- Jeff Ammons
- ammonsOnline.com

HTTP and C#

Working With HTTP/REST APIs in .Net

What Is HTTP?

- Hypertext Transport Protocol
- Data Transfer Layer
- Typically Sits On Top of TCP/IP
- Specification: http://www.w3.org/standards/techs/http#w3c_all

- Literally intended to transport hypertext documents

- Stateless

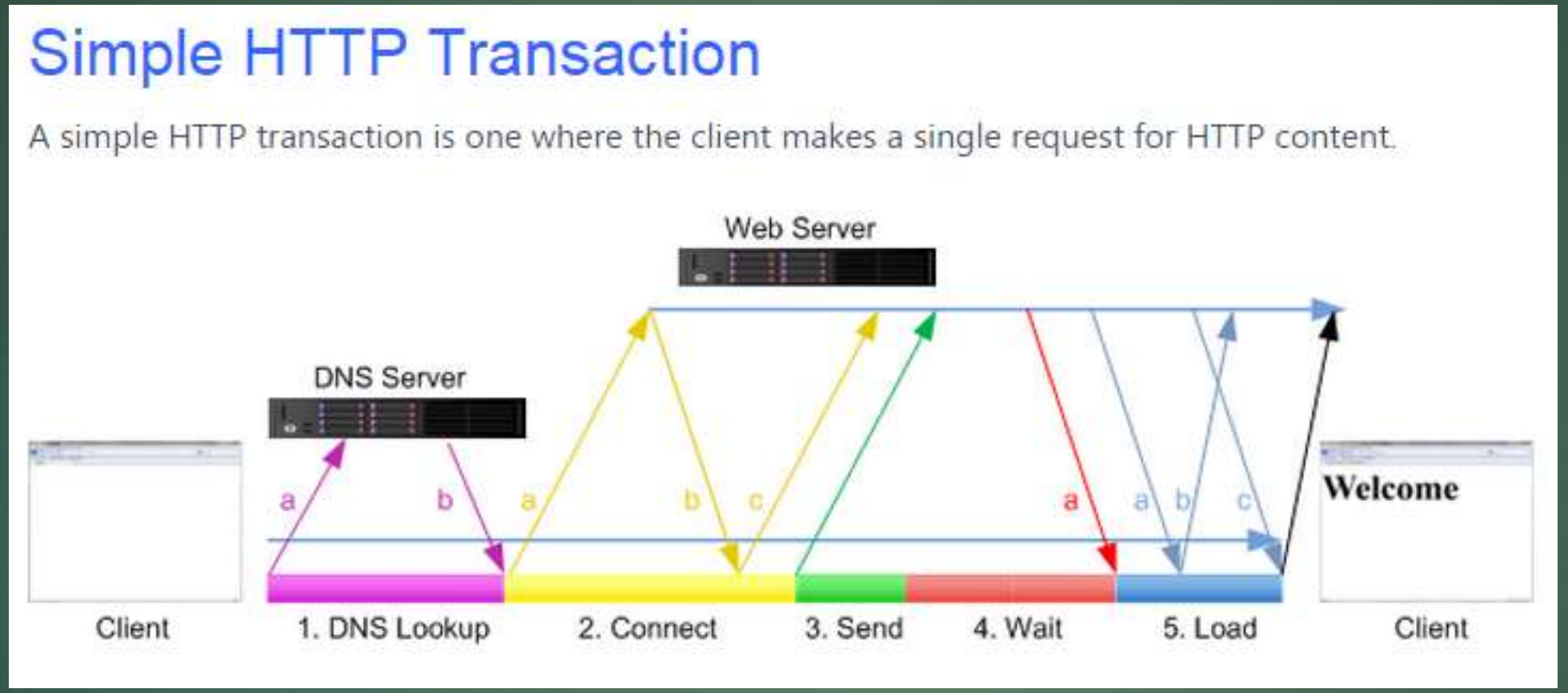
- Basis for World Wide Web

OSI Layers

OSI Model				
Layer		Data unit	Function[3]	Examples
Host layers	7. Application	Data	High-level APIs, including resource sharing, remote file access, directory services and virtual terminals	HTTP, FTP, SMTP
	6. Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption	
	5. Session		Managing communication sessions, i.e. continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes	
	4. Transport	Segments	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing	TCP, UDP, L2TP
Media layers	3. Network	Packet/Datagram	Structuring and managing a multi-node network, including addressing, routing and traffic control	IPv4, IPv6, IPsec, AppleTalk
	2. Data link	Bit/Frame	Reliable transmission of data frames between two nodes connected by a physical layer	PPP, IEEE 802.2
	1. Physical	Bit	Transmission and reception of raw bit streams over a physical medium	DSL, USB

- From Wikipedia: http://en.wikipedia.org/wiki/OSI_model
- OSI = Open System Interconnection

HTTP Visualized



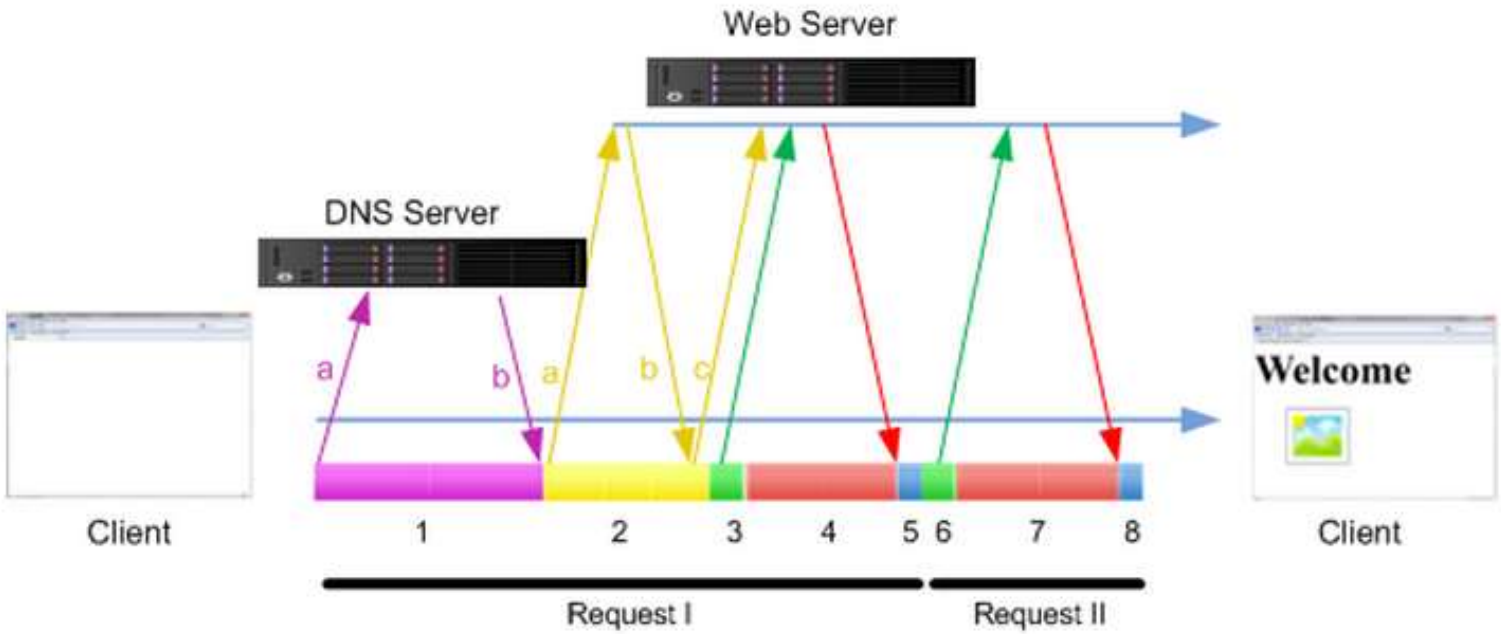
Note that reality is usually a bit more complex
With load balancers and web farms.

From Anatomy of an HTTP Transaction from the Catchpoint Blog:
http://blog.catchpoint.com/2010/09/17/anatomyhttp/

HTTP Visualized

Persistent HTTP Transactions

Persistent connections allow the browser / HTTP client to utilize the same connection for different object requests to the same hostname. The HTTP 1.1 protocol supports persistent connections natively, and does not require any specific HTTP header information. For HTTP 1.0, persistent connections are controlled via the Keep-Alive HTTP header.



From Anatomy of an HTTP Transaction from the Catchpoint Blog: http://blog.catchpoint.com/2010/09/17/anatomyhttp/

REST

- Representational State Transfer
- Interactive APIs
- Usually built on top of HTTP
- Makes use of established HTTP
  - Verbs
  - Headers
  - Status Codes

HTTP Verbs

- GET
- HEAD
- POST
- PUT
- DELETE
- TRACE
- OPTIONS
- CONNECT
- PATCH

HTTP Verbs (Most Used)

- GET
- HEAD
- POST
- PUT
- DELETE
- TRACE
- OPTIONS
- CONNECT
- PATCH

HTTP Verbs (Most Used)

GET	Idempotent	Return entity or entities
HEAD		
POST	Not Idempotent	Create new entity
PUT	Idempotent	Update or replace entity
DELETE	Idempotent	Remove entity
TRACE		
OPTIONS		
CONNECT		
PATCH		

Headers

- Instructions & Metadata
  - POST
  - Content-Length: 3
  - Connection: Keep-Alive

Status Codes

- **1xxx Informational**
  - 100 Continue
- **2xxx Success**
  - 200 OK
  - 201 Created
  - 202 Accepted
- **3xxx Redirection**
  - 301 Moved Permanently
- **4xxx Client Error**
  - 403 Forbidden
  - 404 Not Found
- **5xxx Server Error**
  - 500 Internal Server Error
  - 503 Service Unavailable

Sample HTTP Request

```
POST http://jeffasage.azurewebsites.net/api/Values/Post HTTP/1.1
Content-Type: application/json
Host: jeffasage.azurewebsites.net
Content-Length: 3
Expect: 100-continue
Connection: Keep-Alive

Foo
```

Sample HTTP Request

Headers	POST http://jeffasage.azurewebsites.net/api/Values/Post HTTP/1.1 Content-Type: application/json Host: jeffasage.azurewebsites.net Content-Length: 3 Expect: 100-continue Connection: Keep-Alive
Content	Foo

Microsoft’s (Traditional) View Of The World

Windows Desktop

Microsoft’s (Traditional) View Of The World

Windows Desktop

Windows Desktop On Mobile

Microsoft's (Traditional) View Of The World

Windows Desktop

Windows Desktop On Server

Windows Desktop On Mobile

Microsoft's (Traditional) View Of The World

- Desktop Windows Is The Default
- Everything Else Is A Variant

- Important Because
  - Explains Some Oddities in .Net
    - Default of 2 Outbound Connections Per host

Example From WebRequest Documentation:

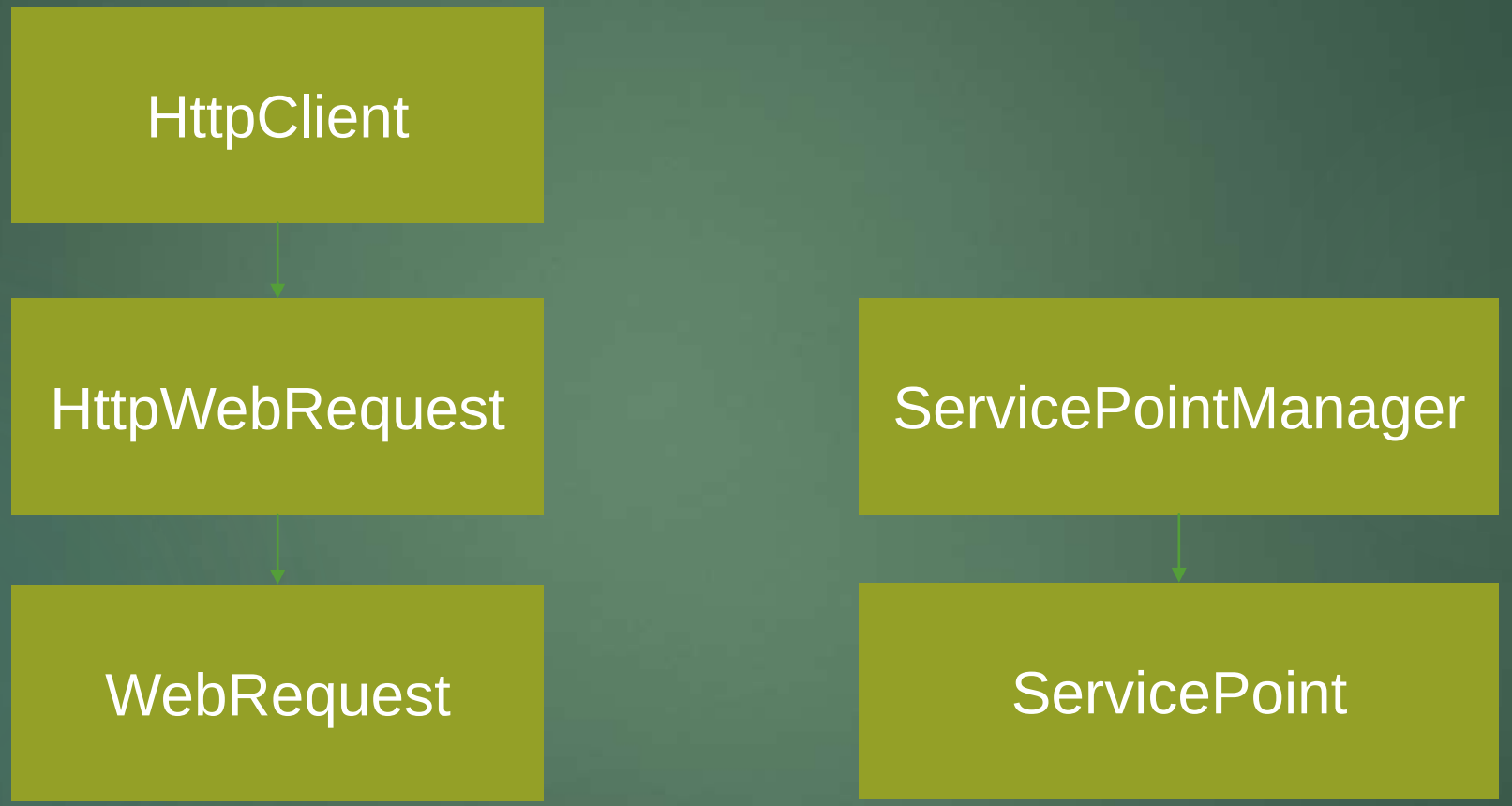
"The DefaultWebProxy property reads proxy settings from the app.config file.

If there is no config file, the current user's Internet Explorer (IE) proxy settings are used."

Microsoft's (New) View Of The World

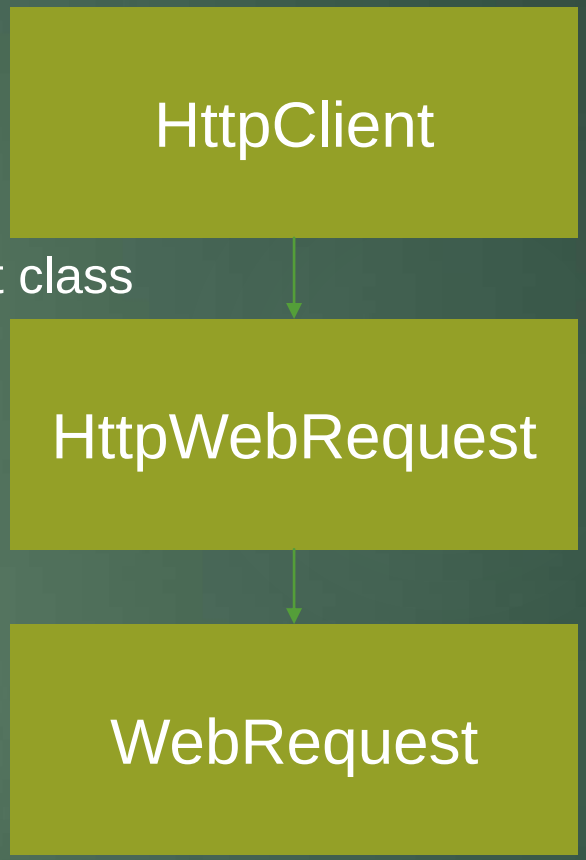
- Slowly adopting a less Windows-centric view
- Cross platform is becoming a priority
  - Open sourcing .Net code needed for server
  - Announced Official Linux and Mac .Net
  - ASP.Net 5 breaks MVC and Web API away from Web Forms
    - Drop desktop windows underpinnings
  - Github & Git for development in the open (explains the Git integration with TFS)
  - 20% of VMs on Azure are Linux

.Net Client Interactions With HTTP APIs



HttpClient vs WebRequest vs HttpWebRequest

- HttpClient is a wrapper around
- HttpWebRequest which is an implementation of the abstract class
- WebRequest



- If you have a choice for REST, choose HttpClient
  - Much simpler interface
  - Good level of abstraction
  - Implements IDisposable
    - Can be wrapped in using()
  - .Net 4.5 for full features
  - Built with async in mind

What About WebClient?

- WebClient also wraps HttpWebRequest
- Derived from Component
  - Primarily meant for Windows UI
  - Sporadically supported on server
  - Supports UI events like progress
- Works with older .Net versions
- Great for interacting with web pages in a GUI

HttpClient Example

```
using (var client = new HttpClient())
{
    client.BaseAddress = new Uri(WebApiBase);
    client.DefaultRequestHeaders.Accept.Clear();
    client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
    client.DefaultRequestHeaders.Add(“User-Agent”,”.NET Framework Example Client”);

    HttpResponseMessage response = await client.PostAsJsonAsync<string>("api/Values/Post", "Foo");

    Console.WriteLine("Request successful? {0}", response.IsSuccessStatusCode);
}
```

HTTPWebRequest Example

```
Stream dataStream = null;
WebResponse webResponse = null;

try
{
    var webRequestClient = (HttpWebRequest)WebRequest.Create(WebApiBase + "api/Values/Post");
    webRequestClient.UserAgent = ".NET Framework Example Client";
    webRequestClient.Method = "POST";
    webRequestClient.ContentType = "application/json; charset=utf-8";
    webRequestClient.Accept = “application/json”;

    byte[] contentArr = Encoding.UTF8.GetBytes("\"Foo\"");
    webRequestClient.ContentLength = contentArr.Length;

    dataStream = webRequestClient.GetRequestStream();
    dataStream.Write(contentArr, 0, contentArr.Length);
    dataStream.Close();

    webResponse = await webRequestClient.GetResponseAsync();

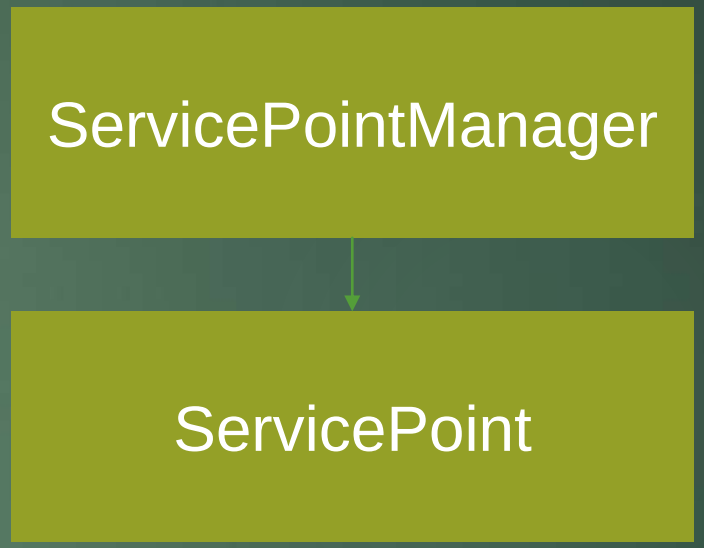
    var responseMessage = new HttpResponseMessage(((HttpWebResponse)webResponse).StatusCode);

    Console.WriteLine("Request successful? {0}", responseMessage.IsSuccessStatusCode);
}
finally
{
    if(dataStream != null)
        dataStream.Close();

    if (webResponse != null)
        webResponse.Close();
}
```

ServicePoint and ServicePointManager

- ServicePoint represents a connection to a server
  - Creates socket connection (potentially 2 if it sees IPv6)
  - Ultimately hits Winsock (native code)
  - Maintains list of how many connections exist
  - Enforce max connections
  - Options
- ServicePointManager manages a list of ServicePoint objects
  - STATIC! Remember that! *Practically* global in scope (per app).
  - Hold default settings
  - Factory
    - Return Existing or



Fun Fact: Socket class has 218 instances of the word “unsafe”

ServicePointManager Options (Defaults)

- CheckCertificateRevocationList
- DefaultConnectionLimit
- DnsRefreshTimeout
- EnableDnsRoundRobin
- EncryptionPolicy
- Expect100Continue
- MaxServicePointIdleTime
- MaxServicePoints
- SecurityProtocol
- ServerCertificateValidationCallback
- UseNagleAlgorithm

ServicePointManager Options (Defaults)

- CheckCertificateRevocationList
  - Should server certificates be checked against the certificate authority revocation list
  - Default is false
- DefaultConnectionLimit
  - Set the default max number of connections to a URI
  - Defaults to 2!
    - Browsers are supposed to limit themselves to 2
  - Under ASP defaults to 10
- DnsRefreshTimeout
  - How long should a DNS resolution be considered valid?
  - In milliseconds
  - Default is 120,000 (2 minutes)
- EnableDnsRoundRobin
  - Should DNS cycle through all IP addresses (if more than 1)
  - Default is false
  - Usage Example: DNS loadbalancer

ServicePointManager Options (Defaults)

- EncryptionPolicy
  - Whether or not to encrypt HTTPS traffic!
    - AllowNoEncryption
    - NoEncryption
    - RequireEncryption (Blessed be, at least this is the default)
  - Remember this is for all ServicePoints (connections) for this app!
- Expect100Continue
  - Inform server that 100-Continue behavior is expected
    - Client will send HTTP header ONLY
    - Server will reply with status 100-Continue
    - Client will send content
  - Good for large payloads
    - Don’t send payload if server isn’t ready
  - Not uniformly implemented by servers
  - Default is true
  - Applies to PUT and POST
  - Writes HTTP Header: “Expect: 100-Continue”

ServicePointManager Options (Defaults)

- MaxServicePointIdleTime
  - How long a ServicePoint must be idle before it is removed from the list
  - This is important because new ServicePointManager defaults will NOT be applied to existing ServicePoints!
  - Defaults are applied ONLY in the ServicePoint constructor
  - In milliseconds
  - Default is 100,000 (100 seconds)
- MaxServicePoints
  - Literally the max number of ServicePoints allowed
  - Want to limit this application to only one connection to one URI?
  - 0 is default meaning, no limit
- SecurityProtocol
  - Which protocol to use for HTTPS
    - Ssl3
    - Tls
    - Tls11
    - Tls12

ServicePointManager Options (Defaults)

- ServerCertificateValidationCallback
  - Assign a callback to use to compare server name to name in certificate
  - Dangerous if used indiscriminately
  - Could be useful for example for subdomain specific cert
    - Cert: www.contoso.com
    - URI: blog.contoso.com
- UseNagleAlgorithm
  - Nagle's Algorithm tries to reduce flood of messages by ganging small ones together
  - Great for telnet (queue up keystrokes rather than send them one at a time)
  - Not great for REST (typically one message, no need to try to reduce packets)
  - Default is true

Common ServicePoint Options

- Expect100Continue
- UseNagleAlgorithm
- ConnectionLimit
  - Max connections to this URI
- ConnectionLeaseTimeout
  - How long to wait before terminating *active* connections
- MaxIdleTime
  - How long to wait before terminating idle connections
- ReceiveBufferSize
  - Can be useful to set to larger size for large files

Note: ServicePoint settings are set via ServicePointManager defaults
or directly in code not via config file.

Configure Via Code

- **Set Defaults via ServicePointManager**
```
ServicePointManager.Expect100Continue = false;
```
- **Set individual values via ServicePoint**
```
ServicePoint uriServicePoint =
ServicePointManager.FindServicePoint(new Uri(WebApiBase));
uriServicePoint.Expect100Continue = false;
```
  - Note: May or may not be the instance you get later!
  - With archaic default of 2 connections, you have 50/50 chance
- **Set via HttpWebRequest**
webRequestClient.ServicePoint.Expect100Continue = false;
- **Set via HttpClient**
client.DefaultRequestHeaders.ExpectContinue = false;

Note:
Changes to defaults will NOT affect existing ServicePoint objects.

Those must be idle long enough to timeout and be recreated.

Config Via Config File

- Machine.config, App.Config, Web.Config

- ```
  <system.net>
    <connectionManagement>
      <add address = “http://www.sage.com” maxconnection=“15” />
      <add address = “*” maxconnection = “5” />
    </connectionManagement>
    <settings>
      <servicePointManager
        useNagleAlgorithm=“false”
        expect100Continue=“false”
      </servicePointManager>
    </settings>
  </system.net>
  ```
- Remember ServicePointManager *defaults* are set here
  - Not individual ServicePoint objects
  - Can be overridden in code
  ```

WCF Service Reference

- Generated from WSDL
- Limited options
- Still uses ServicePoints and ServicePointManager
  - You can set the defaults via ServicePointManager
  - You can theoretically get the HttpWebRequest and thereby ServicePoint
    - I have had no luck actually doing so
- You can also access the headers directly via the OperationContextScope

WCF Service Reference Example

- OperationContextScope

```
using (new OperationContextScope((IClientChannel)client.InnerChannel))
{
    WebOperationContext.Current.OutgoingRequest.Headers.Add(HttpRequestHeader.Expect,
"100-continue");
    WebOperationContext.Current.OutgoingRequest.Headers.Remove(HttpRequestHeader.Expect);
    // Your code goes here
}
```

  - Client is the instantiated client generated by the tool
  - Once you leave scope of using, context reverts to original
    - You must make your call at my “Your code goes here” comment
  - I put both an add and remove to illustrate the point… Don’t actually do that.

Fiddler: View Network Traffic, But Don’t Fool Yourself

- Fiddler lets you watch HTTP traffic
- Works by acting as a proxy server
- WARNING: WebRequest will use the proxy and ALL sites it requests from ServicePointManager will share the same ServicePoint key!
  - 127.0.0.1:8888
- Wireshark does not act as a proxy, because it intercepts packets

- 99% of the time Fiddler will be good enough
- 1% of the time you will have a mystery on your hands…
  - “Wait, I didn’t change the settings on *this* ServicePoint? Why has it changed?”

- http://docs.telerik.com/fiddler/configure-fiddler/tasks/ConfigureDotNETApp

Resources

- **.Net Source Viewer:** http://referencesource.microsoft.com/
- **Open Sourcing .Net:** http://blogs.msdn.com/b/dotnet/archive/2014/11/12/net-core-is-open-source.aspx
- **.Net Foundation:** http://www.dotnetfoundation.org/
- **OSI Model:** http://en.wikipedia.org/wiki/OSI_model
- **REST:** http://en.wikipedia.org/wiki/Representational_state_transfer
- **HTTP Visualized:** http://blog.catchpoint.com/2010/09/17/anatomyhttp/
- **Nagle's Algorithm:** http://en.wikipedia.org/wiki/Nagle%27s_algorithm
- **Fiddler:** http://www.telerik.com/fiddler
- **ServicePointManager Docs:** http://msdn.microsoft.com/en-us/library/System.Net.ServicePointManager(v=vs.110).aspx
- **ServicePoint Docs:** http://msdn.microsoft.com/en-us/library/system.net.servicepoint(v=vs.110).aspx
- **HttpClient Tutorial:** http://www.asp.net/web-api/overview/advanced/calling-a-web-api-from-a-net-client
- **My HTTP Tester console app:** https://github.com/jeffa00/httptester

My Contact Info

- jeffa00@gmail.com
- Blog: ammonsonline.com
- Twitter: jeffa00