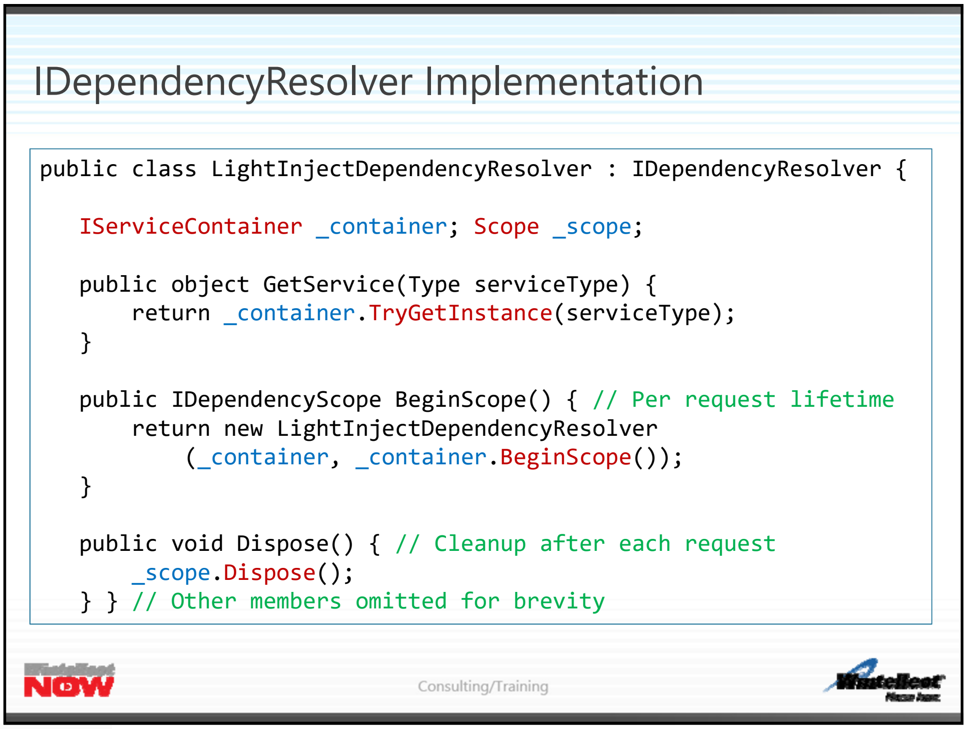


IDependencyResolver Abstraction

- HttpConfiguration has a **DependencyResolver** property
  - Used to obtain controller instances and resolve dependencies
  - IDependencyResolver acts as a **service locator** by adapting an IoC container
  - BeginScope allows **lifetime management** and cleanup

```
public interface IDependencyResolver :
    IDependencyScope, IDisposable {
    IDependencyScope BeginScope();
}

public interface IDependencyScope : IDisposable {
    object GetService(Type serviceType);
    IEnumerable<object> GetServices(Type serviceType);
}
```

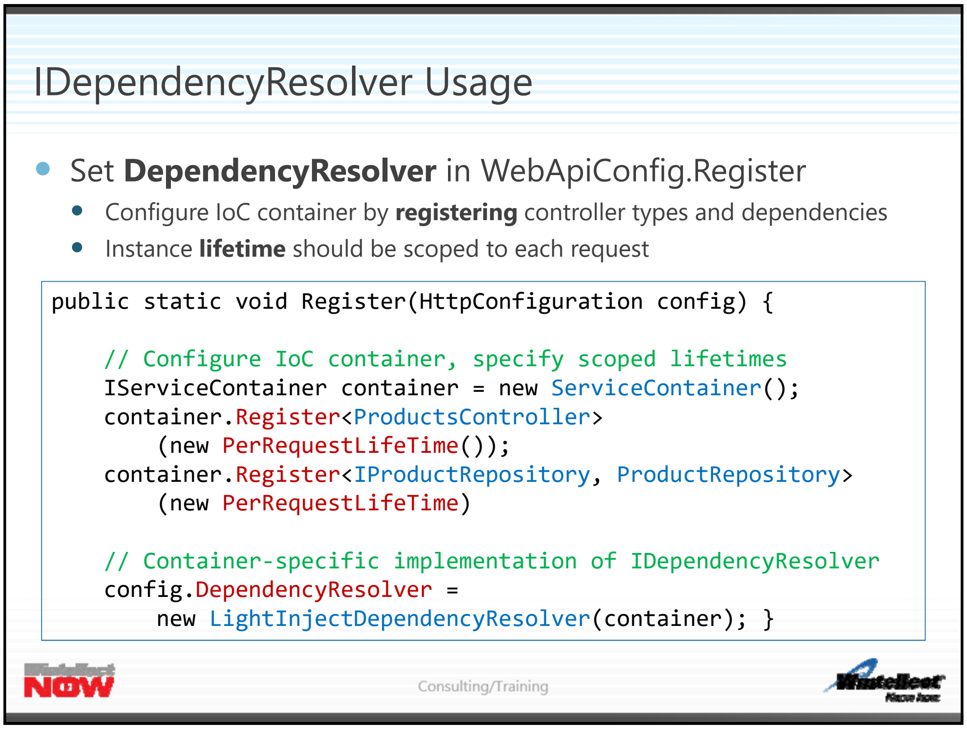
Consulting/Training



IDependencyResolver Implementation

```
public class LightInjectDependencyResolver : IDependencyResolver {

   IServiceContainer _container; Scope _scope;

   public object GetService(Type serviceType) {
       return _container.TryGetInstance(serviceType);
   }

   public IDependencyScope BeginScope() { // Per request lifetime
       return new LightInjectDependencyResolver
           (_container, _container.BeginScope());
   }

   public void Dispose() { // Cleanup after each request
       _scope.Dispose();
   } } // Other members omitted for brevity
```

Consulting/Training

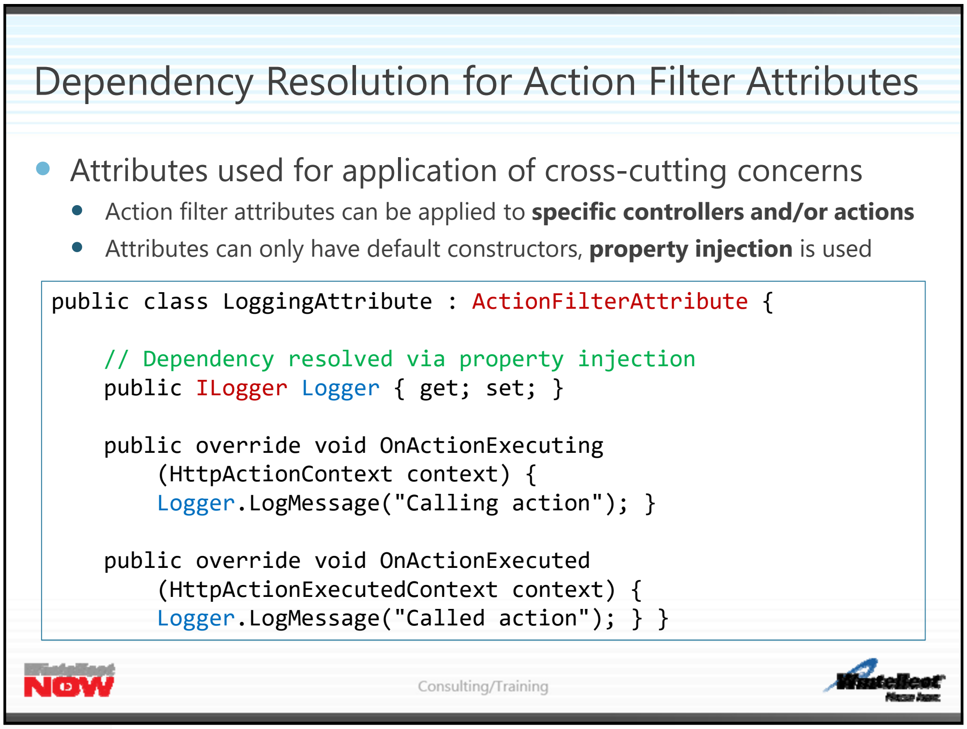


IDependencyResolver Usage

- Set **DependencyResolver** in WebApiConfig.Register
  - Configure IoC container by **registering** controller types and dependencies
  - Instance **lifetime** should be scoped to each request

```
public static void Register(HttpConfiguration config) {

    // Configure IoC container, specify scoped lifetimes
    IServiceContainer container = new ServiceContainer();
    container.Register<ProductsController>
        (new PerRequestLifeTime());
    container.Register<IProductRepository, ProductRepository>
        (new PerRequestLifeTime)

    // Container-specific implementation of IDependencyResolver
    config.DependencyResolver =
        new LightInjectDependencyResolver(container); }
```

Consulting/Training

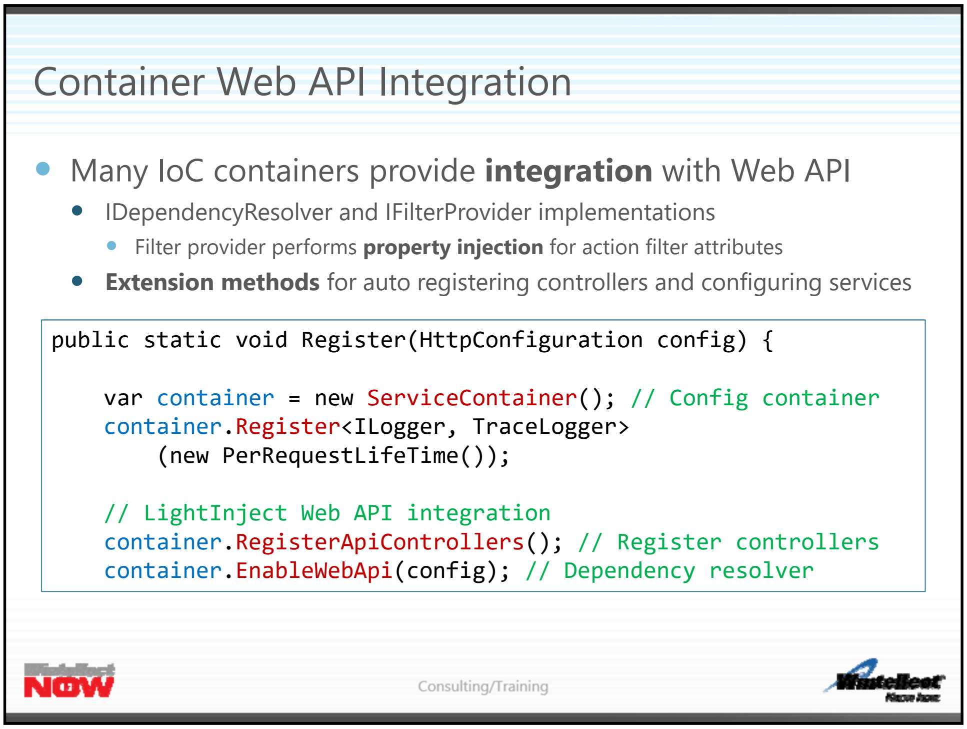


Dependency Resolution for Action Filter Attributes

- Attributes used for application of cross-cutting concerns
  - Action filter attributes can be applied to **specific controllers and/or actions**
  - Attributes can only have default constructors, **property injection** is used

```
public class LoggingAttribute : ActionFilterAttribute {

    // Dependency resolved via property injection
    public ILogger Logger { get; set; }

    public override void OnActionExecuting
        (HttpActionContext context) {
        Logger.LogMessage("Calling action"); }

    public override void OnActionExecuted
        (HttpActionExecutedContext context) {
        Logger.LogMessage("Called action"); } }
```

Consulting/Training



Container Web API Integration

- Many IoC containers provide **integration** with Web API
  - IDependencyResolver and IFilterProvider implementations
    - Filter provider performs **property injection** for action filter attributes
  - **Extension methods** for auto registering controllers and configuring services

```
public static void Register(HttpConfiguration config) {

    var container = new ServiceContainer(); // Config container
    container.Register<ILogger, TraceLogger>
        (new PerRequestLifeTime());

    // LightInject Web API integration
    container.RegisterApiControllers(); // Register controllers
    container.EnableWebApi(config); // Dependency resolver
```

Consulting/Training

QUESTIONS

Consulting/Training

Wintellect® Know how.

Dependency Injection with ASP.NET Web API

Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

Wintellect NOW

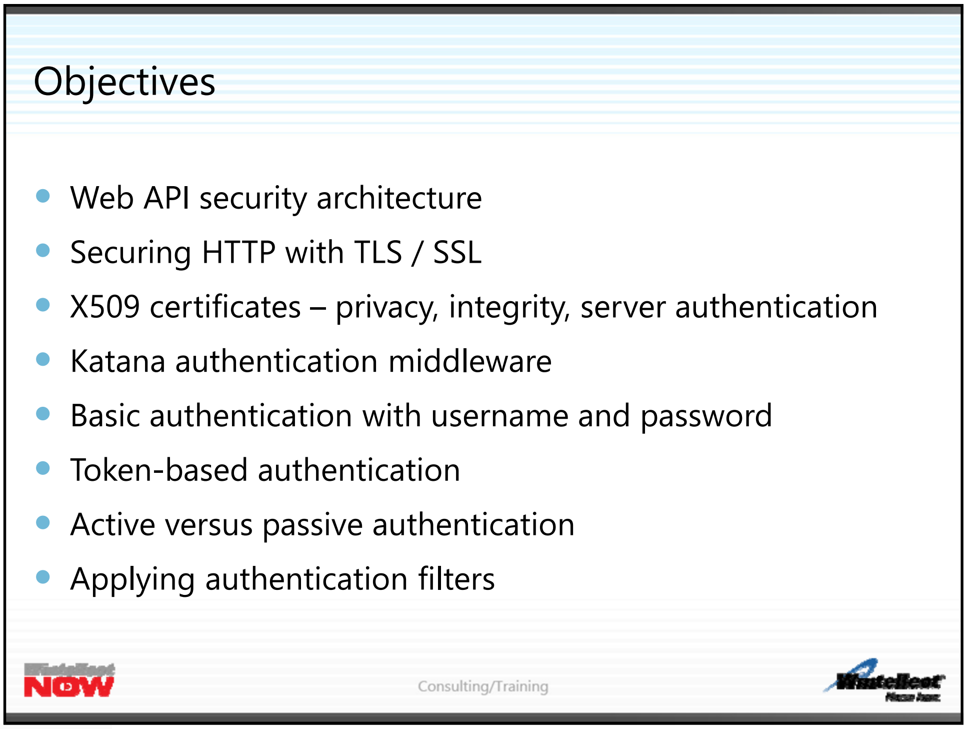
Consulting/Training

Wintellect Know how.

Web API Security: Transport, Authentication

Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

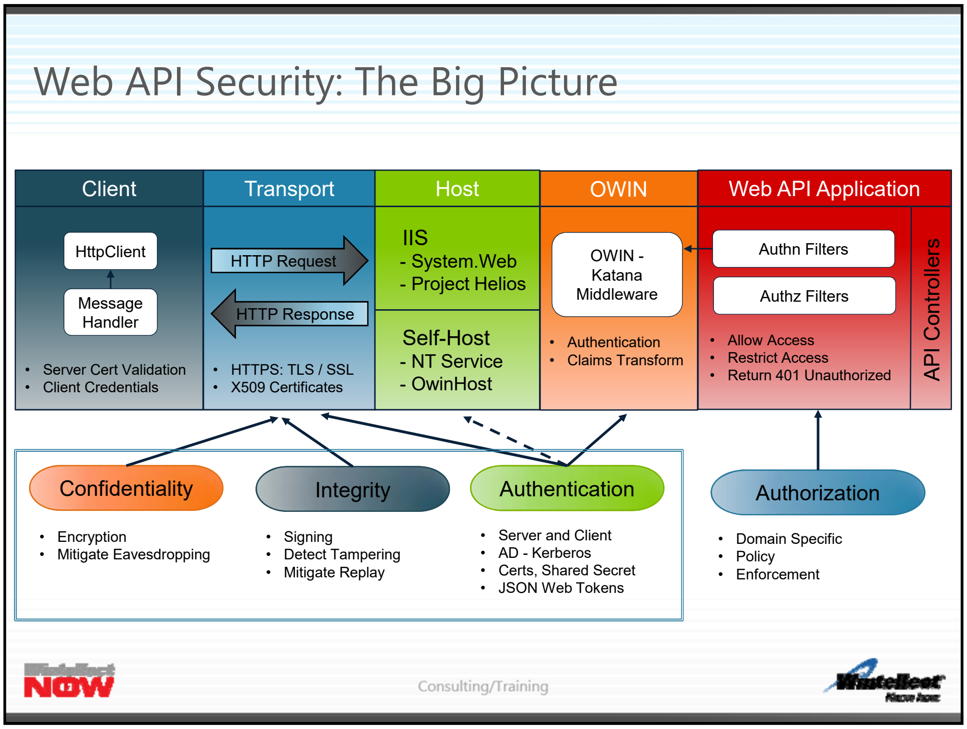
Consulting/Training



Objectives

- Web API security architecture
- Securing HTTP with TLS / SSL
- X509 certificates – privacy, integrity, server authentication
- Katana authentication middleware
- Basic authentication with username and password
- Token-based authentication
- Active versus passive authentication
- Applying authentication filters

Consulting/Training



Web API Security: The Big Picture

Client

HttpClient

Message Handler

- Server Cert Validation
- Client Credentials

Transport

HTTP Request

HTTP Response

- HTTPS: TLS / SSL
- X509 Certificates

Host

IIS
- System.Web
- Project Helios

Self-Host
- NT Service
- OwinHost

OWIN

OWIN - Katana Middleware

- Authentication
- Claims Transform

Web API Application

Authn Filters

Authz Filters

- Allow Access
- Restrict Access
- Return 401 Unauthorized

API Controllers

Confidentiality

- Encryption
- Mitigate Eavesdropping

Integrity

- Signing
- Detect Tampering
- Mitigate Replay

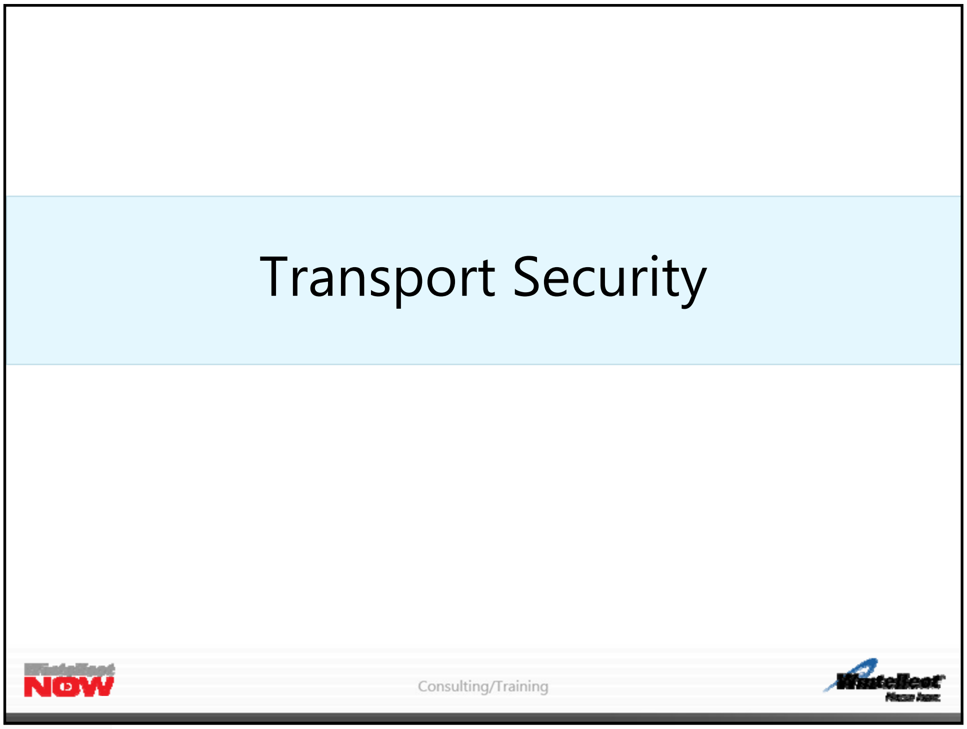
Authentication

- Server and Client
- AD - Kerberos
- Certs, Shared Secret
- JSON Web Tokens

Authorization

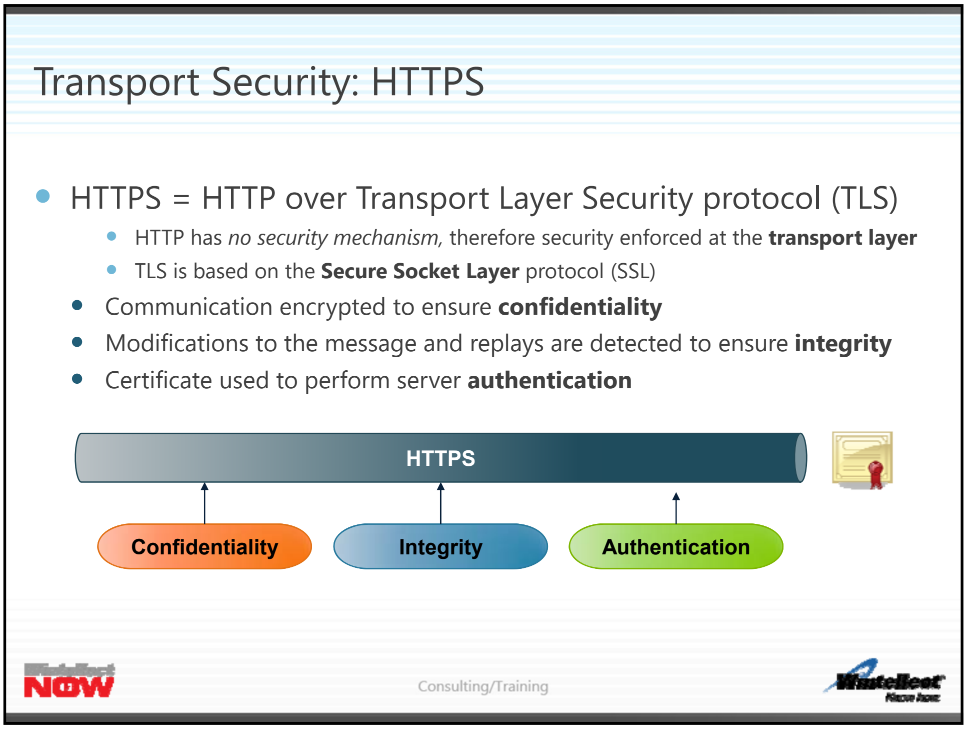
- Domain Specific
- Policy
- Enforcement

Consulting/Training



Transport Security

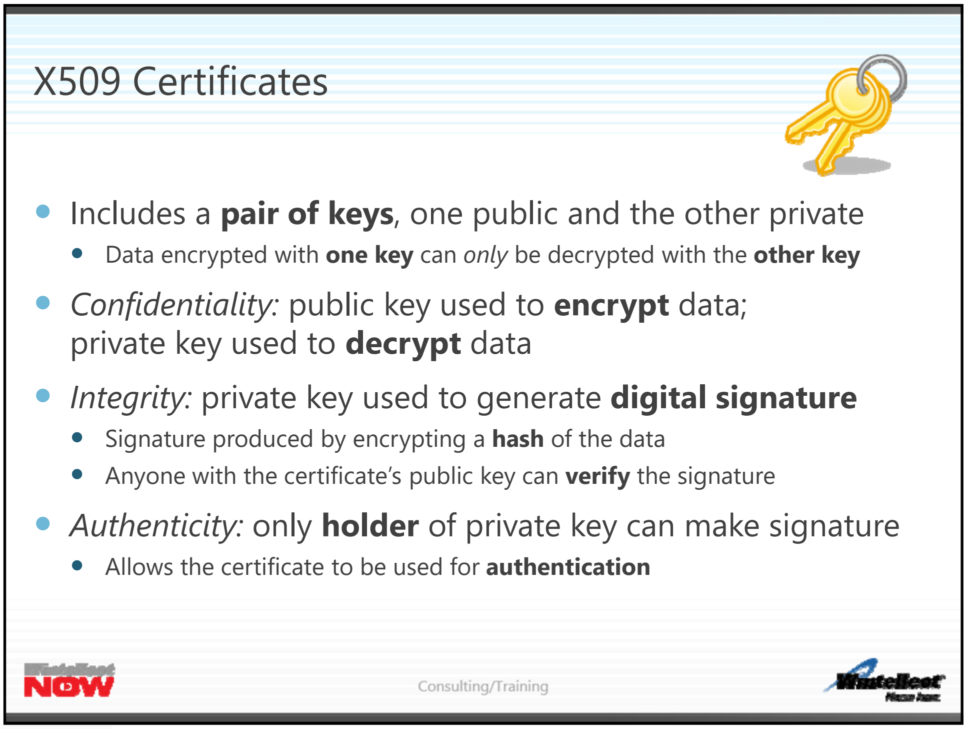
Consulting/Training



Transport Security: HTTPS

- HTTPS = HTTP over Transport Layer Security protocol (TLS)
  - HTTP has *no security mechanism,* therefore security enforced at the **transport layer**
  - TLS is based on the **Secure Socket Layer** protocol (SSL)
  - Communication encrypted to ensure **confidentiality**
  - Modifications to the message and replays are detected to ensure **integrity**
  - Certificate used to perform server **authentication**

Consulting/Training



X509 Certificates

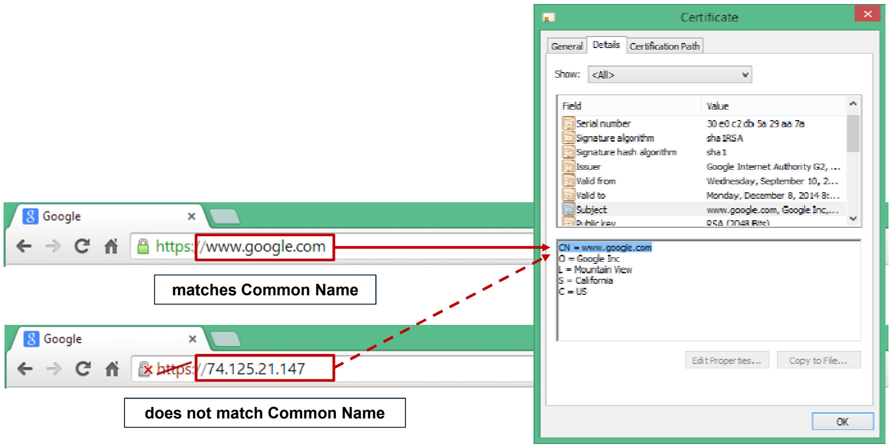
- Includes a **pair of keys**, one public and the other private
  - Data encrypted with **one key** can *only* be decrypted with the **other key**
- *Confidentiality:* public key used to **encrypt** data;
  private key used to **decrypt** data
- *Integrity*: private key used to generate **digital signature**
  - Signature produced by encrypting a **hash** of the data
  - Anyone with the certificate's public key can **verify** the signature
- *Authenticity:* only **holder** of private key can make signature
  - Allows the certificate to be used for **authentication**

Consulting/Training

Server Authentication

- HTTPS also helps to ensure requests are sent to the *correct* origin server
  - **Host name** portion of the URI *must match* certificate's **Common Name** (Subject: CN = www.google.com)

Server Authentication (cont.)



https://www.google.com

matches Common Name

https://74.125.21.147

does not match Common Name

Certificate

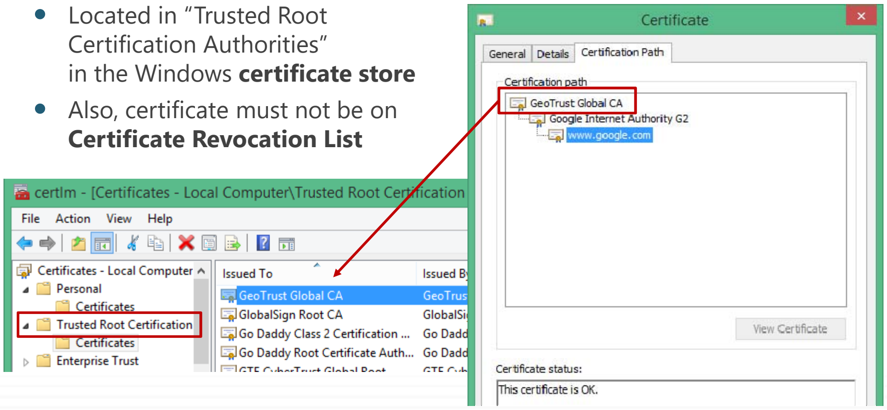
Show: <All>

Field	Value
Serial number	30 e0 c2 db 5a 29 aa 7a
Signature algorithm	sha1RSA
Signature hash algorithm	sha1
Issuer	Google Internet Authority G2, ...
Valid from	Wednesday, September 10, 2...
Valid to	Monday, December 8, 2014 8:...
Subject	www.google.com, Google Inc,...
Public key	RSA (2048 Bits)

CN = www.google.com
O = Google Inc
L = Mountain View
S = California
C = US

Establishing Trust – Certificates Store

- Client must **trust** certificate's issuer
  - Located in "Trusted Root Certification Authorities" in the Windows **certificate store**
  - Also, certificate must not be on **Certificate Revocation List**



certlm - [Certificates - Local Computer\Trusted Root Certification

Issued To	Issued By
GeoTrust Global CA	GeoTrus
GlobalSign Root CA	GlobalSi
Go Daddy Class 2 Certification ...	Go Dadd
Go Daddy Root Certificate Auth...	Go Dadd
GTE CyberTrust Global Root	GTE Cyb

Certificate

Certification path

GeoTrust Global CA
Google Internet Authority G2
www.google.com

Certificate status:

This certificate is OK.

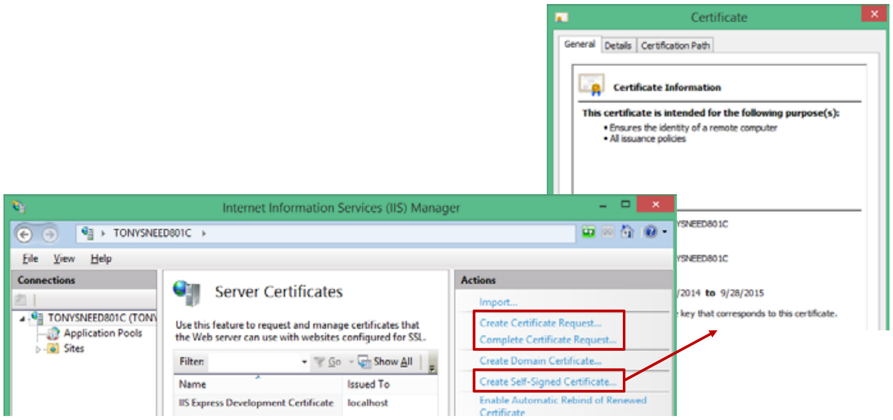
Consulting/Training

Create or Request Certificates with IIS Manager

- Request certificates from internal or external **CA**
  - Windows Server comes with a built-in **certificate authority**
- Create **self-signed** certificates for development and testing
  - Copy to **Trusted Root Certification Authorities** in Windows certificate store

Consulting/Training

Create or Request Certificates with IIS Manager



This certificate is intended for the following purpose(s):

- Ensures the identity of a remote computer
- All issuance policies

/2014 to 9/28/2015

key that corresponds to this certificate.

Use this feature to request and manage certificates that the Web server can use with websites configured for SSL.

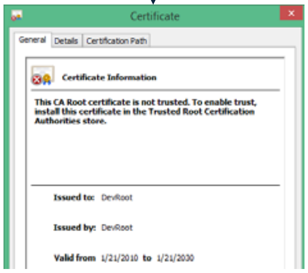
Name	Issued To
IIS Express Development Certificate	localhost

Development Certificates – Root Certificate

Makecert.exe: Root Certificate	
-r	Self-signed
-n "CN=DevRoot"	Name
-pe	Exportable
-sv DevRoot.pvk	Name of private key file
-a sha1	Hashing Algorithm
-len 2048	Key Length
-b 01/21/2014	Valid from
-e 01/21/2034	Valid to
-cy authority	Certificate type
DevRoot.cer	Name of certificate file

Development Certificates – Root Certificate

```
makecert.exe -r -n "CN=DevRoot" -pe -sv
DevRoot.pvk -a sha1 -len 2048 -b 01/21/2014 -e
01/21/2034 -cy authority DevRoot.cer
```



Certificate Information

This CA Root certificate is not trusted. To enable trust, install this certificate in the Trusted Root Certification Authorities store.

Issued to: DevRoot

Issued by: DevRoot

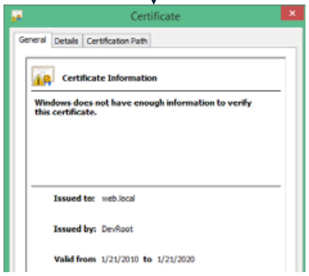
Valid from 1/21/2010 to 1/21/2030

Development Certificates – Server Certificate

Makecert.exe: Server Certificate	
-iv DevRoot.pvk	File name of root private key
-ic DevRoot.cer	File name of root certificate
-n "CN=web.local"	Name
-pe	Mark private key as exportable
-sv web.local.pvk	Name of private key file
-a sha1	Hashing algorithm
-len 2048	Ley length
-b 01/21/2014	Valid from
-e 01/21/2024	Valid to
-sky exchange	Certificate type
web.local.cer	name of certificate file
-eku 1.3.6.1.5.5.7.3.1	Extended key usage

Development Certificates – Server Certificate

```
makecert.exe -iv DevRoot.pvk -ic DevRoot.cer -
n "CN=web.local" -pe -sv web.local.pvk -a sha1
  -len 2048 -b 01/21/2010 -e 01/21/2020 -sky
 exchange web.local.cer -eku 1.3.6.1.5.5.7.3.1
```



Certificate Information

Windows does not have enough information to verify this certificate.

Issued to: web.local

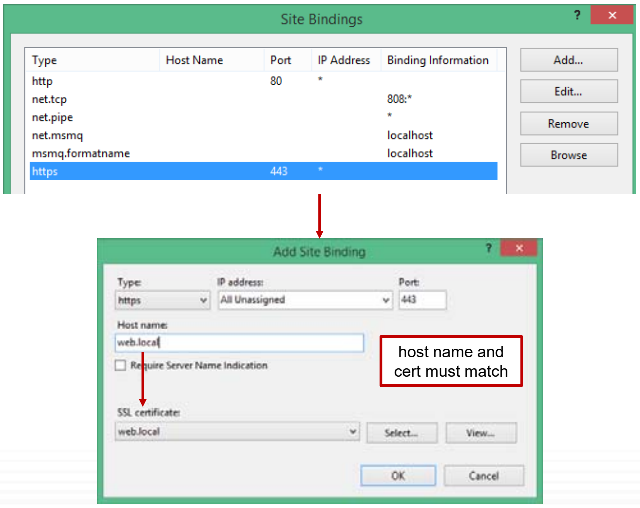
Issued by: DevRoot

Valid from 1/21/2010 to 1/21/2020

SSL Certificate Binding in IIS

- Add a **site binding** to Default Web Site in IIS Manager
  - Select **https** as scheme type
  - Select an **SSL certificate** which has a trusted root
  - Enter a **host name** that *matches* the certificate's "Issue To" field
    - Same as subject's common name (CN)

SSL Certificate Binding in IIS (cont.)



Site Bindings

Type	Host Name	Port	IP Address	Binding Information
http		80	*	
net.tcp				808:*
net.pipe				*
net.msmq				localhost
msmq.formatname				localhost
https		443	*	

Add Site Binding

Type: https

IP address: All Unassigned

Port: 443

Host name: web.local

☐ Require Server Name Indication

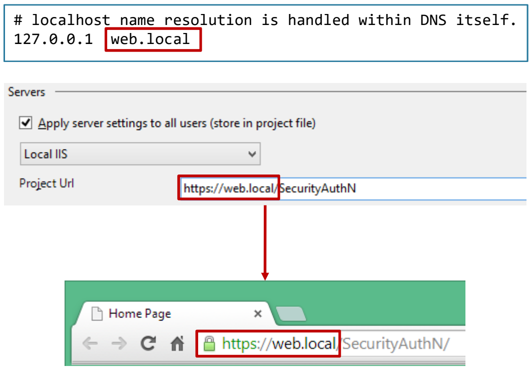
SSL certificate: web.local

host name and cert must match

Mapping IP Address to Host Name

- Edit **hosts** file to map IP address (127.0.0.1) to the host name (web.local)
  - Located: **%windir%\System32\drivers\etc**
- Set server on **Web** tab of Visual Studio project properties page to use **HTTPS**
  - Select **Local IIS**, specify *project url* that includes **https** and matching **host name** (web.local)

Mapping IP Address to Host Name (cont.)



```
# localhost name resolution is handled within DNS itself.
127.0.0.1  web.local
```

Servers

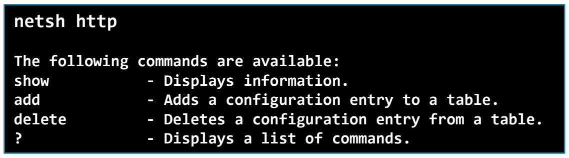
☑ Apply server settings to all users (store in project file)

Local IIS

Project Url https://web.local/SecurityAuthN

NetSh Command-Line Tool

- Use **netsh** commands to query and configure **HTTP.sys** settings
  - Run command shell "**as administrator**" to configure SSL for *self-hosted* apps
  - Execute **urlacl** commands to show, add and delete **url reservations** for domain accounts
  - Execute **sslcert** command to show, add and delete **ssl certificate bindings**



```
netsh http

The following commands are available:
show           - Displays information.
add            - Adds a configuration entry to a table.
delete         - Deletes a configuration entry from a table.
?              - Displays a list of commands.
```

NetSh Command-Line Tool Examples

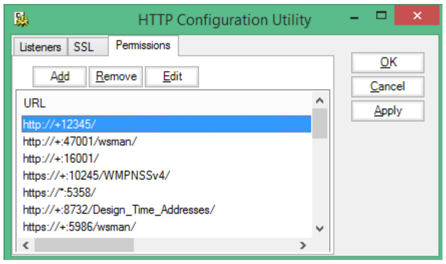
```
netsh http add urlacl

Example:
add urlacl url=http://+:80/MyUri user=DOMAIN\user
```

```
netsh http add sslcert

Example:
add sslcert ipport=1.1.1.1:443
certhash=0102030405060708090A0B0C0D0E0F10111 appid={00112233-
4455-6677-8899-AABBCCDDEEFF}
```

Consulting/Training

GUI for NetSh: HttpConfig Utility

- **HttpConfig** provides a graphical interface for netsh HTTP commands
  - Download it from http://www.stevestechspot.com



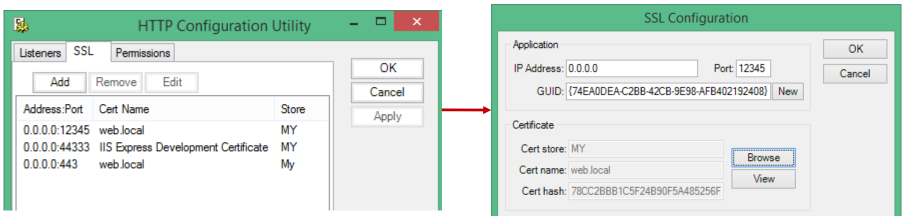
URL
http://+12345/
http://+:47001/wsman/
http://+:16001/
https://+:10245/WMPNSSv4/
https://*:5358/
http://+:8732/Design_Time_Addresses/
https://+:5986/wsman/

Consulting/Training

Self-hosted: SSL Certificate Binding

- To configure SSL for **self-hosted** apps, add a **cert binding**
  - Specify **port** (ipport), **cert thumbprint** (certhash), and **arbitrary guid** (appid)

```
netsh http add sslcert ipport=0.0.0.0:port certhash=thumbprint appid={app-guid}
```



Address:Port	Cert Name	Store
0.0.0.0:12345	web.local	MY
0.0.0.0:44333	IIS Express Development Certificate	MY
0.0.0.0:443	web.local	My

Application

IP Address: 0.0.0.0 Port: 12345

GUID: {74EA0DEA-C2BB-42CB-9E98-AFB402192408}

Certificate

Cert store: MY

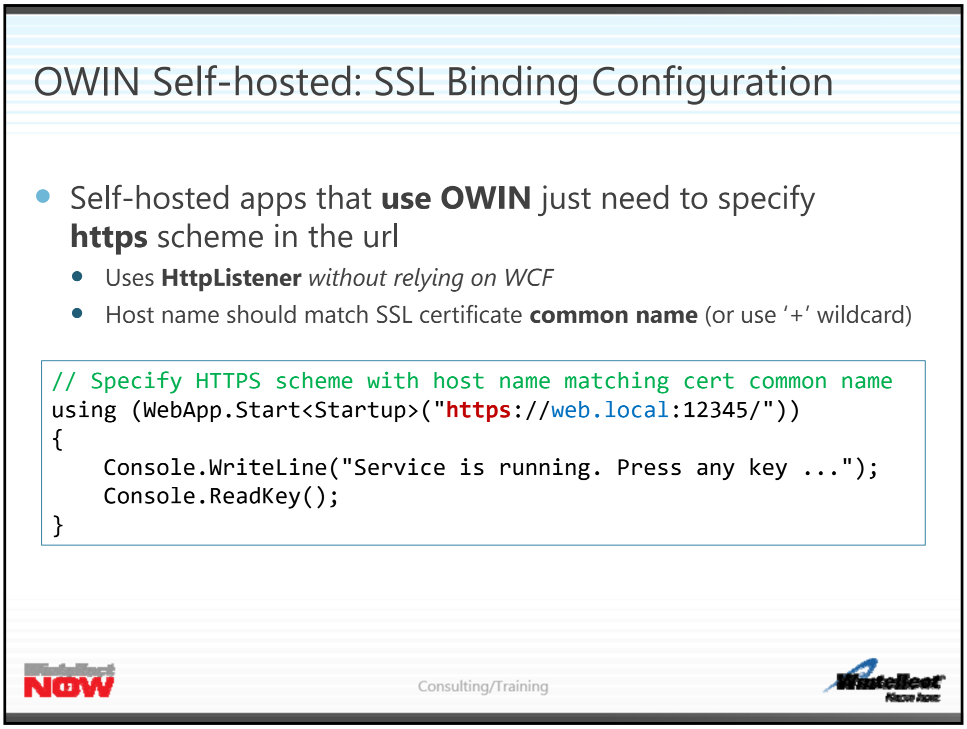
Cert name: web.local

Cert hash: 78CC2BBB1C5F24B90F5A485256F

Web API Self-hosted: SSL Binding Configuration

- Self-hosted apps that do **not** use OWIN rely on **WCF**
  - Extend **HttpSelfHostConfiguration** and override **OnConfigureBinding**
  - Specify **transport security** for HttpBinding

```
// Must configure HttpBinding to use SSL with WCF
public class SecureHttpSelfHostConfiguration :
    HttpSelfHostConfiguration {

    protected override BindingParameterCollection
        OnConfigureBinding(HttpBinding httpBinding) {
        httpBinding.Security.Mode =
            HttpBindingSecurityMode.Transport;
        return base.OnConfigureBinding(httpBinding);
    }
}
```

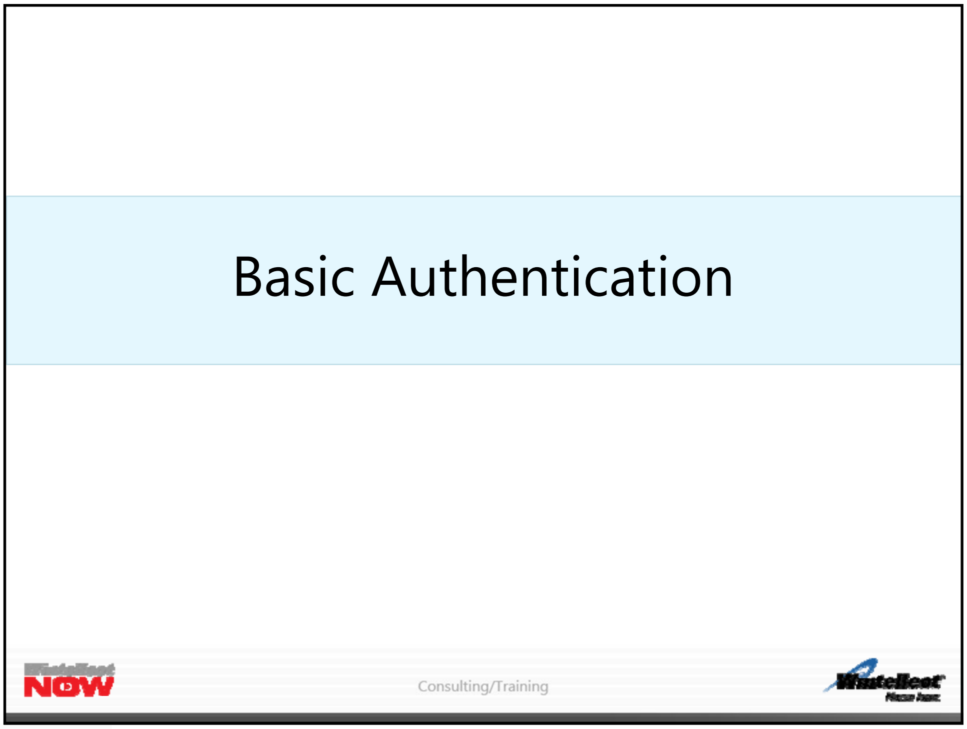


OWIN Self-hosted: SSL Binding Configuration

- Self-hosted apps that **use OWIN** just need to specify **https** scheme in the url
  - Uses **HttpListener** *without relying on WCF*
  - Host name should match SSL certificate **common name** (or use '+' wildcard)

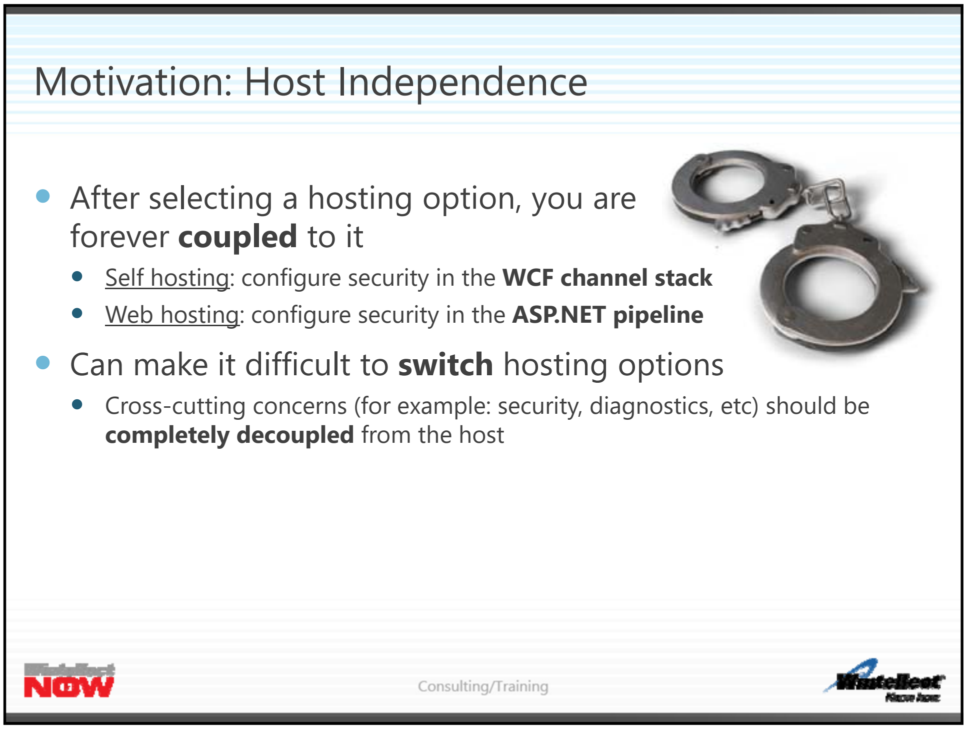
```
// Specify HTTPS scheme with host name matching cert common name
using (WebApp.Start<Startup>("https://web.local:12345/"))
{
    Console.WriteLine("Service is running. Press any key ...");
    Console.ReadKey();
}
```

Consulting/Training



Basic Authentication

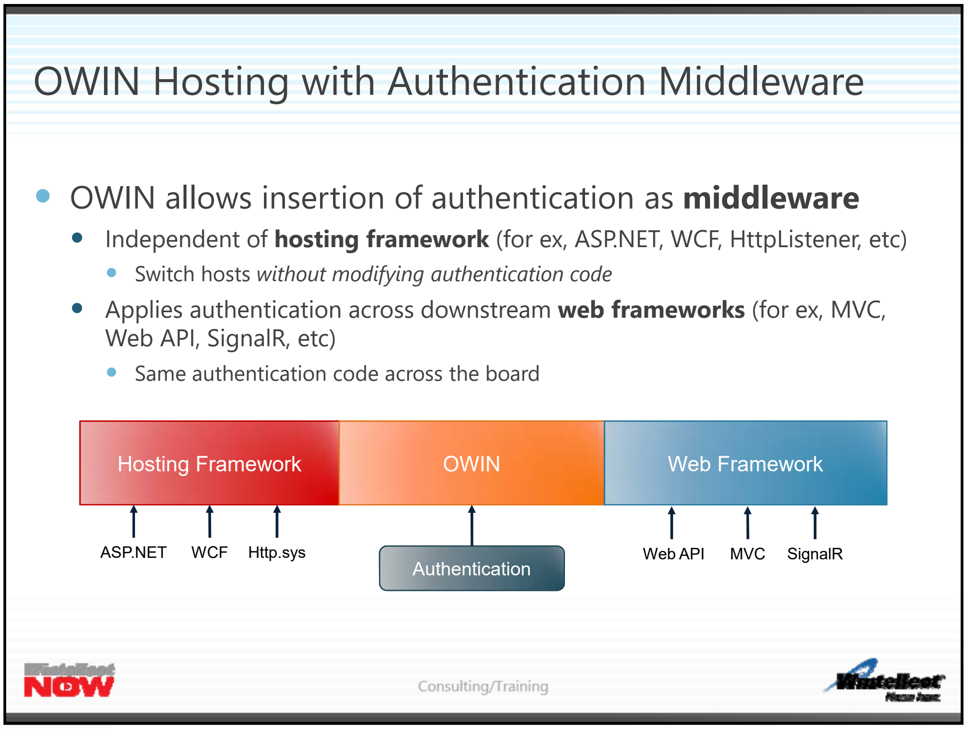
Consulting/Training



Motivation: Host Independence

- After selecting a hosting option, you are forever **coupled** to it
  - Self hosting: configure security in the **WCF channel stack**
  - Web hosting: configure security in the **ASP.NET pipeline**
- Can make it difficult to **switch** hosting options
  - Cross-cutting concerns (for example: security, diagnostics, etc) should be **completely decoupled** from the host

Consulting/Training



OWIN Hosting with Authentication Middleware

- OWIN allows insertion of authentication as **middleware**
  - Independent of **hosting framework** (for ex, ASP.NET, WCF, HttpListener, etc)
    - Switch hosts *without modifying authentication code*
  - Applies authentication across downstream **web frameworks** (for ex, MVC, Web API, SignalR, etc)
    - Same authentication code across the board

Hosting Framework

OWIN

Web Framework

ASP.NET

WCF

Http.sys

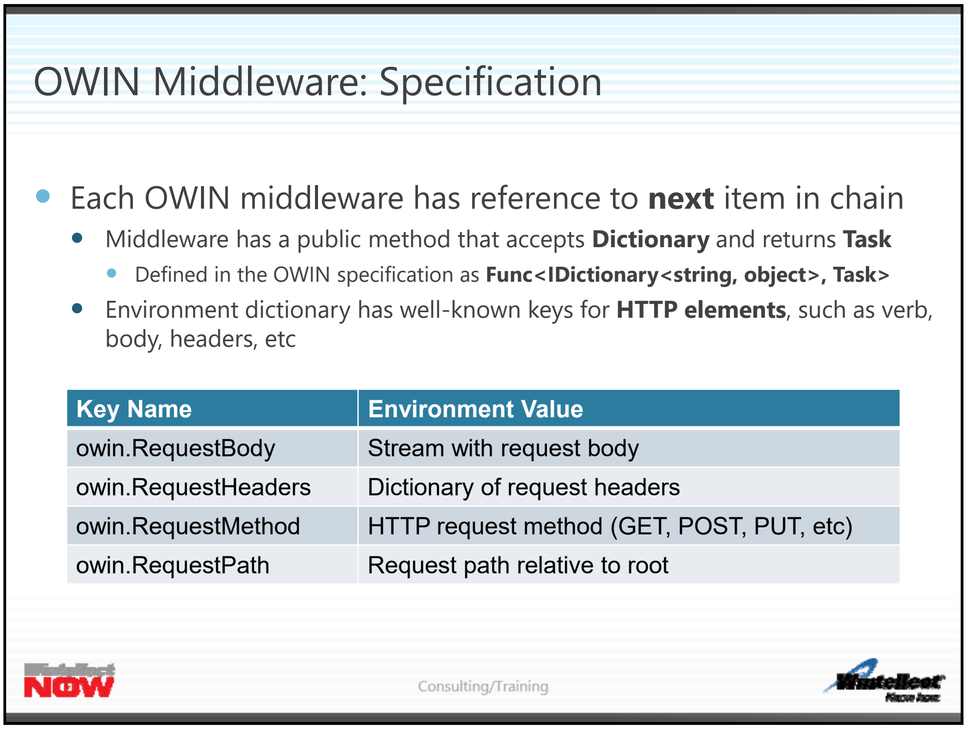
Authentication

Web API

MVC

SignalR

Consulting/Training

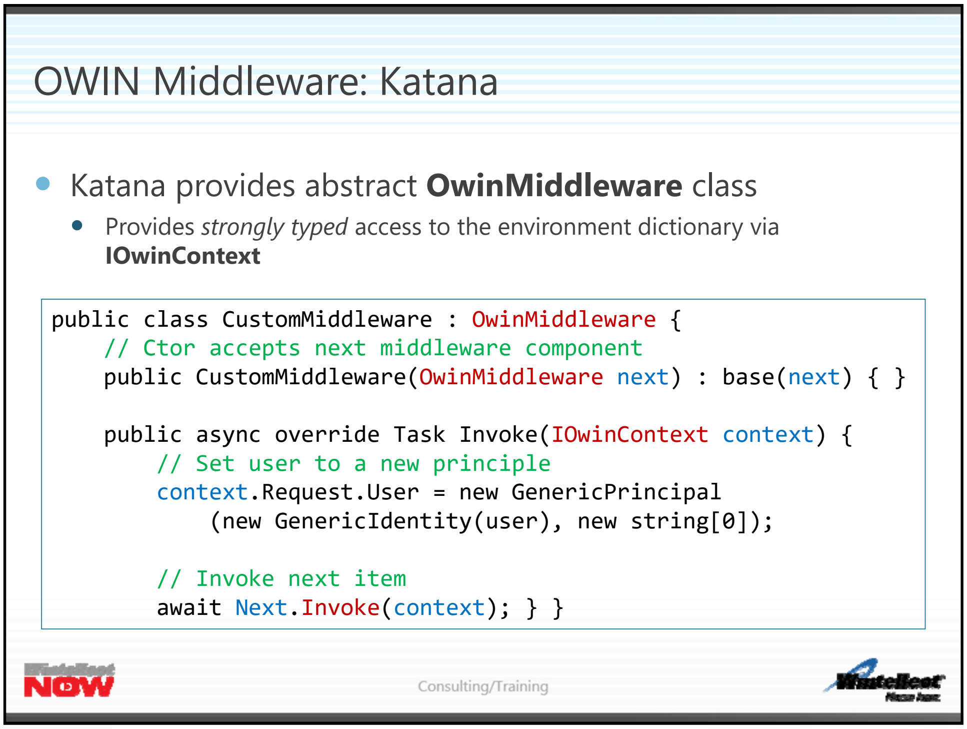


OWIN Middleware: Specification

- Each OWIN middleware has reference to **next** item in chain
  - Middleware has a public method that accepts **Dictionary** and returns **Task**
    - Defined in the OWIN specification as **Func<IDictionary<string, object>, Task>**
  - Environment dictionary has well-known keys for **HTTP elements**, such as verb, body, headers, etc

Key Name	Environment Value
owin.RequestBody	Stream with request body
owin.RequestHeaders	Dictionary of request headers
owin.RequestMethod	HTTP request method (GET, POST, PUT, etc)
owin.RequestPath	Request path relative to root

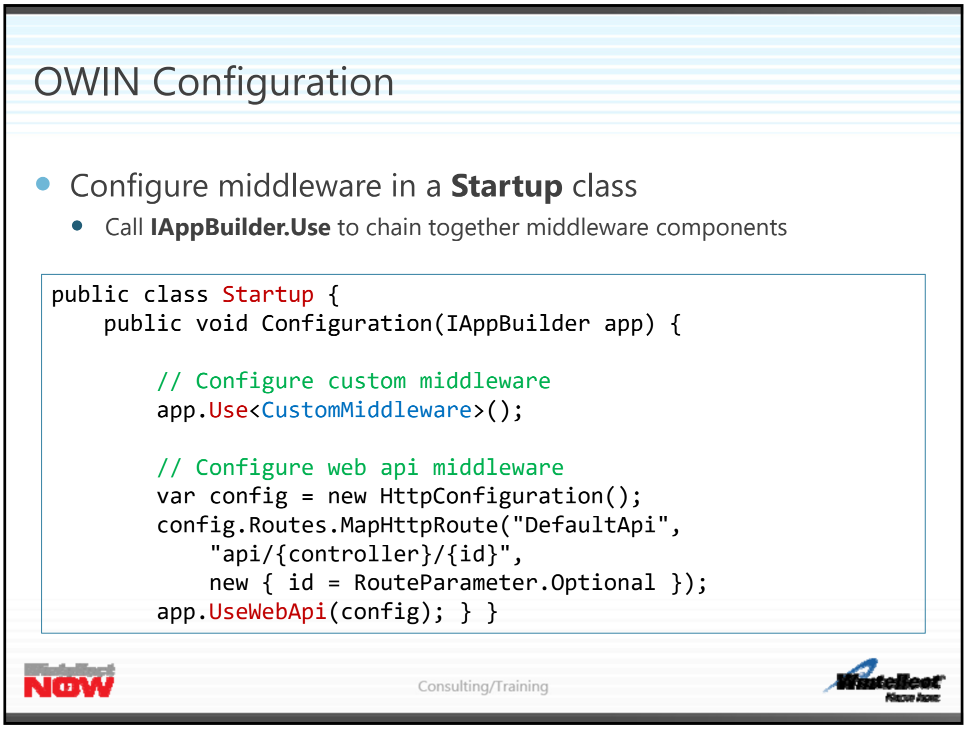
Consulting/Training



OWIN Middleware: Katana

- Katana provides abstract **OwinMiddleware** class
  - Provides *strongly typed* access to the environment dictionary via **IOwinContext**

```
public class CustomMiddleware : OwinMiddleware {
    // Ctor accepts next middleware component
    public CustomMiddleware(OwinMiddleware next) : base(next) { }

    public async override Task Invoke(IOwinContext context) {
        // Set user to a new principle
        context.Request.User = new GenericPrincipal
            (new GenericIdentity(user), new string[0]);

        // Invoke next item
        await Next.Invoke(context); } }
```

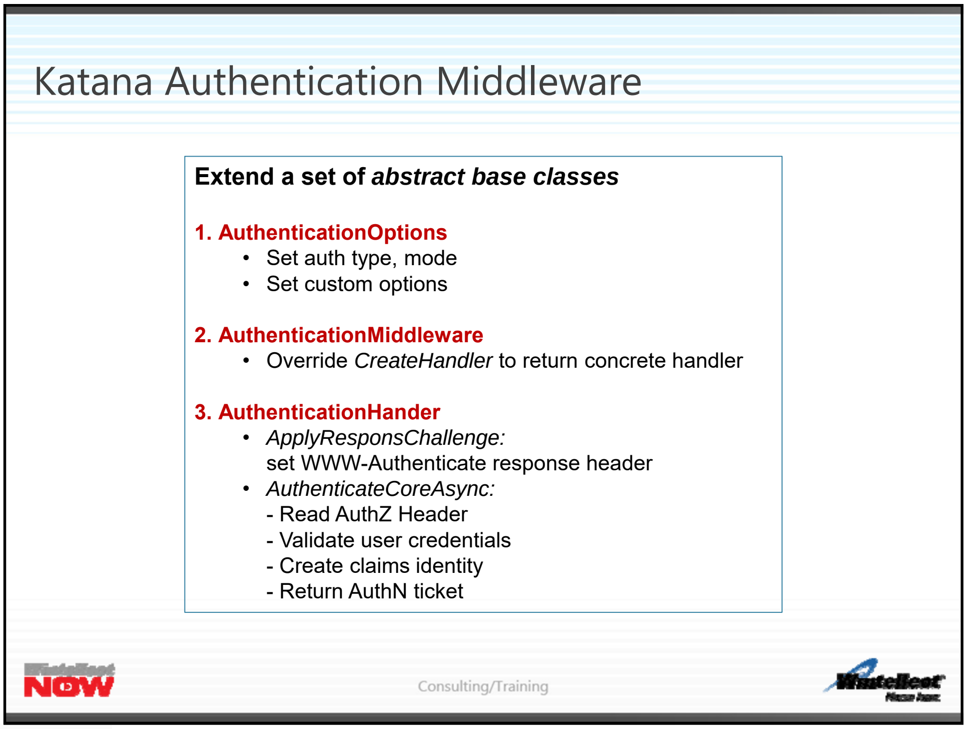


OWIN Configuration

- Configure middleware in a **Startup** class
  - Call **IAppBuilder.Use** to chain together middleware components

```
public class Startup {
    public void Configuration(IAppBuilder app) {

        // Configure custom middleware
        app.Use<CustomMiddleware>();

        // Configure web api middleware
        var config = new HttpConfiguration();
        config.Routes.MapHttpRoute("DefaultApi",
            "api/{controller}/{id}",
            new { id = RouteParameter.Optional });
        app.UseWebApi(config); } }
```

Consulting/Training



Katana Authentication Middleware

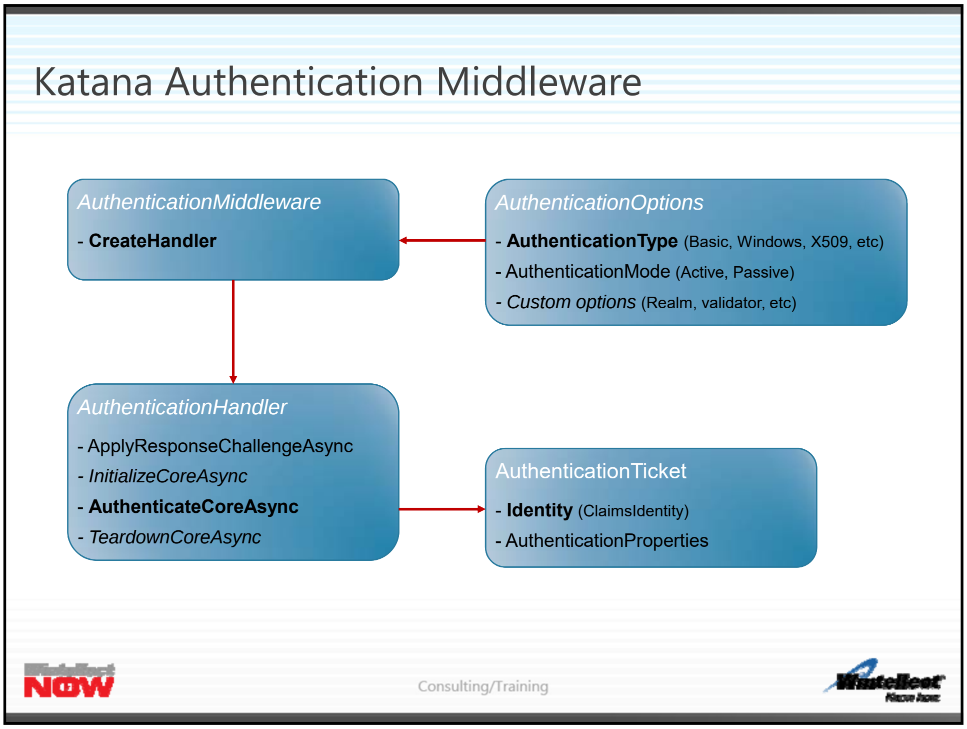
Extend a set of *abstract base classes*

1. AuthenticationOptions
- Set auth type, mode
- Set custom options

2. AuthenticationMiddleware
- Override *CreateHandler* to return concrete handler

3. AuthenticationHander
- *ApplyResponsChallenge:*
  set WWW-Authenticate response header
- *AuthenticateCoreAsync:*
  - Read AuthZ Header
  - Validate user credentials
  - Create claims identity
  - Return AuthN ticket

Consulting/Training



Katana Authentication Middleware

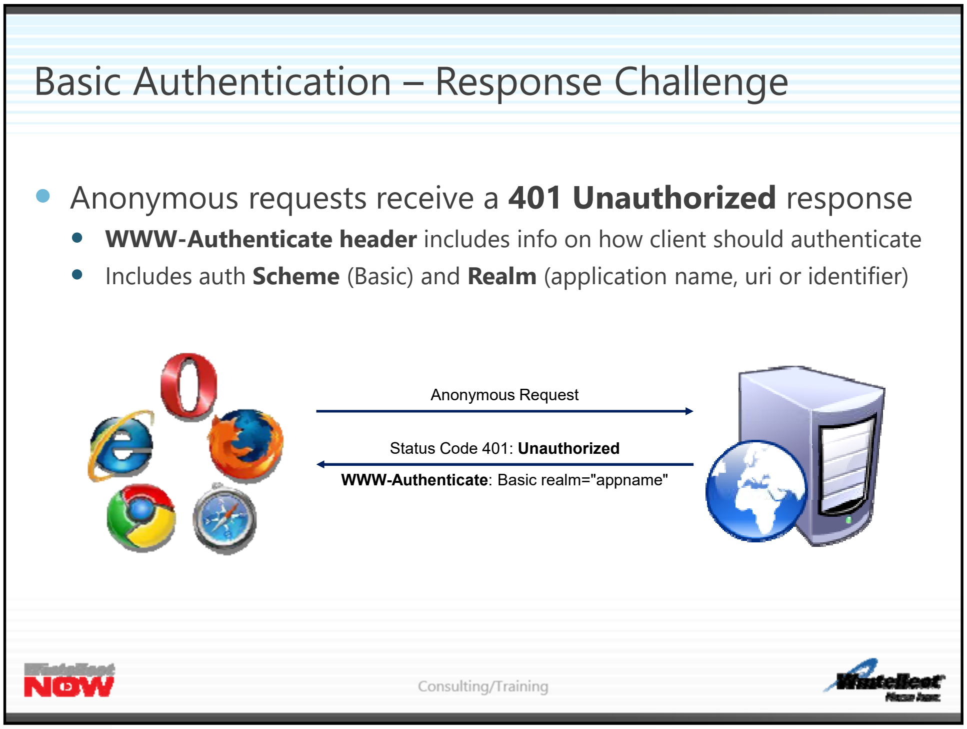
AuthenticationMiddleware
- **CreateHandler**

AuthenticationOptions
- **AuthenticationType** (Basic, Windows, X509, etc)
- AuthenticationMode (Active, Passive)
- *Custom options* (Realm, validator, etc)

AuthenticationHandler
- ApplyResponseChallengeAsync
- *InitializeCoreAsync*
- **AuthenticateCoreAsync**
- *TeardownCoreAsync*

AuthenticationTicket
- **Identity** (ClaimsIdentity)
- AuthenticationProperties

Consulting/Training

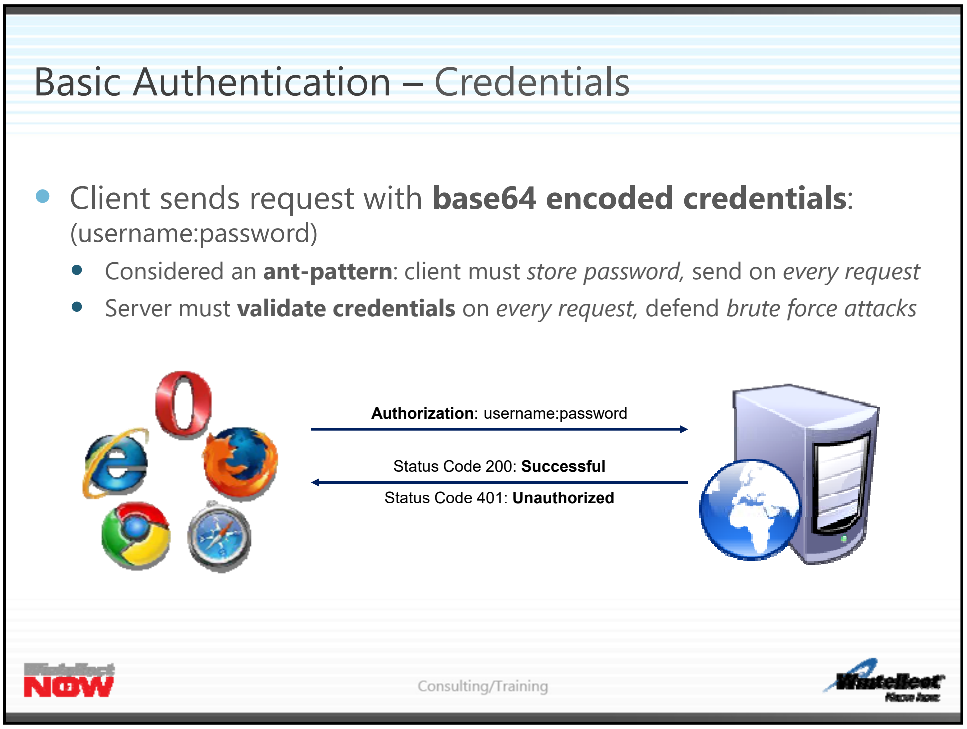


Basic Authentication – Response Challenge

- Anonymous requests receive a **401 Unauthorized** response
  - **WWW-Authenticate header** includes info on how client should authenticate
  - Includes auth **Scheme** (Basic) and **Realm** (application name, uri or identifier)

Anonymous Request
Status Code 401: **Unauthorized**
WWW-Authenticate: Basic realm="appname"

Consulting/Training



Basic Authentication – Credentials

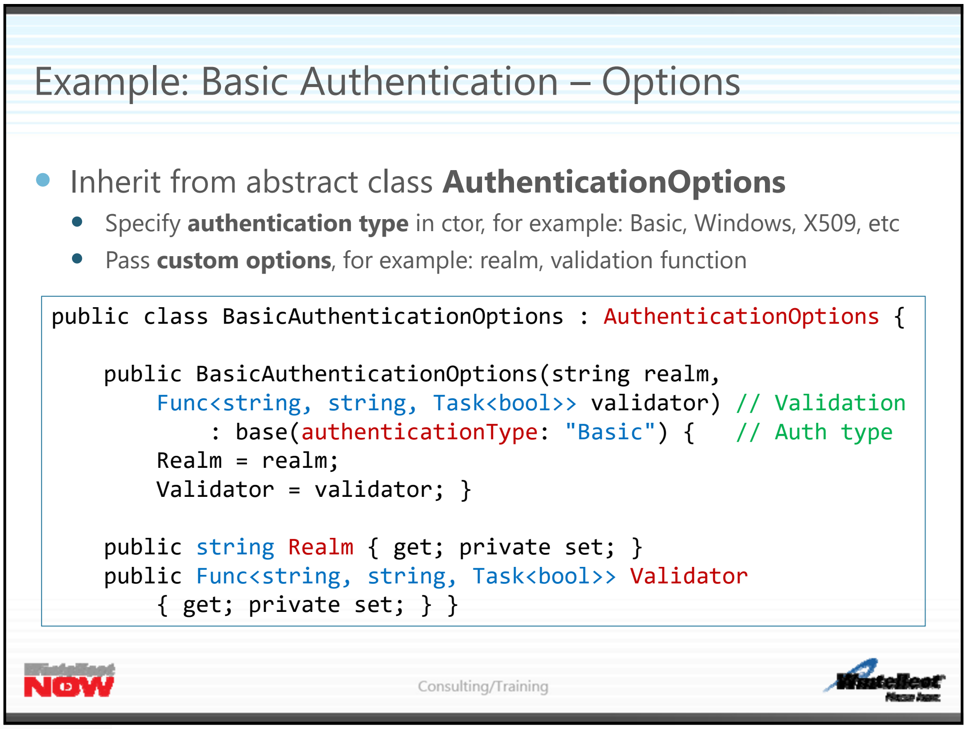
- Client sends request with **base64 encoded credentials**: (username:password)
  - Considered an **ant-pattern**: client must *store password,* send on *every request*
  - Server must **validate credentials** on *every request,* defend *brute force attacks*

Authorization: username:password

Status Code 200: **Successful**

Status Code 401: **Unauthorized**

Consulting/Training

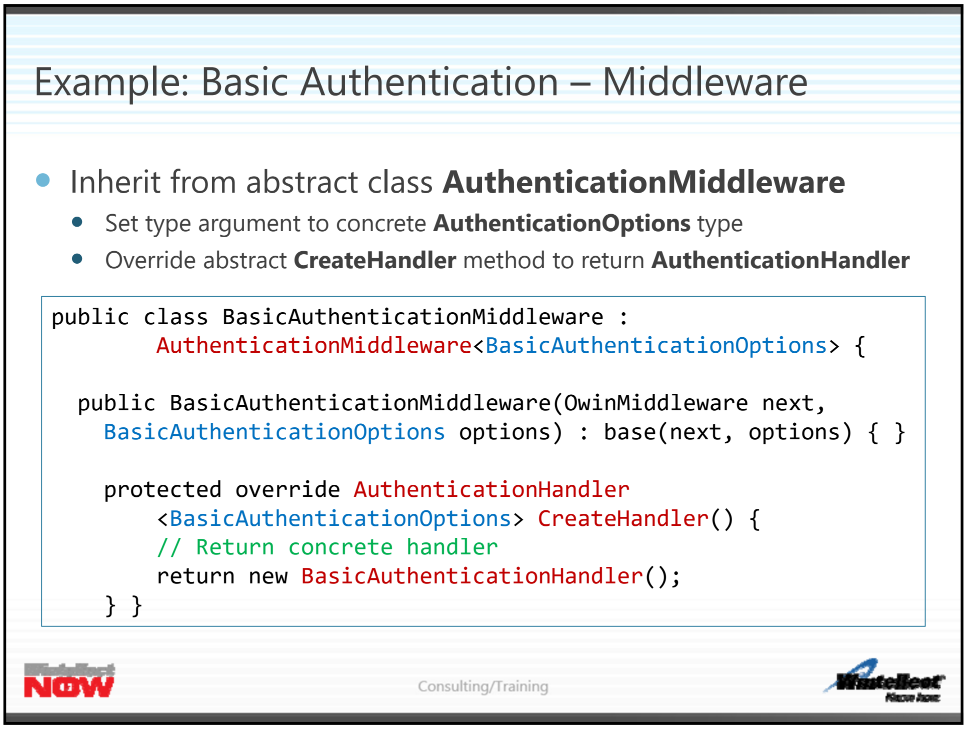


Example: Basic Authentication – Options

- Inherit from abstract class **AuthenticationOptions**
  - Specify **authentication type** in ctor, for example: Basic, Windows, X509, etc
  - Pass **custom options**, for example: realm, validation function

```
public class BasicAuthenticationOptions : AuthenticationOptions {

    public BasicAuthenticationOptions(string realm,
        Func<string, string, Task<bool>> validator) // Validation
            : base(authenticationType: "Basic") {    // Auth type
        Realm = realm;
        Validator = validator; }

    public string Realm { get; private set; }
    public Func<string, string, Task<bool>> Validator
        { get; private set; } }
```

Consulting/Training

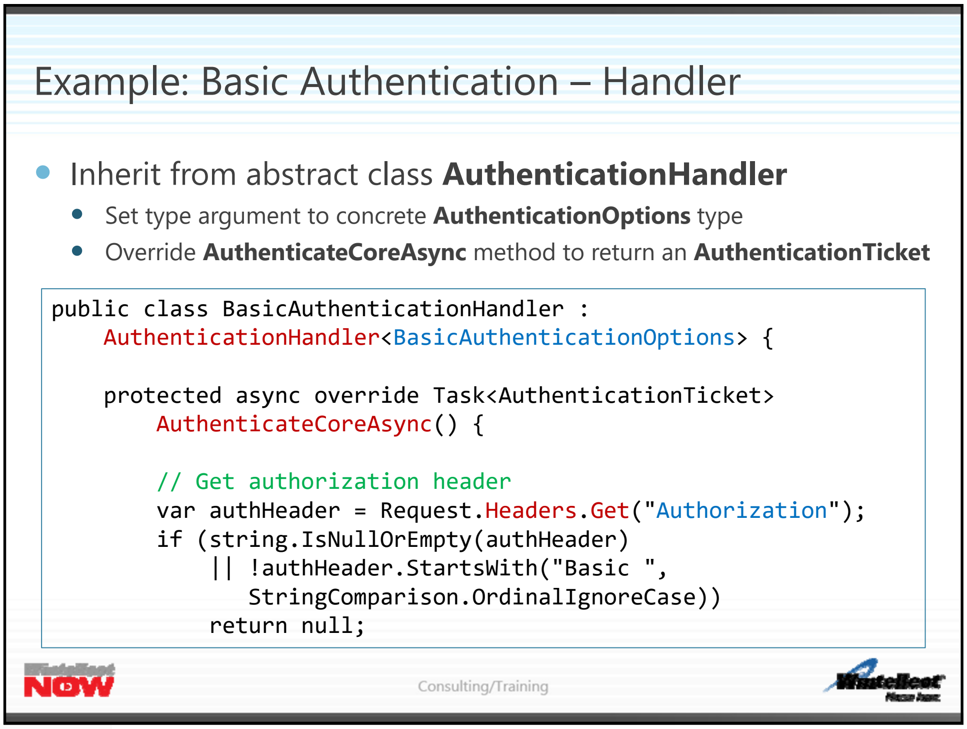


Example: Basic Authentication – Middleware

- Inherit from abstract class **AuthenticationMiddleware**
  - Set type argument to concrete **AuthenticationOptions** type
  - Override abstract **CreateHandler** method to return **AuthenticationHandler**

```
public class BasicAuthenticationMiddleware :
        AuthenticationMiddleware<BasicAuthenticationOptions> {

  public BasicAuthenticationMiddleware(OwinMiddleware next,
    BasicAuthenticationOptions options) : base(next, options) { }

    protected override AuthenticationHandler
        <BasicAuthenticationOptions> CreateHandler() {
        // Return concrete handler
        return new BasicAuthenticationHandler();
    } }
```

Consulting/Training

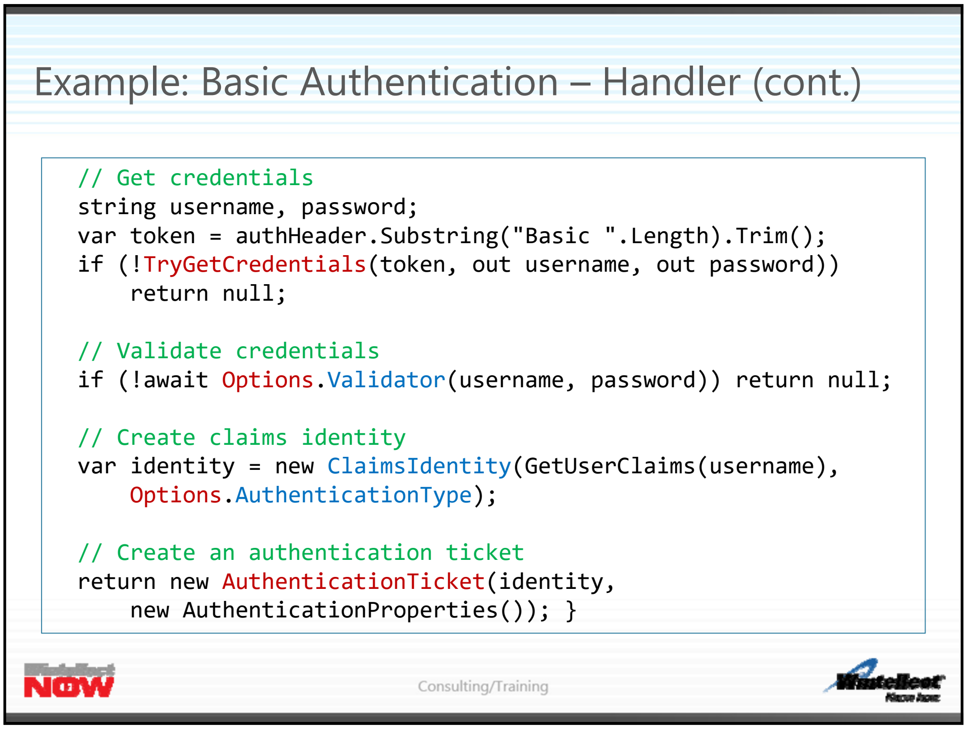


Example: Basic Authentication – Handler

- Inherit from abstract class **AuthenticationHandler**
  - Set type argument to concrete **AuthenticationOptions** type
  - Override **AuthenticateCoreAsync** method to return an **AuthenticationTicket**

```
public class BasicAuthenticationHandler :
    AuthenticationHandler<BasicAuthenticationOptions> {

    protected async override Task<AuthenticationTicket>
        AuthenticateCoreAsync() {

        // Get authorization header
        var authHeader = Request.Headers.Get("Authorization");
        if (string.IsNullOrEmpty(authHeader)
            || !authHeader.StartsWith("Basic ",
               StringComparison.OrdinalIgnoreCase))
            return null;
```

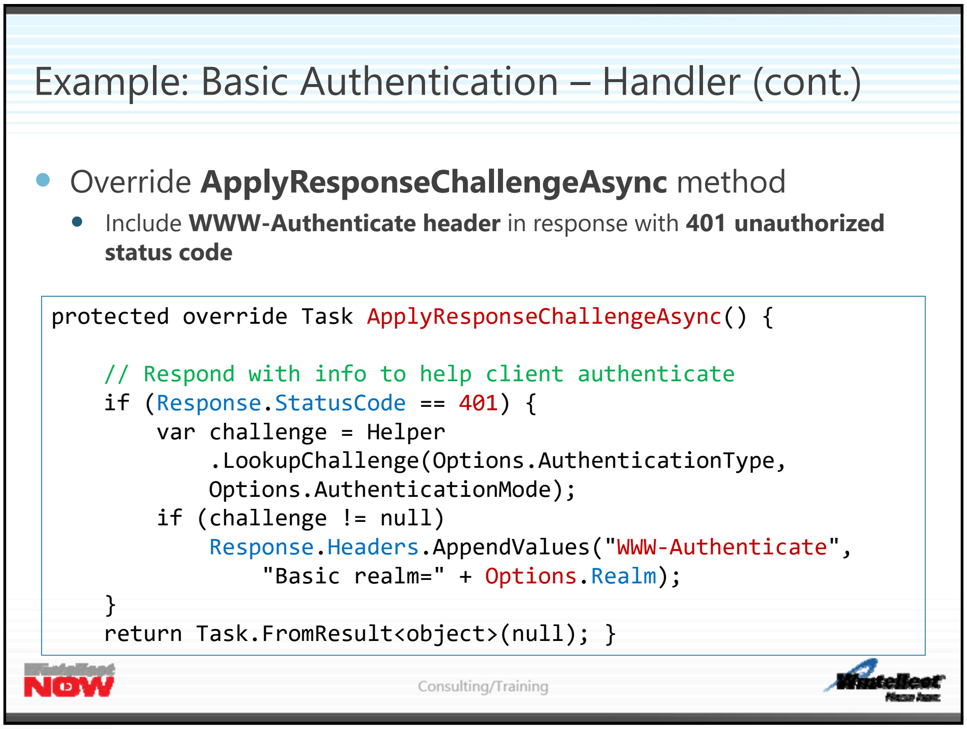
Consulting/Training



Example: Basic Authentication – Handler (cont.)

```
// Get credentials
string username, password;
var token = authHeader.Substring("Basic ".Length).Trim();
if (!TryGetCredentials(token, out username, out password))
    return null;

// Validate credentials
if (!await Options.Validator(username, password)) return null;

// Create claims identity
var identity = new ClaimsIdentity(GetUserClaims(username),
    Options.AuthenticationType);

// Create an authentication ticket
return new AuthenticationTicket(identity,
    new AuthenticationProperties()); }
```

Consulting/Training

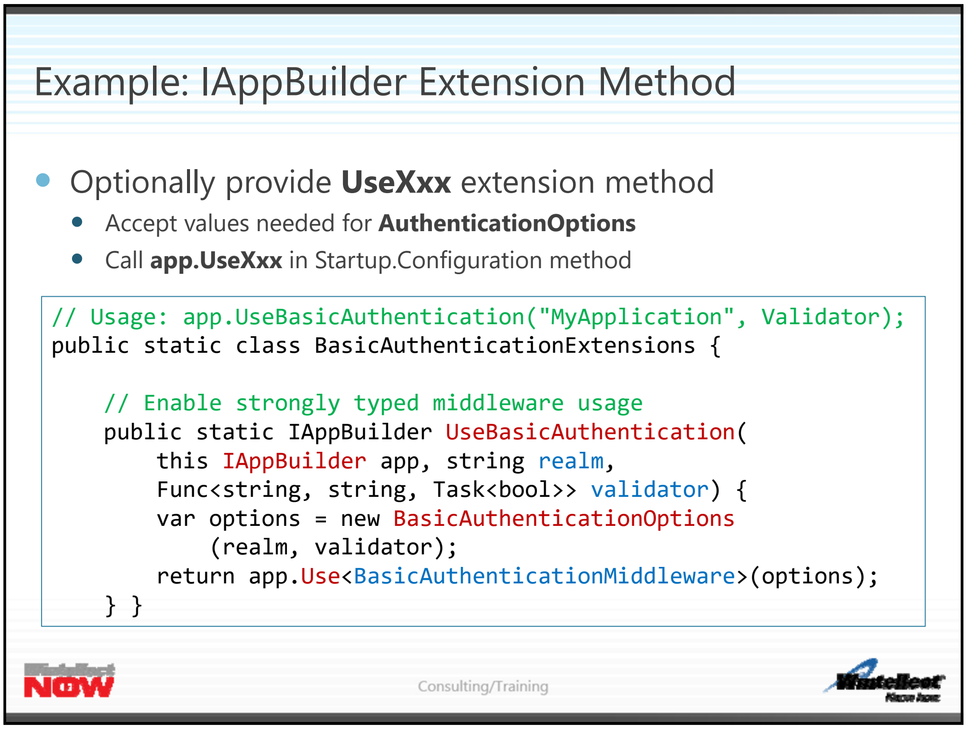


Example: Basic Authentication – Handler (cont.)

- Override **ApplyResponseChallengeAsync** method
  - Include **WWW-Authenticate header** in response with **401 unauthorized status code**

```
protected override Task ApplyResponseChallengeAsync() {

    // Respond with info to help client authenticate
    if (Response.StatusCode == 401) {
        var challenge = Helper
            .LookupChallenge(Options.AuthenticationType,
            Options.AuthenticationMode);
        if (challenge != null)
            Response.Headers.AppendValues("WWW-Authenticate",
                "Basic realm=" + Options.Realm);
    }
    return Task.FromResult<object>(null); }
```

Consulting/Training

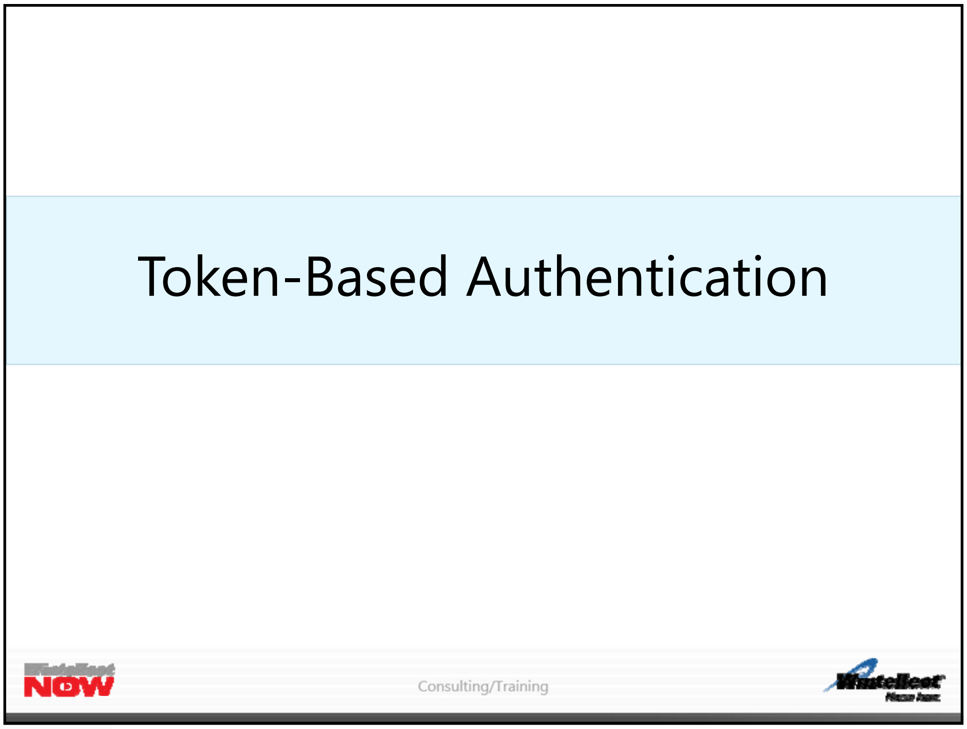


Example: IAppBuilder Extension Method

- Optionally provide **UseXxx** extension method
  - Accept values needed for **AuthenticationOptions**
  - Call **app.UseXxx** in Startup.Configuration method

```
// Usage: app.UseBasicAuthentication("MyApplication", Validator);
public static class BasicAuthenticationExtensions {

    // Enable strongly typed middleware usage
    public static IAppBuilder UseBasicAuthentication(
        this IAppBuilder app, string realm,
        Func<string, string, Task<bool>> validator) {
        var options = new BasicAuthenticationOptions
            (realm, validator);
        return app.Use<BasicAuthenticationMiddleware>(options);
    } }
```

Consulting/Training

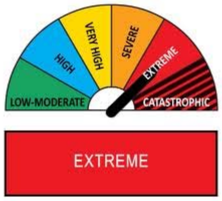


Token-Based Authentication

Consulting/Training

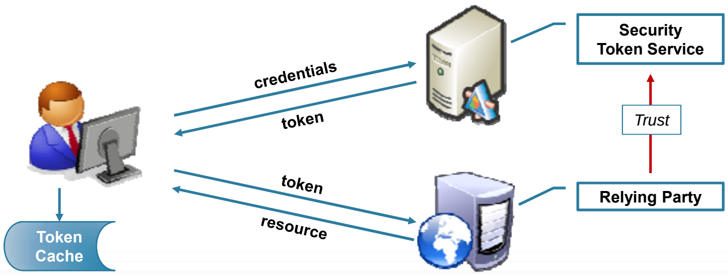
Problems with Basic Authentication

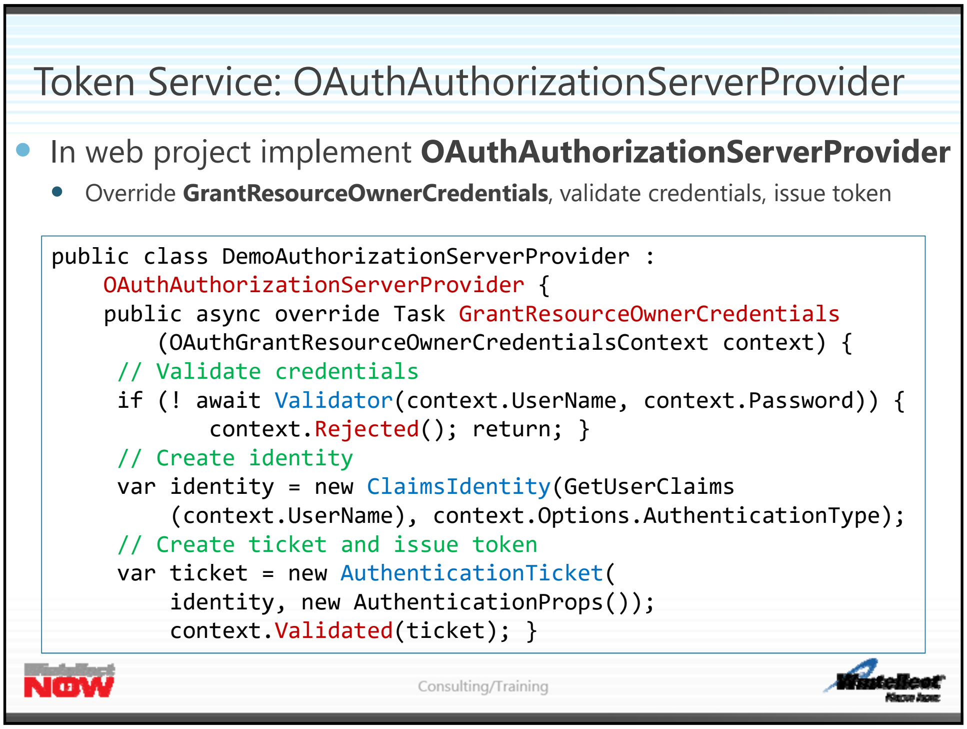
- Basic authentication is considered an **anti-pattern**
  - Username and password must be sent on **every request**
  - Client usually must *store the password*
  - Server must **validate credentials** on every request
  - Transport security (**TLS** / **SSL**) must be used to protect the password as it is sent over the wire



Token-Based Authentication

- Instead of performing authentication, web services can **delegate** authentication to an *external token service*
  - The token service **validates user credentials** and issues a **security token**
  - Client *caches the token* and use it to *authenticate requests* to a relying party



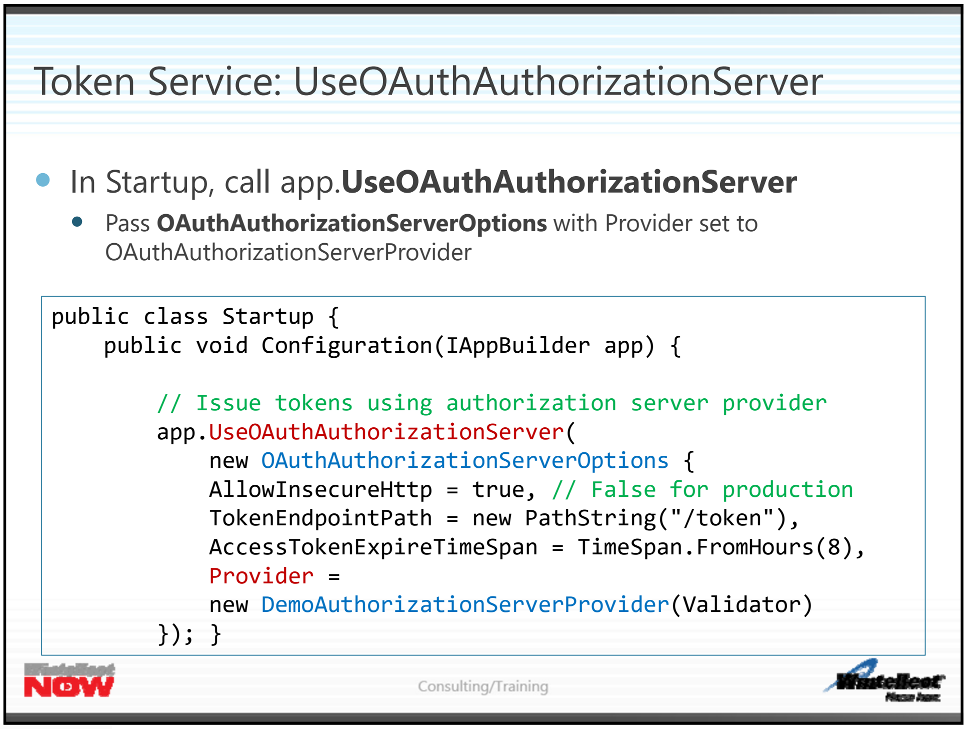


Token Service: OAuthAuthorizationServerProvider

- In web project implement **OAuthAuthorizationServerProvider**
  - Override **GrantResourceOwnerCredentials**, validate credentials, issue token

```
public class DemoAuthorizationServerProvider :
  OAuthAuthorizationServerProvider {
  public async override Task GrantResourceOwnerCredentials
    (OAuthGrantResourceOwnerCredentialsContext context) {
   // Validate credentials
   if (! await Validator(context.UserName, context.Password)) {
      context.Rejected(); return; }
   // Create identity
   var identity = new ClaimsIdentity(GetUserClaims
     (context.UserName), context.Options.AuthenticationType);
   // Create ticket and issue token
   var ticket = new AuthenticationTicket(
     identity, new AuthenticationProps());
     context.Validated(ticket); }
```

Consulting/Training

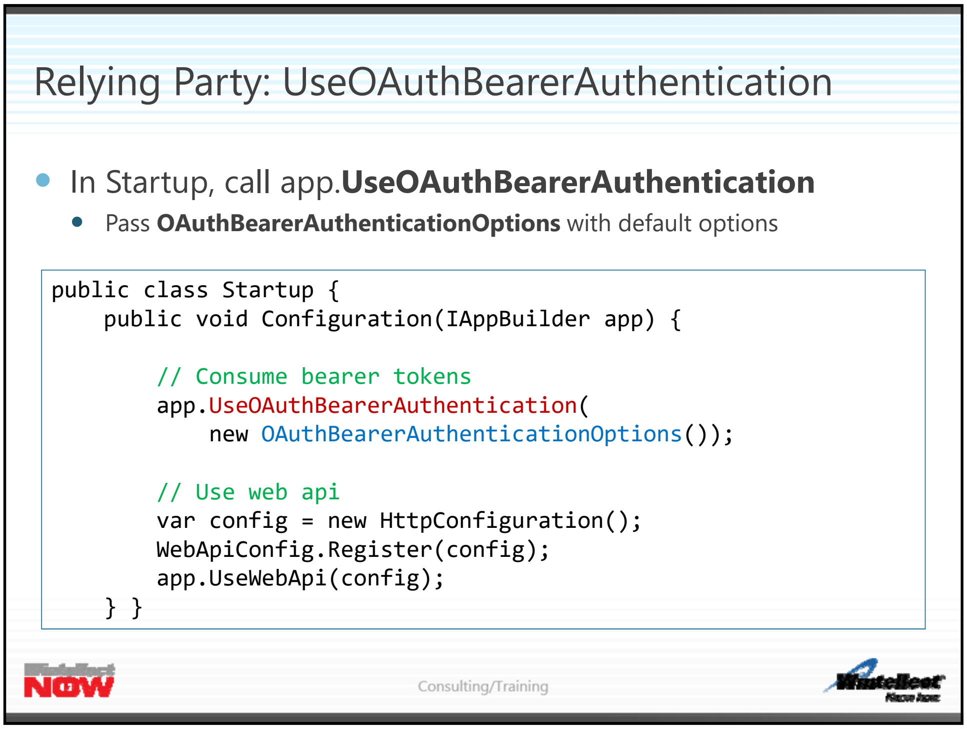


Token Service: UseOAuthAuthorizationServer

- In Startup, call app.**UseOAuthAuthorizationServer**
  - Pass **OAuthAuthorizationServerOptions** with Provider set to OAuthAuthorizationServerProvider

```
public class Startup {
  public void Configuration(IAppBuilder app) {

    // Issue tokens using authorization server provider
    app.UseOAuthAuthorizationServer(
      new OAuthAuthorizationServerOptions {
      AllowInsecureHttp = true, // False for production
      TokenEndpointPath = new PathString("/token"),
      AccessTokenExpireTimeSpan = TimeSpan.FromHours(8),
      Provider =
      new DemoAuthorizationServerProvider(Validator)
    }); }
```

Consulting/Training

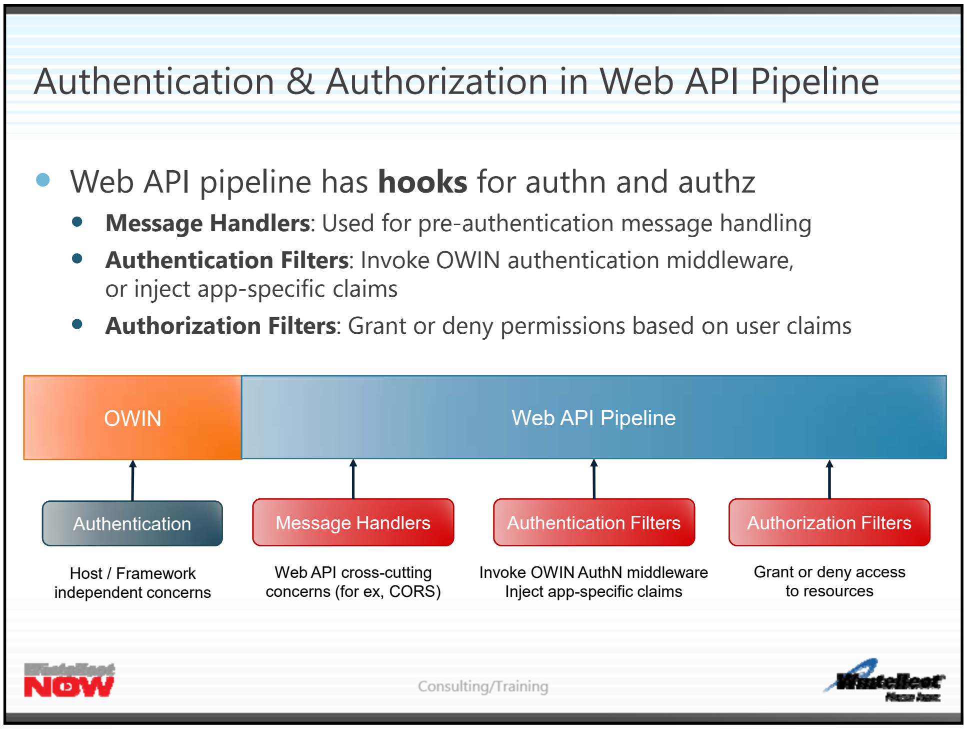


Relying Party: UseOAuthBearerAuthentication

- In Startup, call app.**UseOAuthBearerAuthentication**
  - Pass **OAuthBearerAuthenticationOptions** with default options

```
public class Startup {
    public void Configuration(IAppBuilder app) {

        // Consume bearer tokens
        app.UseOAuthBearerAuthentication(
            new OAuthBearerAuthenticationOptions());

        // Use web api
        var config = new HttpConfiguration();
        WebApiConfig.Register(config);
        app.UseWebApi(config);
    } }
```

Consulting/Training



Authentication & Authorization in Web API Pipeline

- Web API pipeline has **hooks** for authn and authz
  - **Message Handlers**: Used for pre-authentication message handling
  - **Authentication Filters**: Invoke OWIN authentication middleware, or inject app-specific claims
  - **Authorization Filters**: Grant or deny permissions based on user claims

OWIN

Web API Pipeline

Authentication

Host / Framework independent concerns

Message Handlers

Web API cross-cutting concerns (for ex, CORS)

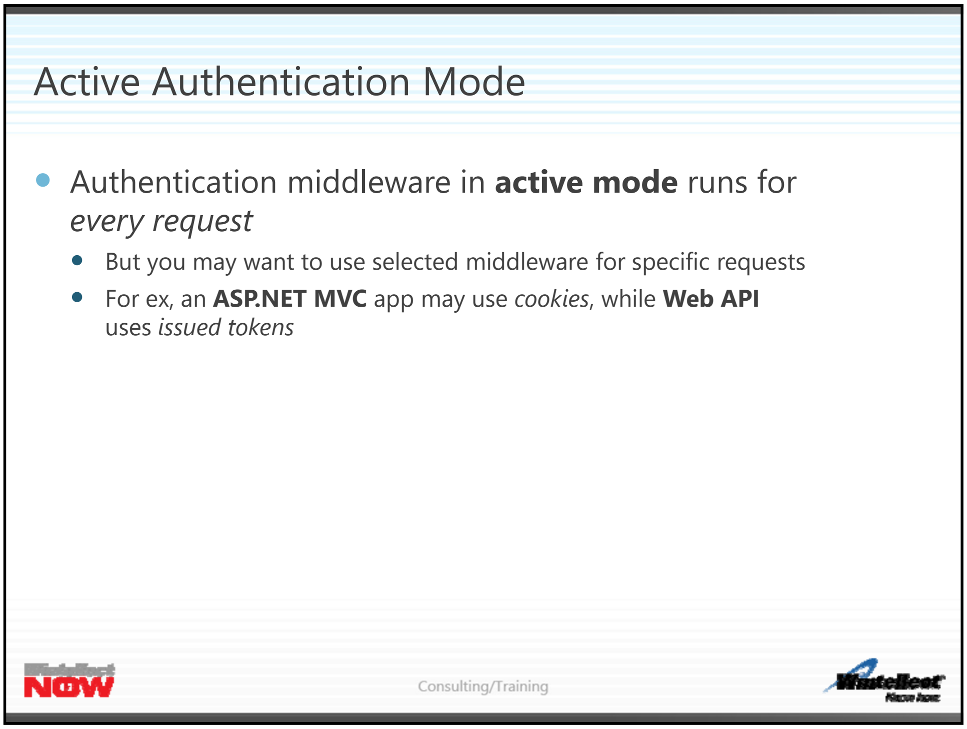
Authentication Filters

Invoke OWIN AuthN middleware Inject app-specific claims

Authorization Filters

Grant or deny access to resources

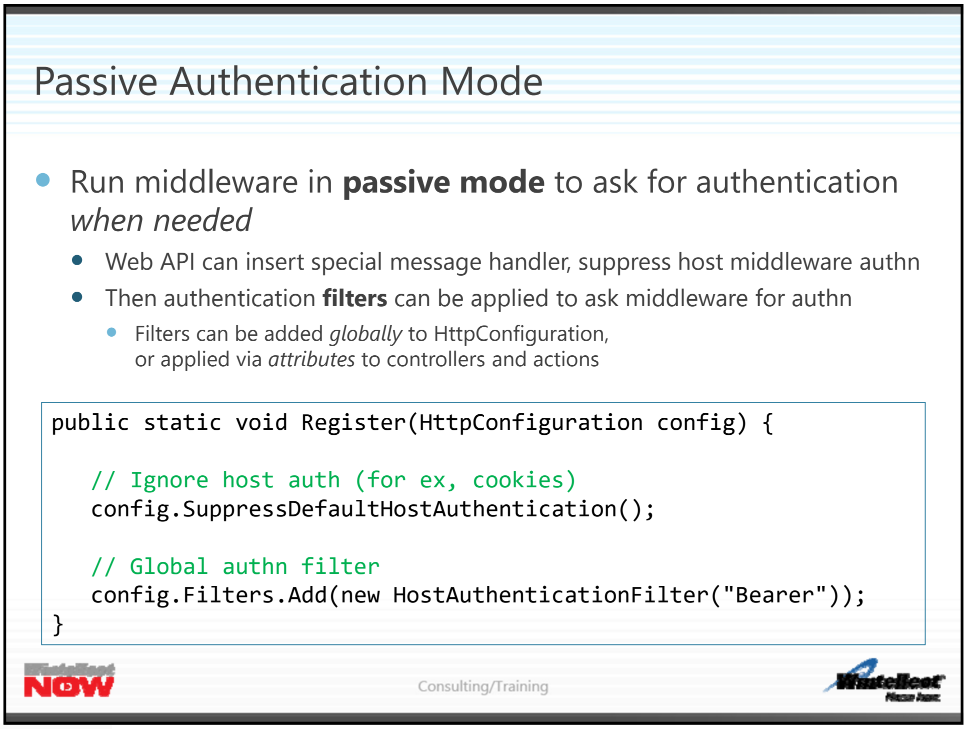
Consulting/Training



Active Authentication Mode

- Authentication middleware in **active mode** runs for *every request*
  - But you may want to use selected middleware for specific requests
  - For ex, an **ASP.NET MVC** app may use *cookies*, while **Web API** uses *issued tokens*

Consulting/Training

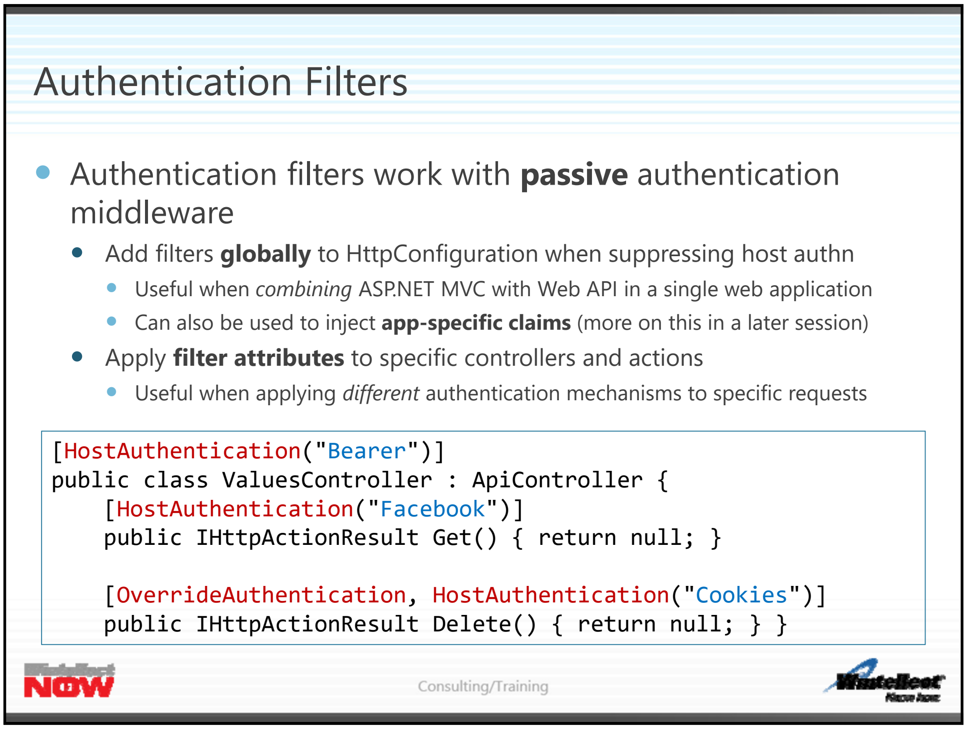


Passive Authentication Mode

- Run middleware in **passive mode** to ask for authentication *when needed*
  - Web API can insert special message handler, suppress host middleware authn
  - Then authentication **filters** can be applied to ask middleware for authn
    - Filters can be added *globally* to HttpConfiguration, or applied via *attributes* to controllers and actions

```
public static void Register(HttpConfiguration config) {

   // Ignore host auth (for ex, cookies)
   config.SuppressDefaultHostAuthentication();

   // Global authn filter
   config.Filters.Add(new HostAuthenticationFilter("Bearer"));
}
```

Consulting/Training



Authentication Filters

- Authentication filters work with **passive** authentication middleware
  - Add filters **globally** to HttpConfiguration when suppressing host authn
    - Useful when *combining* ASP.NET MVC with Web API in a single web application
    - Can also be used to inject **app-specific claims** (more on this in a later session)
  - Apply **filter attributes** to specific controllers and actions
    - Useful when applying *different* authentication mechanisms to specific requests

```
[HostAuthentication("Bearer")]
public class ValuesController : ApiController {
    [HostAuthentication("Facebook")]
    public IHttpActionResult Get() { return null; }

    [OverrideAuthentication, HostAuthentication("Cookies")]
    public IHttpActionResult Delete() { return null; } }
```

Consulting/Training

QUESTIONS

Consulting/Training

Validation and Testing with ASP.NET Web API

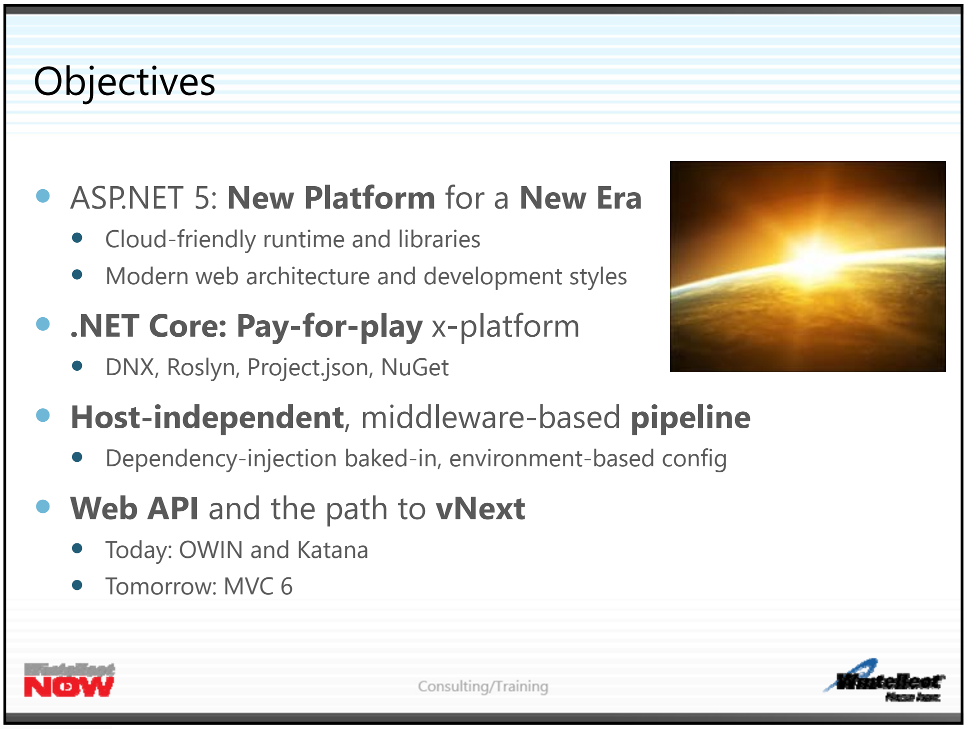
Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

Consulting/Training

Introduction to Web API in ASP.NET 5

Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

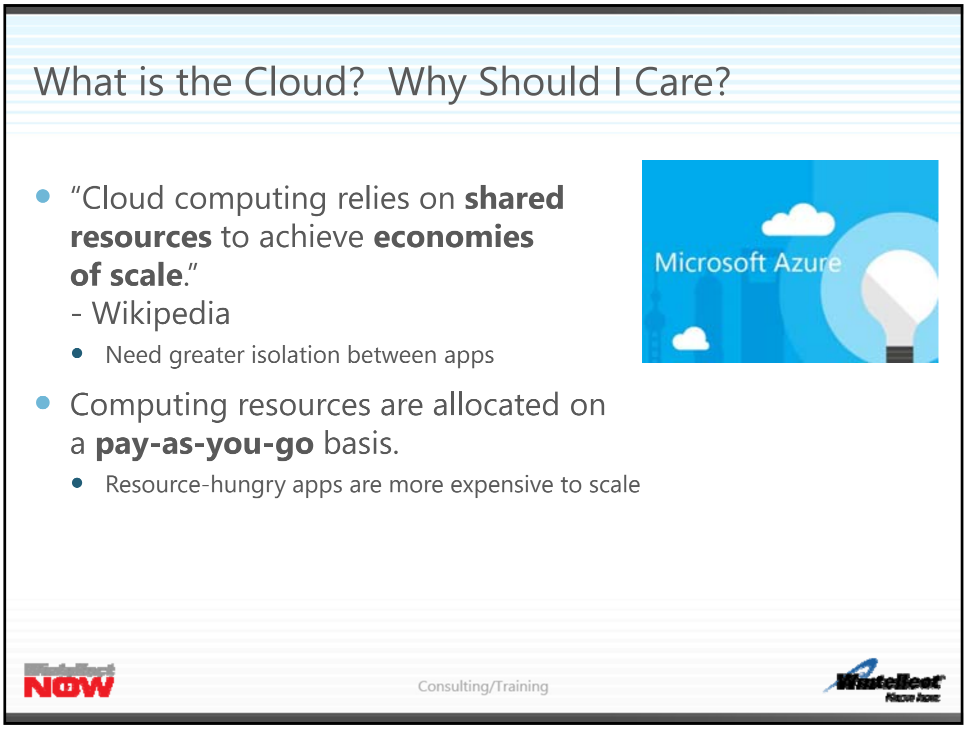
Consulting/Training



Objectives

- ASP.NET 5: **New Platform** for a **New Era**
  - Cloud-friendly runtime and libraries
  - Modern web architecture and development styles
- **.NET Core: Pay-for-play** x-platform
  - DNX, Roslyn, Project.json, NuGet
- **Host-independent**, middleware-based **pipeline**
  - Dependency-injection baked-in, environment-based config
- **Web API** and the path to **vNext**
  - Today: OWIN and Katana
  - Tomorrow: MVC 6

Consulting/Training

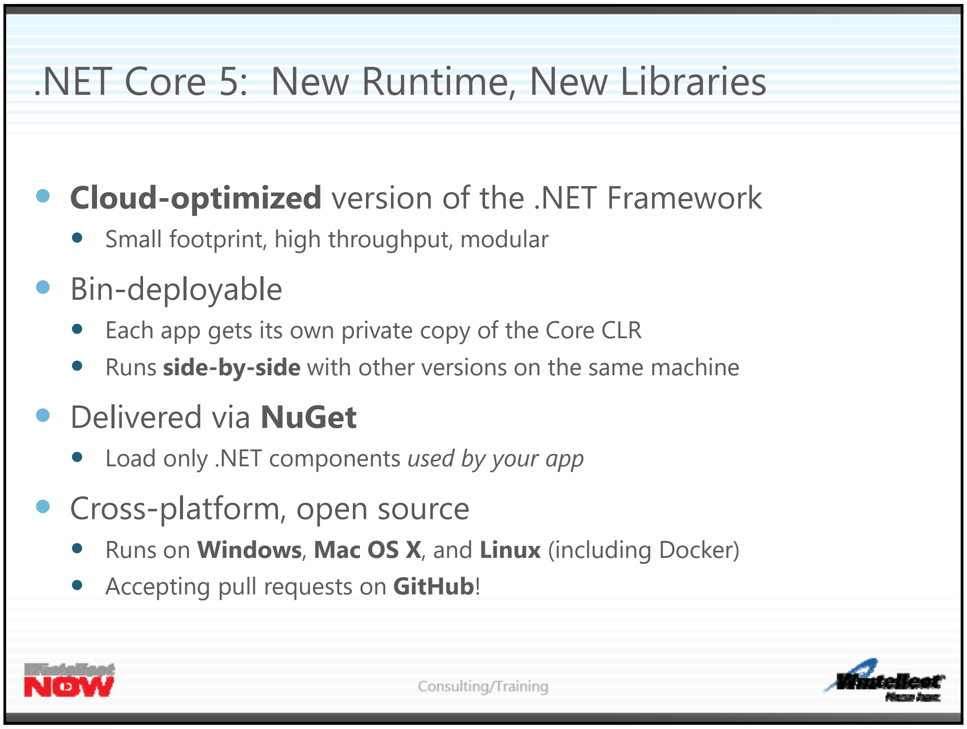


What is the Cloud? Why Should I Care?

- "Cloud computing relies on **shared resources** to achieve **economies of scale**."
  - Wikipedia
  - Need greater isolation between apps
- Computing resources are allocated on a **pay-as-you-go** basis.
  - Resource-hungry apps are more expensive to scale

Microsoft Azure

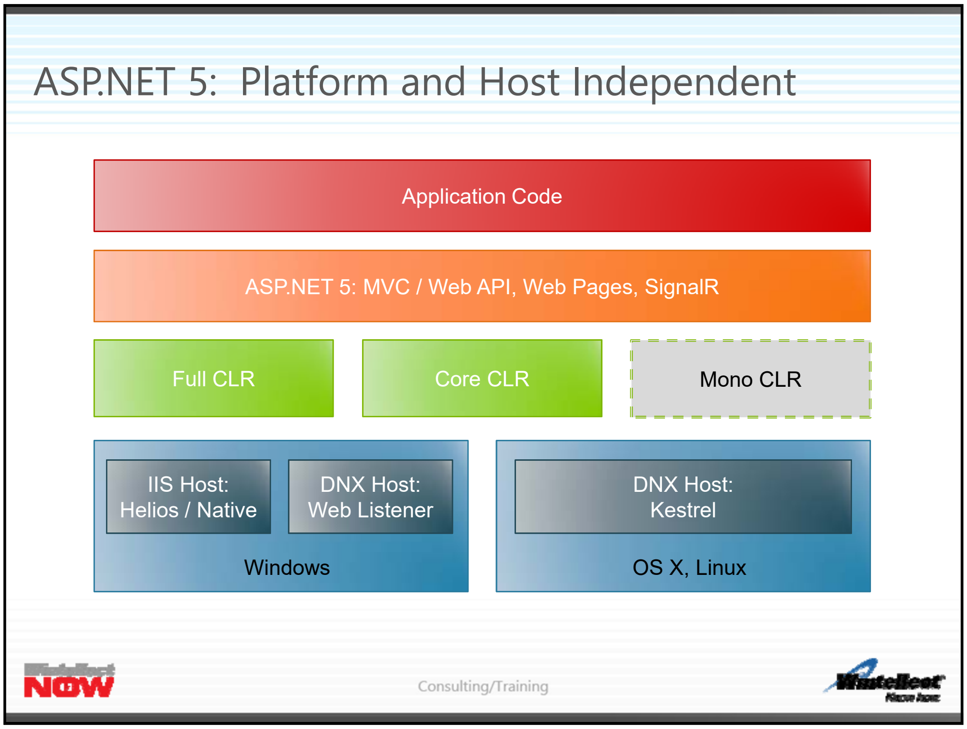
Consulting/Training



.NET Core 5: New Runtime, New Libraries

- **Cloud-optimized** version of the .NET Framework
  - Small footprint, high throughput, modular
- Bin-deployable
  - Each app gets its own private copy of the Core CLR
  - Runs **side-by-side** with other versions on the same machine
- Delivered via **NuGet**
  - Load only .NET components *used by your app*
- Cross-platform, open source
  - Runs on **Windows**, **Mac OS X**, and **Linux** (including Docker)
  - Accepting pull requests on **GitHub**!

Consulting/Training



ASP.NET 5: Platform and Host Independent

Application Code

ASP.NET 5: MVC / Web API, Web Pages, SignalR

Full CLR

Core CLR

Mono CLR

IIS Host:
Helios / Native

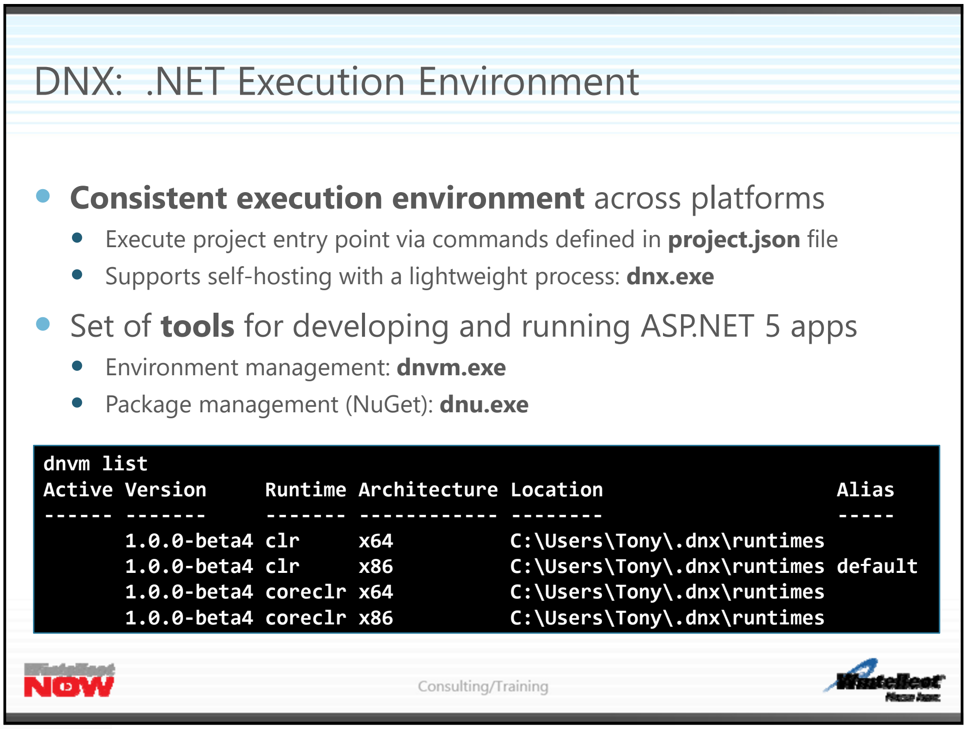
DNX Host:
Web Listener

Windows

DNX Host:
Kestrel

OS X, Linux

Consulting/Training

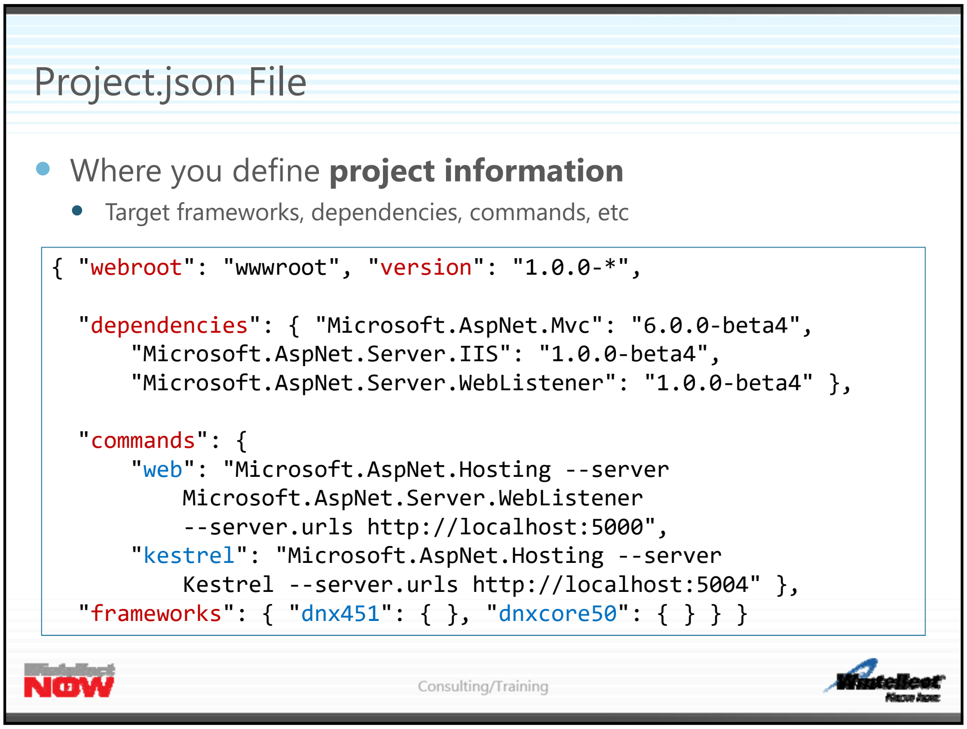


DNX: .NET Execution Environment

- **Consistent execution environment** across platforms
  - Execute project entry point via commands defined in **project.json** file
  - Supports self-hosting with a lightweight process: **dnx.exe**
- Set of **tools** for developing and running ASP.NET 5 apps
  - Environment management: **dnvm.exe**
  - Package management (NuGet): **dnu.exe**

```
dnvm list
Active Version     Runtime Architecture Location                     Alias
------ -------     ------- ------------ --------                     -----
       1.0.0-beta4 clr     x64          C:\Users\Tony\.dnx\runtimes
       1.0.0-beta4 clr     x86          C:\Users\Tony\.dnx\runtimes default
       1.0.0-beta4 coreclr x64          C:\Users\Tony\.dnx\runtimes
       1.0.0-beta4 coreclr x86          C:\Users\Tony\.dnx\runtimes
```

Consulting/Training

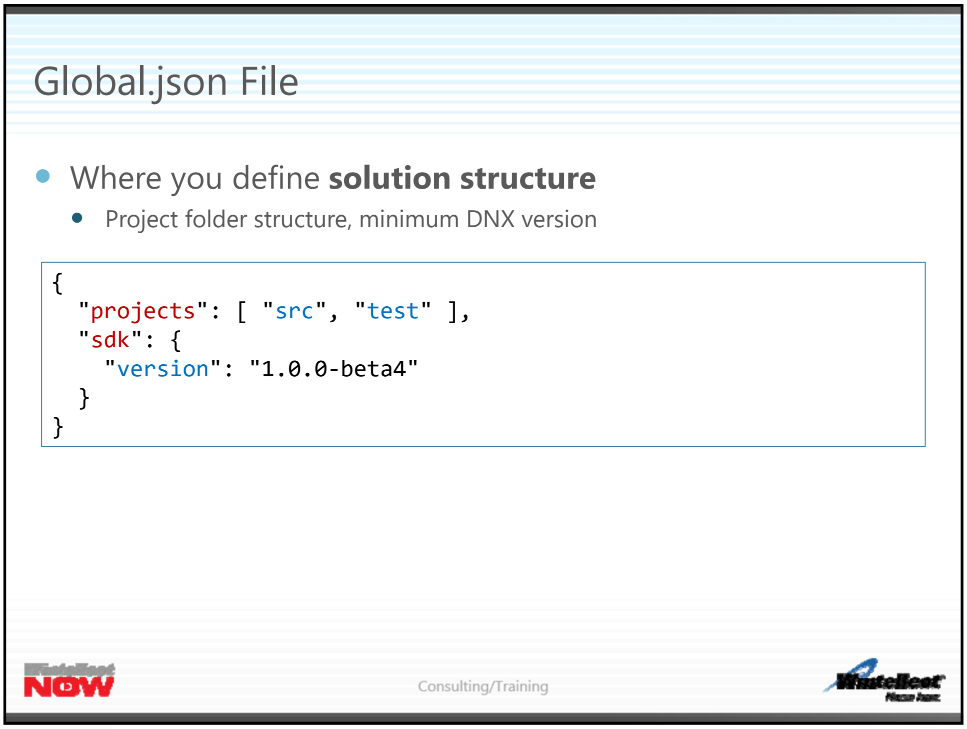


Project.json File

- Where you define **project information**
  - Target frameworks, dependencies, commands, etc

```
{ "webroot": "wwwroot", "version": "1.0.0-*",

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

  "commands": {
      "web": "Microsoft.AspNet.Hosting --server
          Microsoft.AspNet.Server.WebListener
          --server.urls http://localhost:5000",
      "kestrel": "Microsoft.AspNet.Hosting --server
          Kestrel --server.urls http://localhost:5004" },
  "frameworks": { "dnx451": { }, "dnxcore50": { } } }
```

Consulting/Training

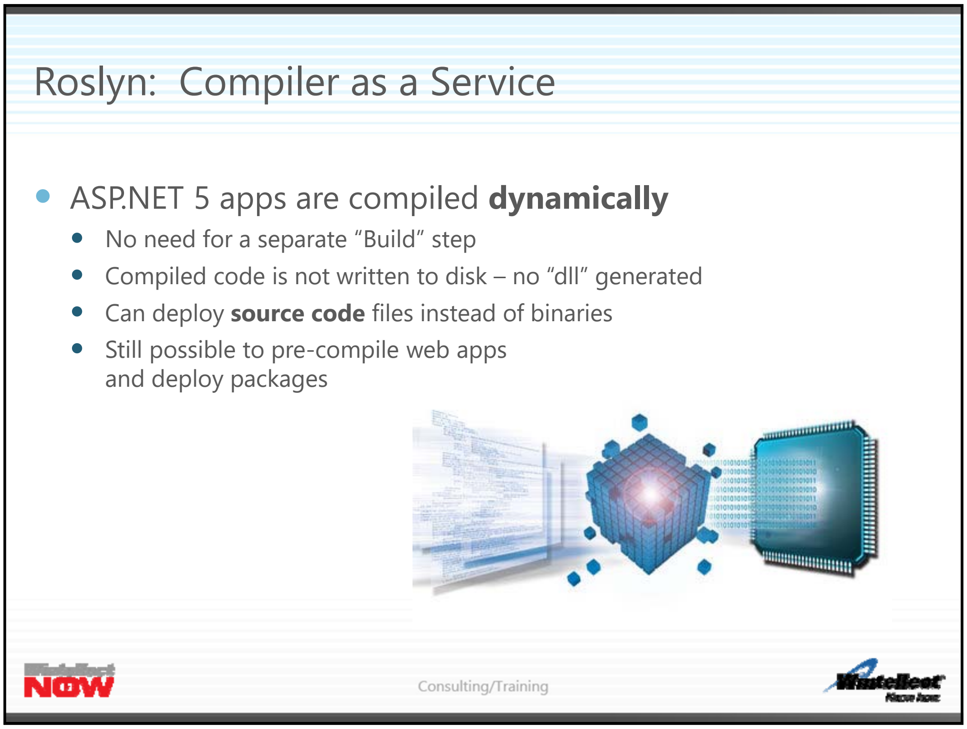


Global.json File

- Where you define **solution structure**
  - Project folder structure, minimum DNX version

```
{
  "projects": [ "src", "test" ],
  "sdk": {
    "version": "1.0.0-beta4"
  }
}
```

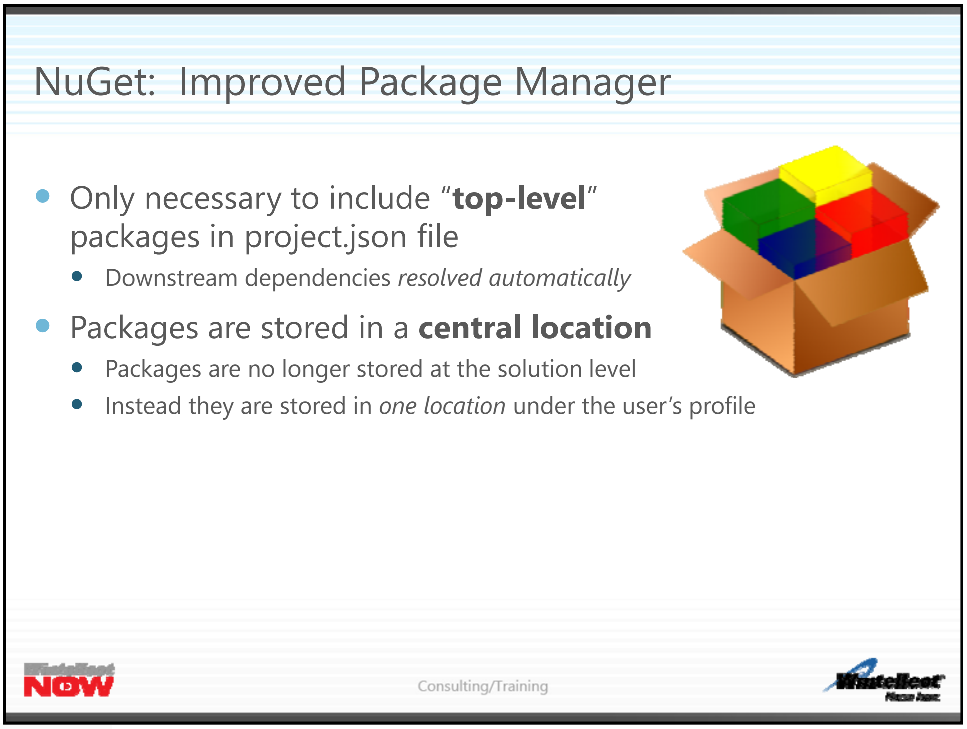
Consulting/Training



Roslyn: Compiler as a Service

- ASP.NET 5 apps are compiled **dynamically**
  - No need for a separate “Build” step
  - Compiled code is not written to disk – no “dll” generated
  - Can deploy **source code** files instead of binaries
  - Still possible to pre-compile web apps and deploy packages

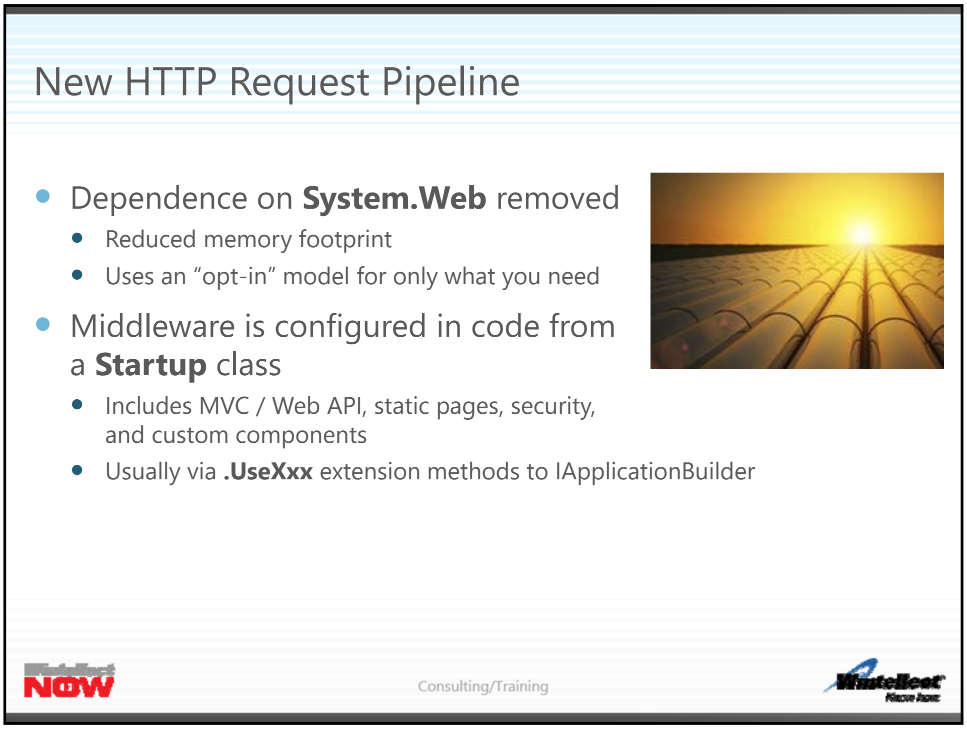
Consulting/Training



NuGet: Improved Package Manager

- Only necessary to include “**top-level**” packages in project.json file
  - Downstream dependencies *resolved automatically*
- Packages are stored in a **central location**
  - Packages are no longer stored at the solution level
  - Instead they are stored in *one location* under the user’s profile

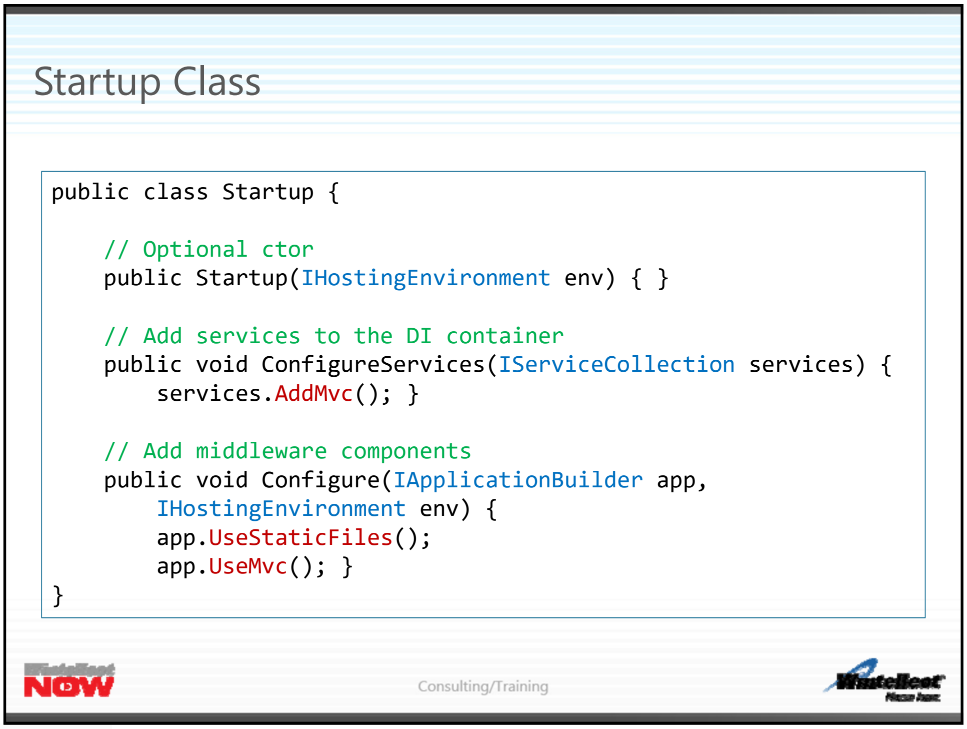
Consulting/Training



New HTTP Request Pipeline

- Dependence on **System.Web** removed
  - Reduced memory footprint
  - Uses an “opt-in” model for only what you need
- Middleware is configured in code from a **Startup** class
  - Includes MVC / Web API, static pages, security, and custom components
  - Usually via **.UseXxx** extension methods to IApplicationBuilder

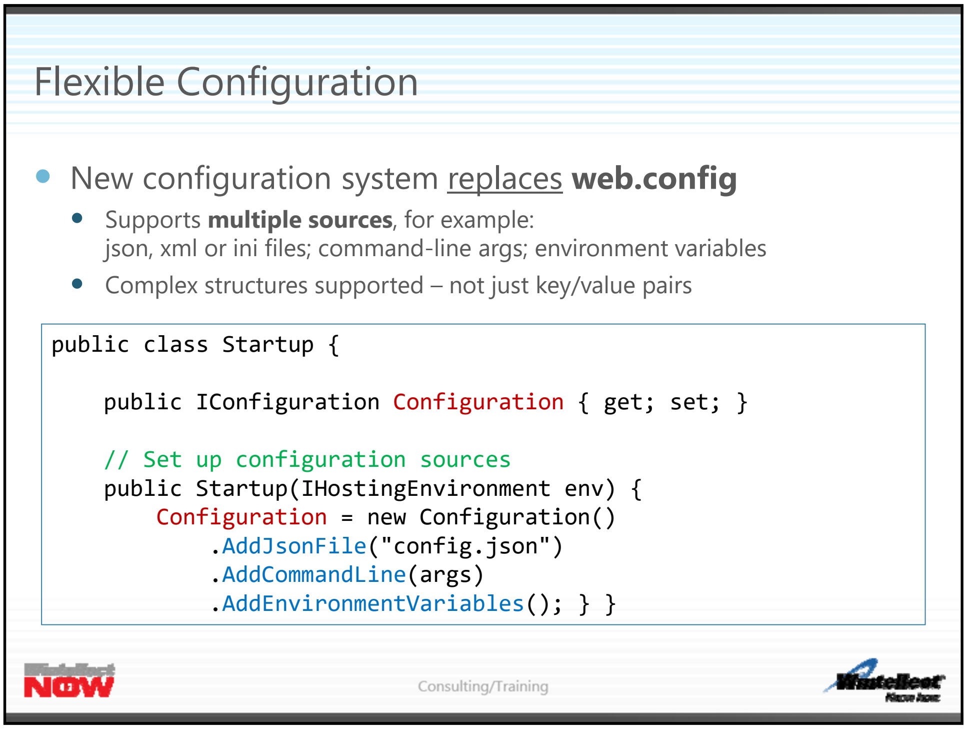
Consulting/Training



Startup Class

```
public class Startup {

    // Optional ctor
    public Startup(IHostingEnvironment env) { }

    // Add services to the DI container
    public void ConfigureServices(IServiceCollection services) {
        services.AddMvc(); }

    // Add middleware components
    public void Configure(IApplicationBuilder app,
        IHostingEnvironment env) {
        app.UseStaticFiles();
        app.UseMvc(); }
}
```

Consulting/Training

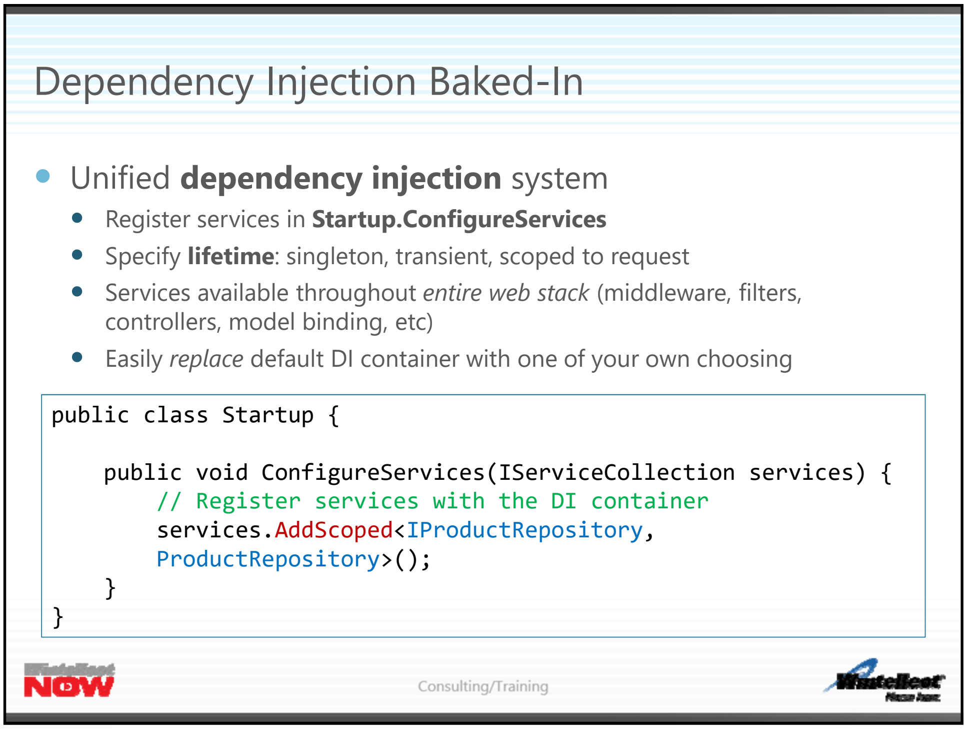


Flexible Configuration

- New configuration system <u>replaces</u> **web.config**
  - Supports **multiple sources**, for example:
    json, xml or ini files; command-line args; environment variables
  - Complex structures supported – not just key/value pairs

```
public class Startup {

    public IConfiguration Configuration { get; set; }

    // Set up configuration sources
    public Startup(IHostingEnvironment env) {
        Configuration = new Configuration()
            .AddJsonFile("config.json")
            .AddCommandLine(args)
            .AddEnvironmentVariables(); } }
```

Consulting/Training

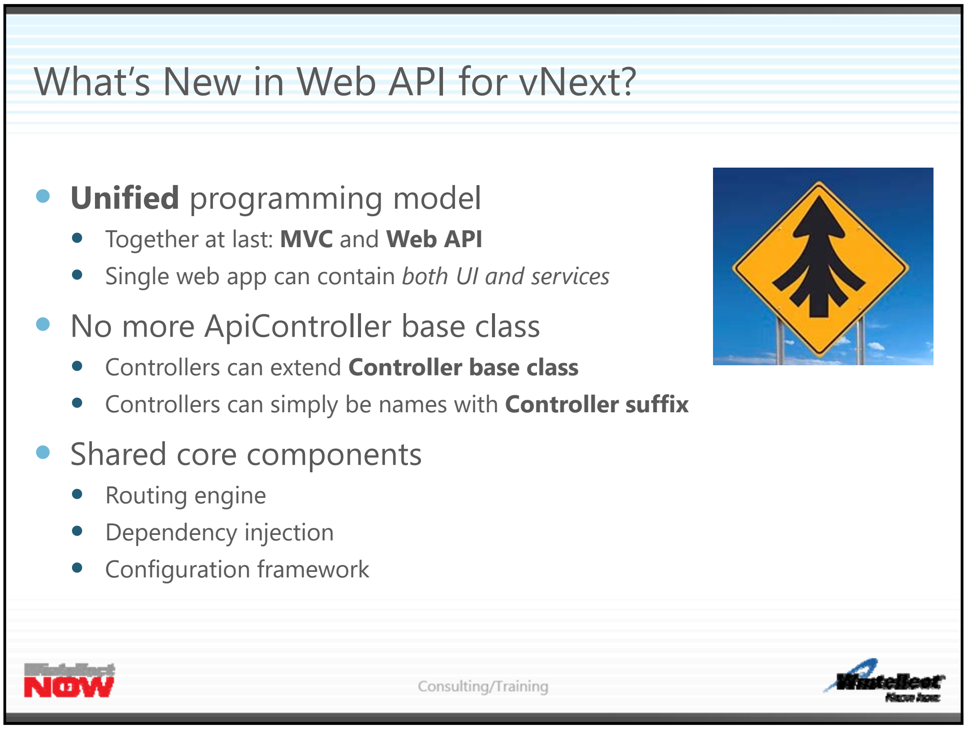


Dependency Injection Baked-In

- Unified **dependency injection** system
  - Register services in **Startup.ConfigureServices**
  - Specify **lifetime**: singleton, transient, scoped to request
  - Services available throughout *entire web stack* (middleware, filters, controllers, model binding, etc)
  - Easily *replace* default DI container with one of your own choosing

```
public class Startup {

    public void ConfigureServices(IServiceCollection services) {
        // Register services with the DI container
        services.AddScoped<IProductRepository,
        ProductRepository>();
    }
}
```

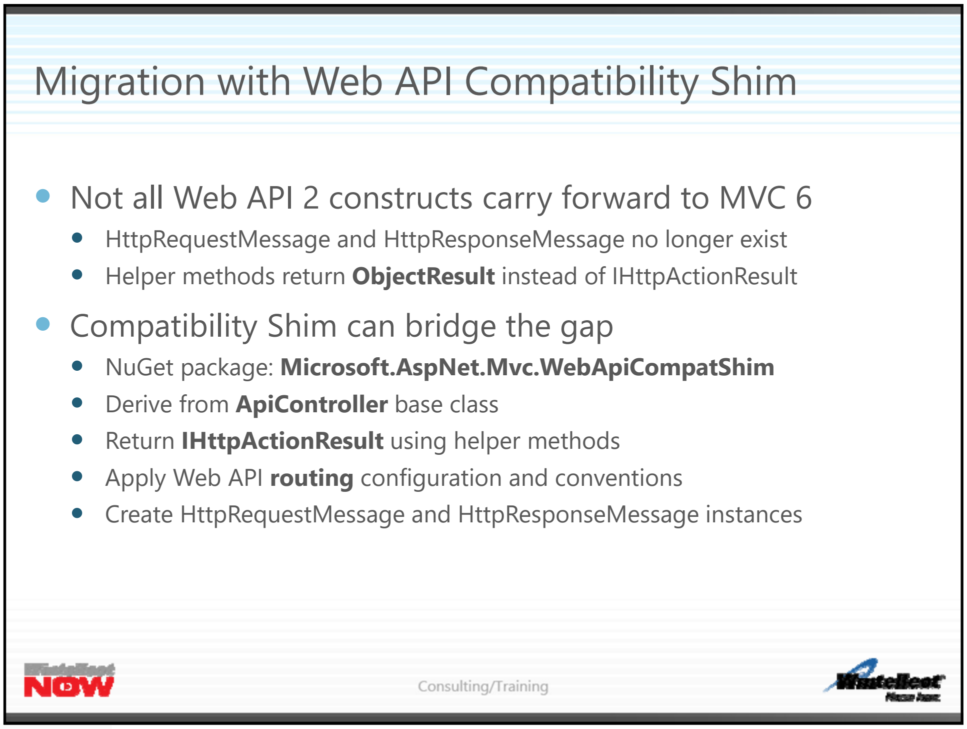
Consulting/Training



What’s New in Web API for vNext?

- **Unified** programming model
  - Together at last: **MVC** and **Web API**
  - Single web app can contain *both UI and services*
- No more ApiController base class
  - Controllers can extend **Controller base class**
  - Controllers can simply be names with **Controller suffix**
- Shared core components
  - Routing engine
  - Dependency injection
  - Configuration framework

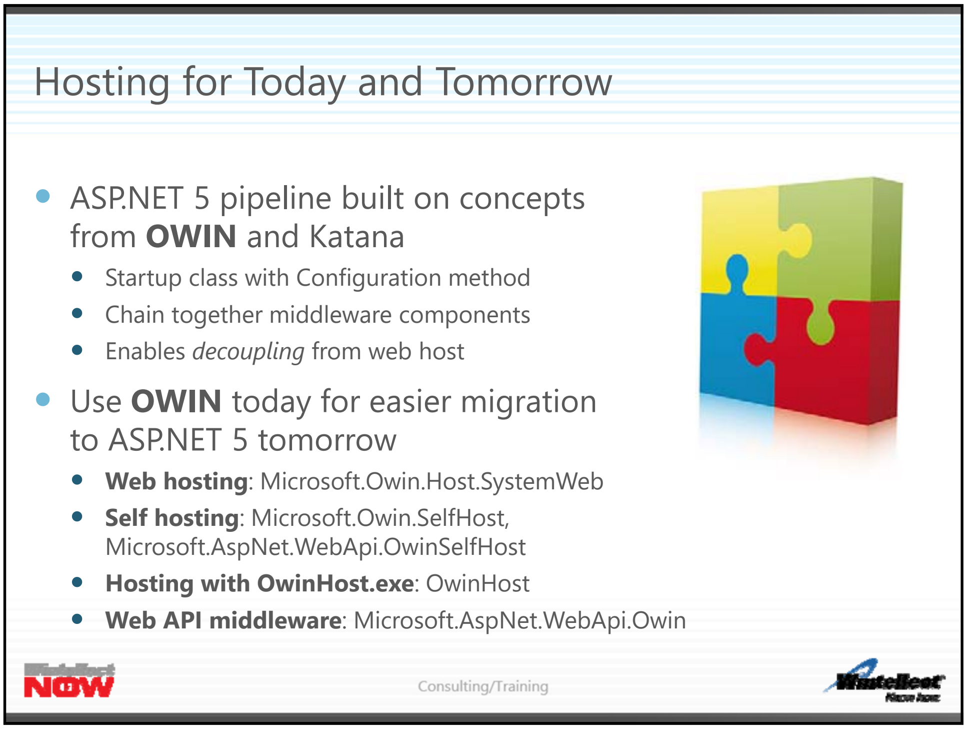
Consulting/Training



Migration with Web API Compatibility Shim

- Not all Web API 2 constructs carry forward to MVC 6
  - HttpRequestMessage and HttpResponseMessage no longer exist
  - Helper methods return **ObjectResult** instead of IHttpActionResult
- Compatibility Shim can bridge the gap
  - NuGet package: **Microsoft.AspNet.Mvc.WebApiCompatShim**
  - Derive from **ApiController** base class
  - Return **IHttpActionResult** using helper methods
  - Apply Web API **routing** configuration and conventions
  - Create HttpRequestMessage and HttpResponseMessage instances

Consulting/Training



Hosting for Today and Tomorrow

- ASP.NET 5 pipeline built on concepts from **OWIN** and Katana
  - Startup class with Configuration method
  - Chain together middleware components
  - Enables *decoupling* from web host
- Use **OWIN** today for easier migration to ASP.NET 5 tomorrow
  - **Web hosting**: Microsoft.Owin.Host.SystemWeb
  - **Self hosting**: Microsoft.Owin.SelfHost, Microsoft.AspNet.WebApi.OwinSelfHost
  - **Hosting with OwinHost.exe**: OwinHost
  - **Web API middleware**: Microsoft.AspNet.WebApi.Owin

Consulting/Training

QUESTIONS

Consulting/Training

Introduction to Web API in ASP.NET 5

Tony Sneed
Twitter: @tonysneed
Email: tsneed@wintellect.com
Blog: http://blog.tonysneed.com

Consulting/Training

NOTES

NOTES

NOTES

NOTES