**DPS907 notes – week 1 – Sep 7 and Sep 10**

Welcome to the web services course! This is our first session together.

**Weekly topic coverage**

For the Fall 2018 term, each “week” of topic coverage will begin with the Friday session in classroom S2154, and continue with the Monday session in the computer lab, room S2119.

**The first part of this course**

The first three weeks of the course (weeks 1, 2, and 3) will help you get started with the course content. You will transition knowledge from your web apps experience, and develop essential web services competencies.

The next three weeks (weeks 4, 5, and 6) will cover the very important design and development requirements needed to implement modern web APIs.

Then, we continue to build upon this knowledge in weeks 7, 8, 9, and 10, by introducing and then thoroughly covering security topics in web services.

We close the course (weeks 11 and 12) by covering topics that will complement your skills as a web service programmer, and enable you to compete for such opportunities in the future.

**What is a web service?**

**A web service is an application that runs on a web server, and is accessed programmatically.**

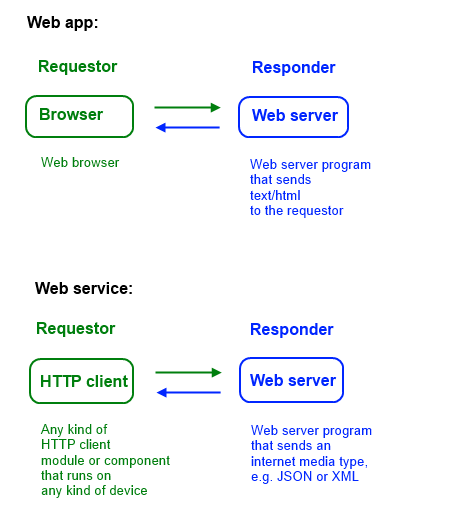
When we parse this short sentence, and consider the meanings behind the simple words, we reveal some very important concepts:

* HTTP is the protocol, enabling wide use and scalability
* Humans don’t use a web service directly – instead, the application they are using creates and sends a request to the web service, and handles the response in a way that’s meaningful to the application
* A web service’s functionality is discoverable, and typically known as an application programming interface (API)

Web services can be developed on any web-connected technology platform, in any language, and can be used on any kind of device.

**What’s the difference between a web app, and a web service?**

Study this diagram to understand the differences, and then be prepared to explain them to someone else:



**Give me a brief history lesson on web services**

With the rise of the web’s use and popularity in the 1990s, efforts were made to define and specify web services.

This led to the [*de facto*](http://www.merriam-webster.com/dictionary/de+facto) standardization of [SOAP XML web services](https://en.wikipedia.org/wiki/SOAP). Often described as “big web services”, SOAP XML web services are the implementation of [remote procedure calls](https://en.wikipedia.org/wiki/Remote_procedure_call) on the web. This kind of web service typically has one specific endpoint address, and requestors must create and send a data package (which conforms to a data format named SOAP), and then process the response (which also conforms to SOAP).

However, other efforts took advantage of the web and its existing features and benefits. In other words, they simply followed the HTTP specification and its [*ex post facto*](http://www.merriam-webster.com/dictionary/ex%20post%20facto) architecture definition, to create true and pure web services. These kinds of web services, often termed “Web API”, or “HTTP services”, exploded in use and popularity from about 2005 onwards, and are now the preferred design approach.

In this course, you will learn both (for maximum employment value). However, most of our work will use the modern web services.

Web services can be developed on any web-connected technology platform, and in any language. Web services are interoperable.

**Are web services important?**

Yes.

Absolutely.

Web services are *vital* for modern software architectures.

For *all* device platforms.

**Show me a web service**

We will use Visual Studio 2017 Enterprise in this course to create web services.

Your professor will briefly introduce the tooling, the “Web API” project type, and the process of creating a web service.

Fiddler will be used to interact with the web service. Your professor will briefly introduce you to Fiddler, and a later section will provide more information.

**How do I start learning web services?**

You must learn (more) about HTTP.

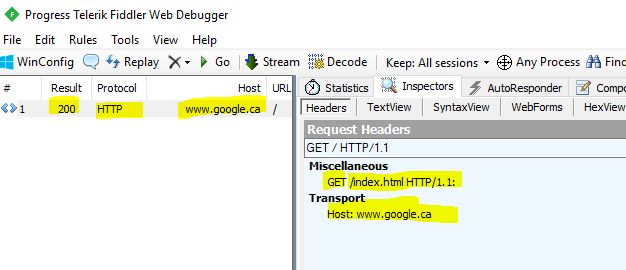
In class, we will discuss some HTTP topics, and refer to the [overview on Wikipedia](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol), and the official specification series, [RFC 7230](http://tools.ietf.org/html/rfc7230) to RFC 7235, which replaced RFC 2616 in June 2014.

**Request, response**

HTTP is a typical Internet protocol that relies on an exchange of messages between a requestor and a responder. The messages are plain text and must conform to a specific format.

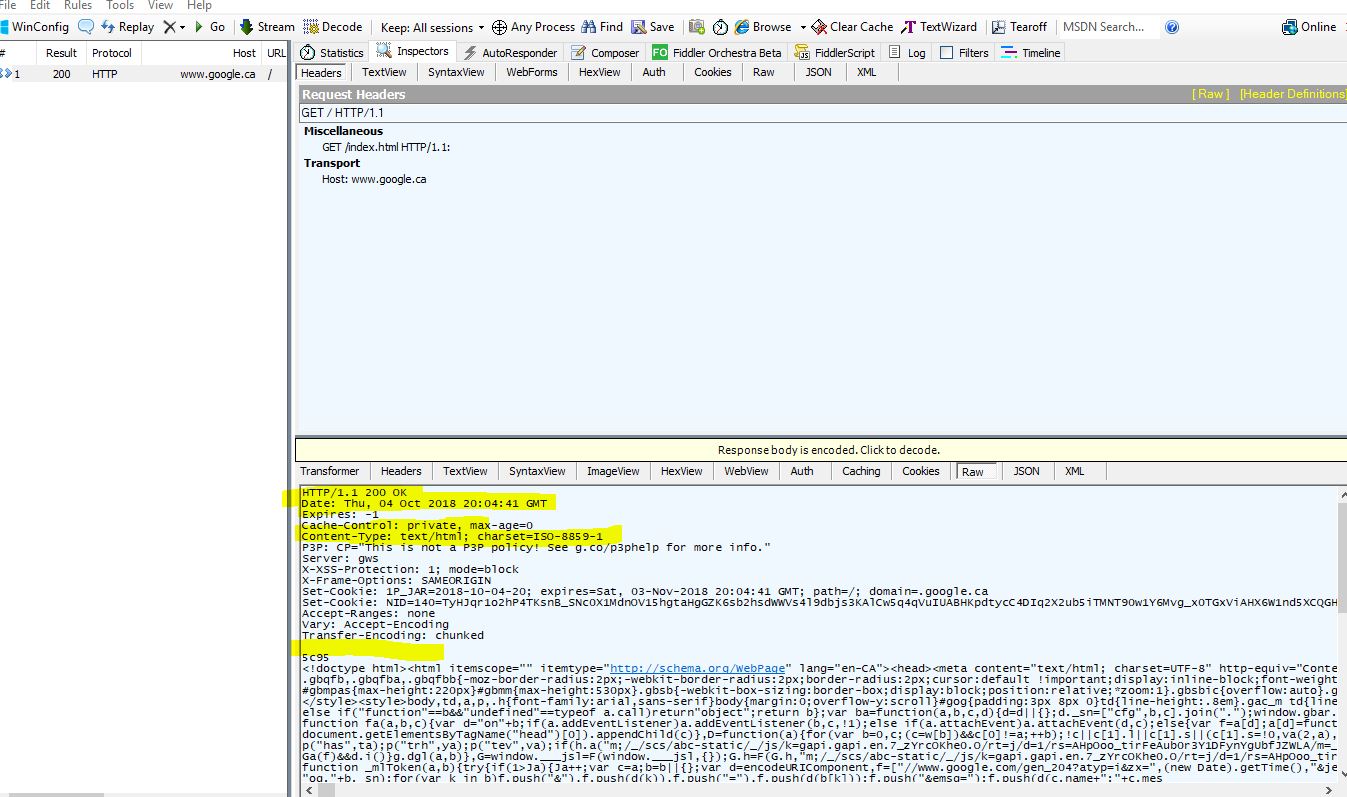
An example of a request message is shown below. The resource at “/index.html” is being requested:

|  |  |
| --- | --- |
| 1  2 | GET /index.html HTTP/1.1  Host: www.google.ca |

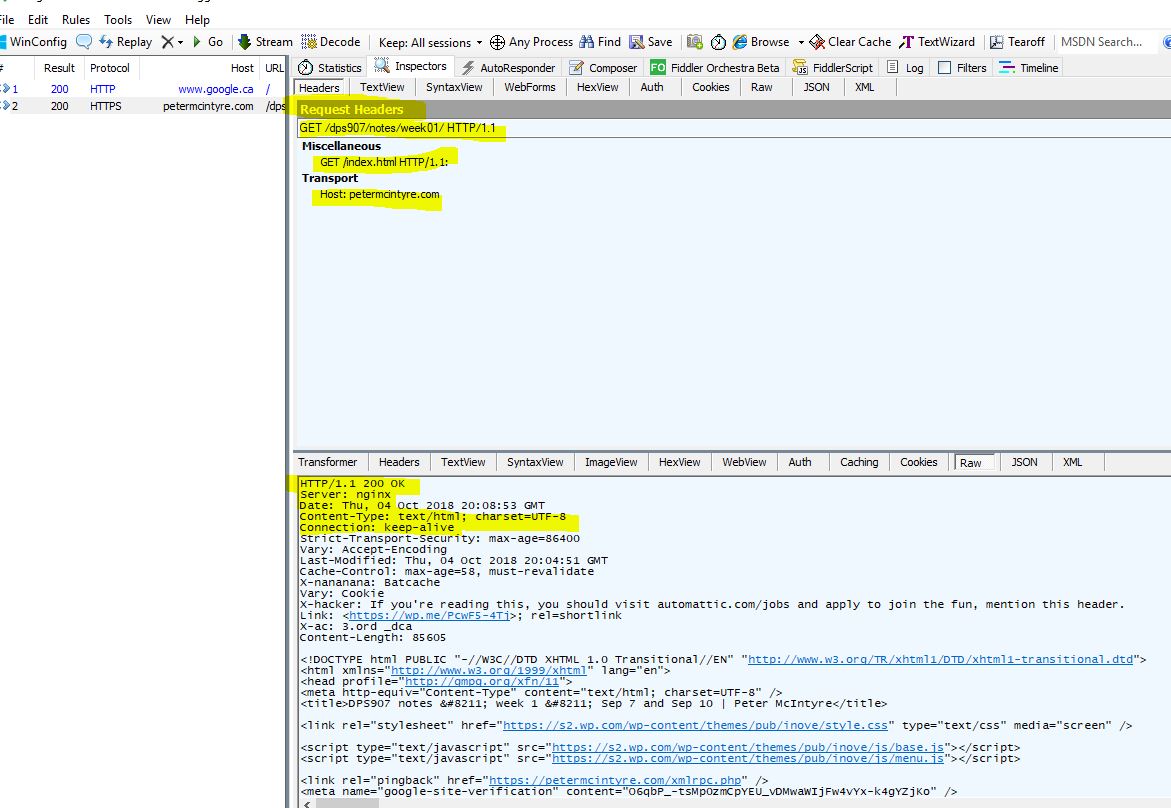


An example of the response message is shown below. Some data has been removed to simplify the example. Notice the empty line in between the headers and the message body:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | HTTP/1.1 200 OK  Date: Sun, 03 Sep 2017 16:29:52 GMT  Content-Type: text/html; charset=ISO-8859-1    <!doctype html>  <html itemscope="" itemtype="<http://schema.org/WebPage>">  <head>  <meta itemprop="image" content="/images/google\_favicon\_128.png">  <title>Google</title>  <script>  (function(){ |



How can I see these requests and responses? Use Fiddler.



**HTTP requests used in this course**

HTTP supports a number of request methods. We will use (at least) the following in this course:

* GET – read/fetch a resource
* POST – add to a resource (the resource is typically a collection)
* PUT – modify an existing resource
* DELETE – delete an existing resource
* OPTIONS – ask for a list of request methods that are supported by a resource
* HEAD – ask for the response headers for a resource

In addition to learning more about HTTP, it’s important to know, understand, and use the right terminology or vocabulary. We introduce terms like *resource*, *representation*, and *internet media types below*.

**OK… what’s a resource?**

A *resource* is a digital asset.

Familiar examples include a document, or an image.

How do you identify a resource? By using its URI (uniform resource identifier). The URI standard is described in [RFC 3986](http://tools.ietf.org/html/rfc3986), and also in [a Wikipedia article](http://en.wikipedia.org/wiki/Uniform_resource_identifier).

What is the format, or representation, of the resource? Well, it depends on the design of the web service, and sometimes the needs of the requestor.

**OK… what’s a representation?**

As defined above, a resource is a digital asset.

A *representation* is a digital asset that is formatted as a specific internet media type.

Think about a scenario where a web service was used to manage students in a course. Each student is a resource – a digital asset – that can be identified by a URI.

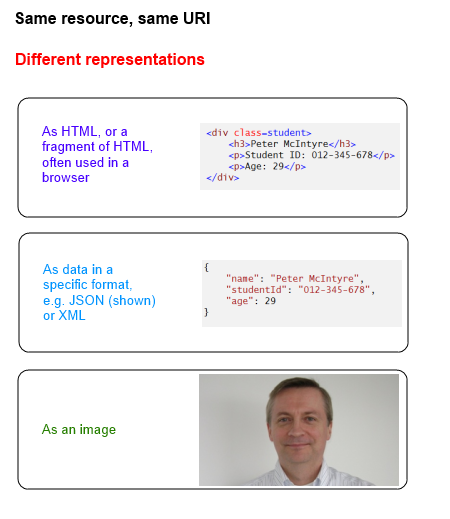
If a user requested a specific resource through a web browser, you would expect that the resource would be *represented* by some HTML that included the student’s name, student ID, and so on.

Alternatively, it’s also possible to request the same specific resource – using the same URI – but also specify that it be returned in a data format (like JSON or XML, discussed later). The server will return a data *representation* of the resource.

Or, maybe the request specified that the student’s photo be returned as the resource’s *representation*. Again, the same URI is used.

So, in summary, a resource’s *representation* can vary to meet the needs of the web service programmer or the web service user. The requestor and responder use a feature called [content negotiation](https://en.wikipedia.org/wiki/Content_negotiation) to make this happen.

Every representation is defined by an internet media type.



**Remind me about internet media types**

Your professor strongly recommends that you review the [notes about media types](https://petermcintyre.com/bti420/notes/mar22/) on the web apps course web site. *Do that now*. Read the main section titled “Review the concepts, requesting photos/images from a browser”.

Welcome back.

An *internet media type* is defined as a *data format* for a *representation* of a *resource* on the internet.

The data formats are standardized, published, and well-known, by the [IANA](http://www.iana.org/assignments/media-types) (the Internet Assigned Numbers Authority).

[This Wikipedia article](http://en.wikipedia.org/wiki/Internet_media_type) is an acceptable introduction to internet media types.

For web service programmers, two important internet media types are used as data formats, JSON and XML.

Both are plain-text data formats. They are somewhat human-readable.

Later, you will learn how to work with non-text media types. Now, we will work with plain-text JSON.

**Get started with JSON**

JSON is an initialism for JavaScript Object Notation.

It is a lightweight data-interchange format. It is language-independent, however it uses conventions that were first suggested by the JavaScript object literal or initializer. Many other languages (including C#) now use a similar approach when creating objects. JSON has become a good and comfortable way to describe data for interchange purposes.

Here’s an [overview of JSON](http://en.wikipedia.org/wiki/Json) from Wikipedia.

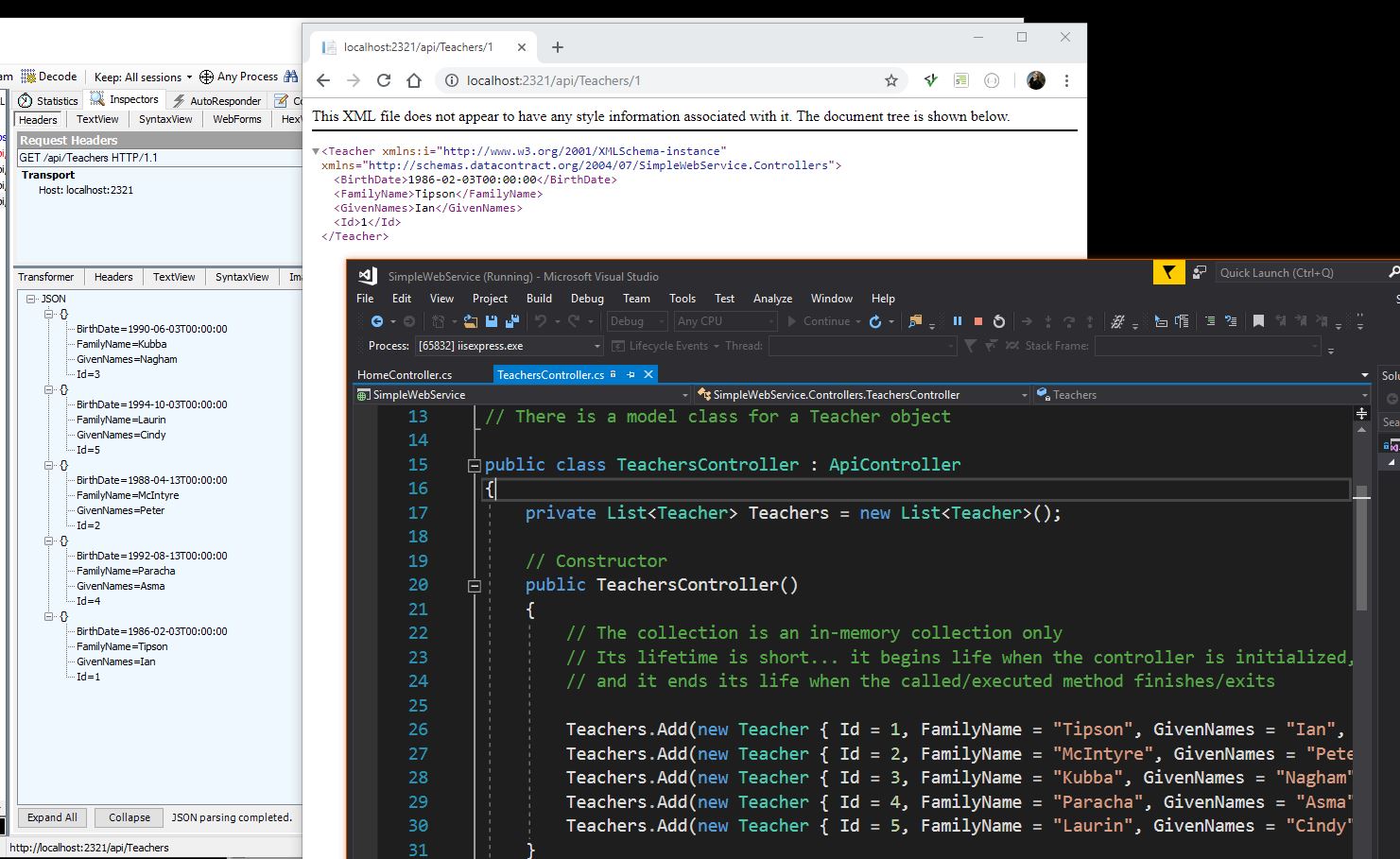
Here’s the [official web site for JSON](http://json.org/), by Douglas Crockford.

Although JSON is historically derived from JavaScript object literals, there are a few notable differences to programmers who are new to JSON:

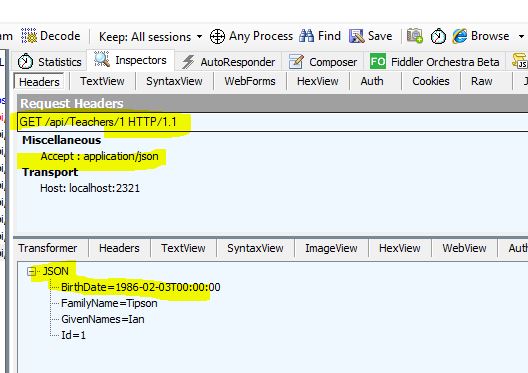
> In JSON, each property name is surrounded by double-quotes. That’s not the situation in JavaScript.

> In JSON, there is no Date type. Dates are expressed as strings, almost always in [ISO 8601](https://en.wikipedia.org/wiki/ISO_8601)format. In contrast, JavaScript does have a Date type.

The web services that you create in this course will rely on the JSON internet media type.



**Request for json**



**Get started with XML**

XML is an initialism for eXtensible Markup Language.

Students in this course will recognize XML and recall their past experience with applications of XML, possibly in xHTML, and definitely in the Web.config settings in ASP.NET.

In general, XML uses programmer-defined element names to define the data. There are several popular XML document formats that have been created over the years, which can ease the burden of designing a “[schema](https://en.wikipedia.org/wiki/XML_schema)“.

When working with XML, these are the most important rules:

1. Must be well-formed
2. Must be based on a schema

Well-formed rules:

* The (optional) first line is the XML declaration:

 <?xml version="1.0" encoding="UTF-8" ?>

* The second line is a single ‘root’ element to contain all other elements; the root element will likely include a reference to a schema
* Elements must have a start tag and an end tag, and are case-sensitive
* Elements must be properly nested, and cannot overlap

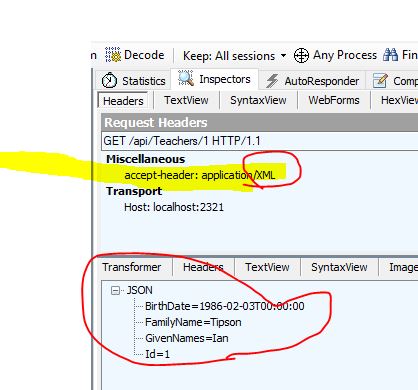
Schema rules:

* The XML documents we use later in this course typically use the XML Schema standard
* However, you may encounter examples that use the RSS or Atom formats

Here’s an [overview of XML](http://en.wikipedia.org/wiki/Xml) from Wikipedia.

 Not accept-header: application/xml

Accept : application/xml

****

**Return to the tooling – create and test a web service**

Your professor will use Visual Studio 2017 to create a simple web service again.

Then, Fiddler will be used to test (and interact) with the web service.

**Introducing ASP.NET Web API**

Microsoft describes “ASP.NET Web API [as] a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework.”

The framework includes classes, ready-to-use components, and a server runtime. Overall, the framework makes you very productive.

Web API has a well-defined processing model – known as the request-handling pipeline – for handling requests and generating responses. The pipeline is also configurable, and extensible.

**Web API projects in Visual Studio**

Every student is familiar with the way to create an ASP.NET MVC web app project in Visual Studio 2015.

Creating a Web API project is done in a similar manner. Select the “Web API” project type in the dialog. For best results, in the first few projects you create, configure the project with no authentication.

In the Controllers folder, notice the “Values” controller. It inherits from a type ApiController.

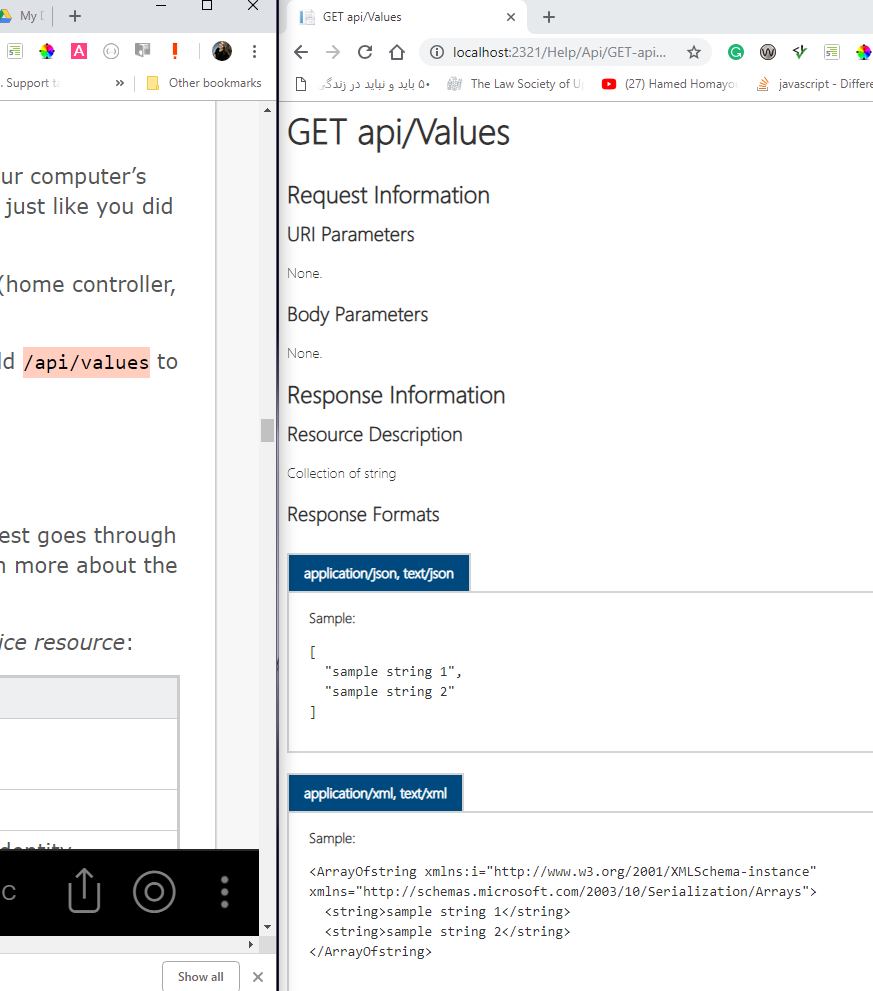
Please notice that the Web API project also includes the web app components. Notice the “Home” controller. Therefore, a Web API project can also have user-visible and user-browsable web pages. Very convenient.

**Interacting with your project**

Similar to web app projects, you can load and run your Web API project on your computer’s on-demand web server. Without debugging, the keyboard shortcut is Ctrl+F5, just like you did with web app projects.

A browser will open, and show the project’s home page, which is a web page (home controller, index view).

Copy the URL, and then use it as the base URL when you are using Fiddler. Add /api/values to the URL to fetch content.



**The request-processing pipeline**

In your ASP.NET MVC web apps programming course, you learned that a request goes through a processing “pipeline”, as the response is being prepared. You will learn much more about the pipeline in this course, but you should know at least the following facts now.

Compare the pipelines for requests to a *web page resource* and to a *web service resource*:

|  |  |  |
| --- | --- | --- |
|  | **Web app** | **Web service** |
| Incoming request | Route inspection and matching | Same |
|  | Locate and initialize the controller | Same |
|  | Perform authorization, by inspecting the incoming identity package (a cookie) | Similar, but the identity package is a token |
|  | Pass incoming data to the method, do model binding | Same |
|  | Execute method, which typically passes data (object or collection) to a view engine, which uses a view object to render HTML | Similar, but the data is passed to a built-in *media type formatter*, which renders JSON or XML |
| Outgoing response | Delivered to the requestor | Same |

Do you notice the differences?

* The passed-in identity package is in a different format
* The method execution result is handled by different modules in the pipeline

**Get started with Fiddler**

Fiddler is an HTTP inspector.

We will use Fiddler to interact with a web service.

It is installed on the College computer-lab computers, and is also available from the [Fiddler web site](http://www.telerik.com/fiddler).

Why Fiddler? This type of app enables us to focus on web service creation – which is a server-focused development effort – without the burden to create a client/requestor app that interacts with your web service. Later in the course, we will cover some client/requestor app creation topics, but plan to learn and use Fiddler. It’s your friend.

**Important features**

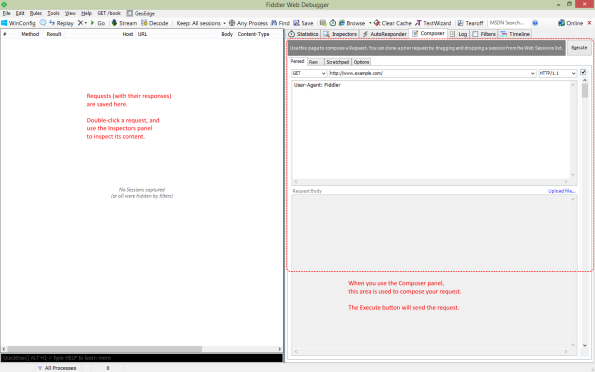
There are three important features in Fiddler, for beginners:

1. Composer – enables you to compose and send an HTTP request
2. List of requests – saves all of the requests from the current session
3. Inspector – enables you to inspect the contents of the request and its response

The following shows the Fiddler feature layout. (Click to open it full-size in its own tab/window.)

When Fiddler loads, ensure that it is not “capturing” traffic from other open browser windows on your computer. Look in the lower-left corner; if it says “Capturing”, then press F12 to stop capturing.

Can you save this setting? Yes. Open Tools > Fiddler Options. On the Connections tab, un-check (clear) the checkbox named “Act as system proxy on startup”.

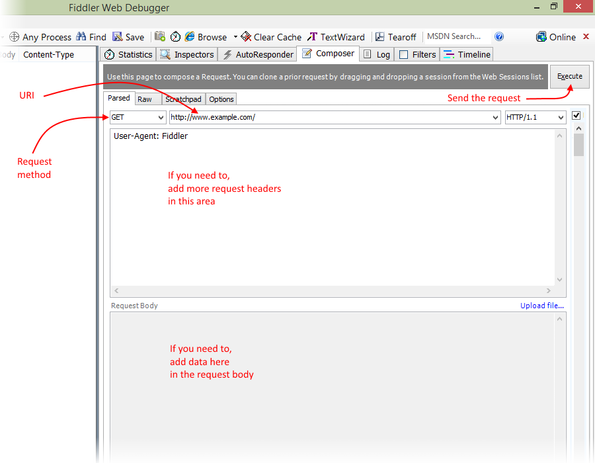
[](https://petermcintyre.files.wordpress.com/2015/09/fiddler-intro.png)

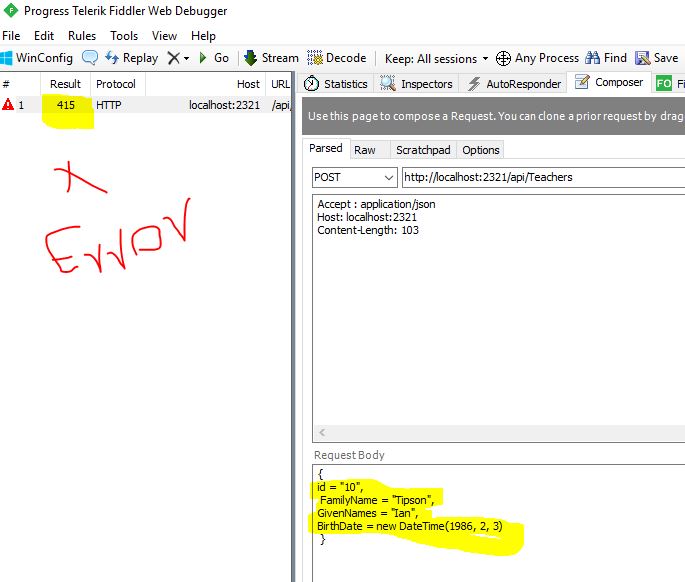
**How to compose and send a request**

Carefully select and edit the following:

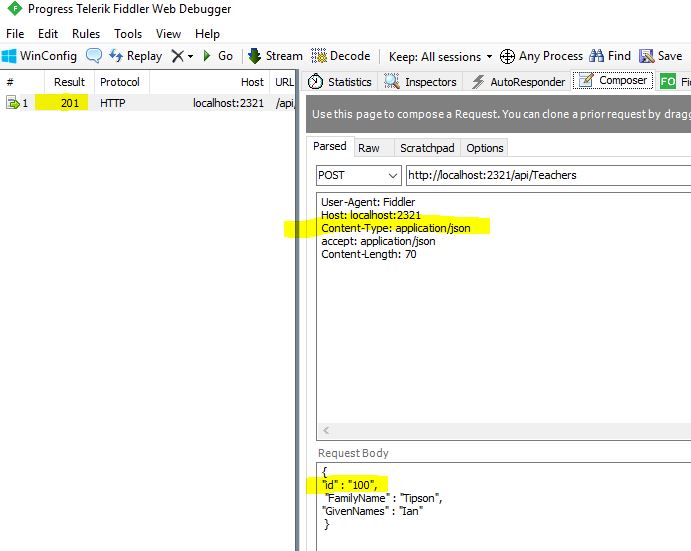
1. Request method – choose GET, POST, PUT, DELETE, OPTIONS, etc.
2. URI – enter the URI of the resource
3. Request headers – if needed, you must add request headers (e.g. Accept, Content-Type, Authorization etc.)
4. Request body – if needed (e.g. in POST and PUT requests), add data that matches the Content-Type specified in the request header; use the “Upload file…” link if you have byte-oriented content to add (e.g. an image, etc.)

After the request has been composed, Execute to send the request. The result will appear in the left-side list of requests. Be patient, because some requests can take several seconds to complete.

[](https://petermcintyre.files.wordpress.com/2015/09/fiddler-how-to.png)

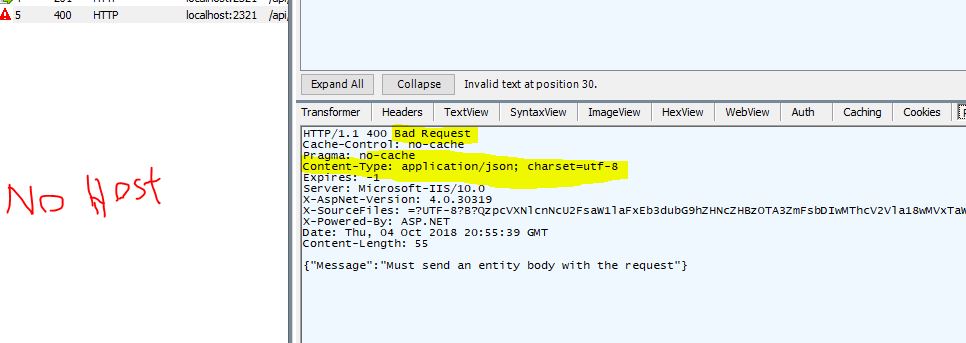


Needed “ Content-Type : application/json “

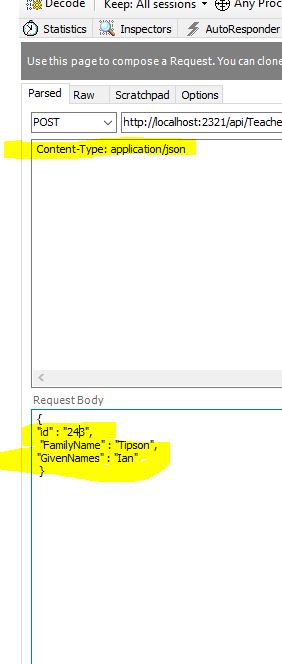


Bad request – because no “ Host : … ” is mentioned

Sorry, the request doesn’t care about the “ Host : “ much since it has the host in the Post URL anyway. This request simply failed because I missed a “” quotation mark.



To create a successful Post request, we MUST use “ Content-Type “



**Interact with the Apple iTunes Store “search” web service**

Use Fiddler to call the Apple iTunes Store “search” web service, which delivers a JSON document. An example URI, for all items from [Yo-Yo Ma](https://en.wikipedia.org/wiki/Yo-Yo_Ma) that include the word “[oboe](https://en.wikipedia.org/wiki/Oboe)“, is:

<https://itunes.apple.com/search?media=music&term=yo-yo+ma+oboe>

Use Fiddler’s features to inspect and carefully study the request and the response.

(End of Friday content; Monday content is next.)

**Typical agenda for our Monday classes**

On your timetable, this Web Services course is a four-hour course. The Monday session, in a computer-lab room, is part of the course. It is not optional, and you cannot skip it. The bottom line is that you’re paying for a four-hour course, and that’s what you’ll get from me. *I also expect that from you.*

During our Monday classes, we will typically have a mix of discussion/lecture, and hands-on activity. The hands-on activity may be guided, and help you learn the application of a concept or technique. It will also typically include some time where you work on the current assignment. There may be a brief “work report” that you will submit before you leave the room; it will be graded, and become part of your assignment grade.

Also, the Monday classes give you an ideal opportunity to get help from your professor. I can help then, or during my “student help time” in my office. I am not available late in the evening or on weekends to help you in person or by email, so take advantage of the scheduled class times and my drop-in student help time office hours.

**Agenda for the Monday class**

Here’s the agenda for the Monday class:

* Your professor will guide you in the creation of a simple web service
* Fiddler will be demonstrated, used, and learned
* Some work on the current assignment

**Code examples**

Code examples are published on a [public GitHub repository](https://github.com/peteratseneca/dps907fall2018).

*GitHub is a hosting service for source code. It offers a web interface to enable easy browsing. It also offers a set of tools and capabilities that enable software developers to publish code, collaborate with others, and improve their productivity and effectiveness.*

On this web site, a link to the repository is on the right side of this page, near the top. (For best results, open the link in a new tab or window.)

Beginners can use a web browser to view the source code. You can also click the “Download ZIP” button to get a full and complete copy of all code examples in the repository.

For this course, the code examples will be published weekly, in logically-named folders (e.g. “Week\_01”, “Week\_02”, etc.).

During the course’s lifetime, about twenty or more code examples will be published.

As you have observed, each ASP.NET web app or web service project includes a sizable “packages” folder. In the GitHub code repository, each distinct code example does not include the “packages” folder, as a time- and space-saving measure. Each “packages” folder is about 50MB in size, so twenty code examples would require about 1GB storage. The contents of the “packages” folder does not compress much.

Therefore, if you have downloaded and opened a project in Visual Studio, you *must* do a build/compile *before* you begin working with it. The build/compile action will fetch the packages it needs from the web.

All code examples are designed to be usable immediately and error-free.  
Each should compile and execute correctly.

If you have any problems with a code example, please contact your professor.

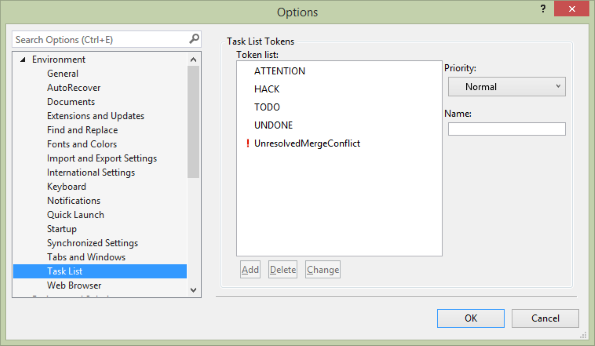
**Reminder about the code comments in the code examples**

It is possible that you know about this from your experience in the web apps course. If not, read on…

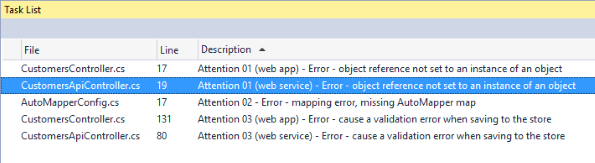
Almost all current and future code examples will include “[comment tokens](https://msdn.microsoft.com/en-us/library/txtwdysk.aspx#tokensComments)“.

As you will learn in the linked article, Visual Studio has a built-in token named “TODO”. In *your*Visual Studio, you will add a custom token named “ATTENTION”. This is a one-time operation on your own computer.

Open Tools > Options. Navigate to the Environment > Task List panel. Enter “ATTENTION” into the “Name:” textbox, and click the “Add” button. It will add it to the “Token list”, as shown below.



Then, when you show your **Task List** (from the View menu in Visual Studio), it will show you a nice list of tokenized comments. Arrange, resize, and sort the columns as desired (by click+drag in the headers):



Each item on the list can be double-clicked. When you do that, the source code file will open, and it will scroll to the position of the comment. Enjoy.

**Code examples for today (this week)**

Simple Web Service

AutoMapper In Web API

**Creating a web service**

We will use Visual Studio 2017 Enterprise (VS) in this course to create web services.

VS runs on Windows, and creates ASP.NET web apps that run on Windows.

If you have a dogmatic view that demands you to use macOS or Linux or something other than VS on Windows, you should probably drop the course. If you are thinking you can do this course without Windows, then good luck. Your professor uses Windows to deliver the course, and to grade your work. If your submissions don’t work and can’t be reviewed on a Windows system, then no grades will be earned.

Remember, in your academic program, we deliberately use different operating systems, development tools, and languages, in an effort to promote learning and enable you to become a more experienced graduate.

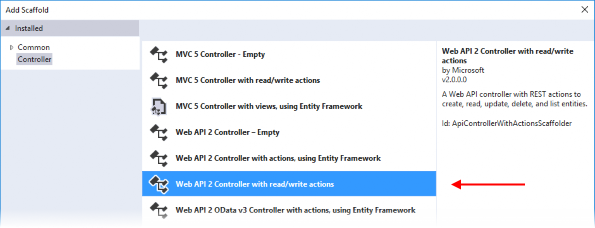
Your professor will introduce the tooling, the “Web API” project type, and the process of creating a web service.

Fiddler will be used to interact with the web service.

**Controller class for a web service**

In Visual Studio, when you add a new controller to the Controllers folder, choose this template in the “Add Scaffold” dialog:

**Web API 2 Controller with read/write actions**



The controller source code will be generated, with method stubs for the typical actions:

* Get all
* Get one by identifier
* Add new
* Change existing
* Delete item

We will edit the method signatures, and the code in the methods, to meet our needs.

Notice that the controller class inherits from [ApiController](https://msdn.microsoft.com/en-us/library/system.web.http.apicontroller(v=vs.118).aspx" \t "_blank). This base class is similar in many ways to the Controller base class that you have in ASP.NET MVC web apps.

However, it was created by a different team, and is tuned to the needs of web services. While we can coax the web apps Controller class into use as a web service, it is missing some qualities that we want in our web services.

**Return type of a controller method**

As you begin this Web Services course, you must learn about method *return types*.

In the next class notes, you will learn about the request-processing pipeline. For now, you should know that a request is handled by a number of distinct code modules or components, including the controller that you are writing. The result of this computational “pipeline” is that a response is generated and returned to the requestor. The response generator is known as a *media type formatter* – included in the Web API framework – which renders, or serializes, the data result to JSON or XML.

Whether your method return type is an object, or a collection of objects, the serializer does its job, and the user receives valid data in the response.

In *all* of your work, until further notice, your methods will use *only two* return types:

1. IHttpActionResult
2. void

The first return type – *IHttpActionResult* – will be used for all methods that return data. This return type will be used on almost all of your methods.

The second return type – *void* – will be used for methods that do not return data.

You may notice that the Visual Studio code-generator – which can create method stubs for new controllers – does not always use these two return types. In those situations, you *must* change the return types to IHttpActionResult or void, or create new methods that replace the code-generated method stubs. *Do not assume that the code generator is correct* for your situations and use cases.

Why do we care about this? Well, when you use IHttpActionResult or void as the return type, then the request-processing pipeline adds the correct HTTP status code to the response. Nice.

Note:

Today, the professor noted the importance of the RFC7230 series that describes the HTTP protocol was highlighted. The series is also prominently featured in the Resources area of this web site.

[RFC7231](https://tools.ietf.org/html/rfc7231) will be the most-used document in this course. It has all the answers you’ll need when writing HTTP services.

The HTTP response status codes are discussed in [Section 6 of RFC7231](https://tools.ietf.org/html/rfc7231#section-6).

When an ApiController method returns an IHttpActionResult, we use a specific *helper method*, depending upon the method logic. Most of the helper methods accept one or more arguments, which is how you actually return an object or collection of objects. For example, assume that the method fetched a specific object named “result”. We would use the Ok() helper method:

|  |  |
| --- | --- |
| 1 | return Ok(result); |

Here is an alphabetically-ordered list of helper methods:

BadRequest (returns an HTTP 400 status code)

Conflict (409)

Created (201)

InternalServerError (500)

Json (200)

NotFound (404)

Ok (200)

Redirect (302)

StatusCode (choose from an [enumeration of status codes](https://msdn.microsoft.com/en-us/library/system.net.httpstatuscode(v=vs.110).aspx))

Unauthorized (401)

Here is the same list, ordered by HTTP status code:

Ok (200)

Json (200)

Created (201)

(remember – a *void* method return type will deliver HTTP 204)

Redirect (302)

BadRequest (400)

Unauthorized (401)

NotFound (404)

Conflict (409)

InternalServerError (500)

…and… StatusCode, from which you can choose from an [enumeration](https://msdn.microsoft.com/en-us/library/system.net.httpstatuscode(v=vs.110).aspx)

If you’re interested, the origin of these methods trace back to the [System.Web.Http.Results namespace](https://msdn.microsoft.com/en-us/library/system.web.http.results(v=vs.118).aspx" \t "_blank).

**Fiddler testing patterns**

As you begin this Web Services course, use the following testing patterns when using the Fiddler tool:

*Get all* request

* Method is GET
* URI is the collection URI
* Optionally, you can specify an Accept header

*Get one* request

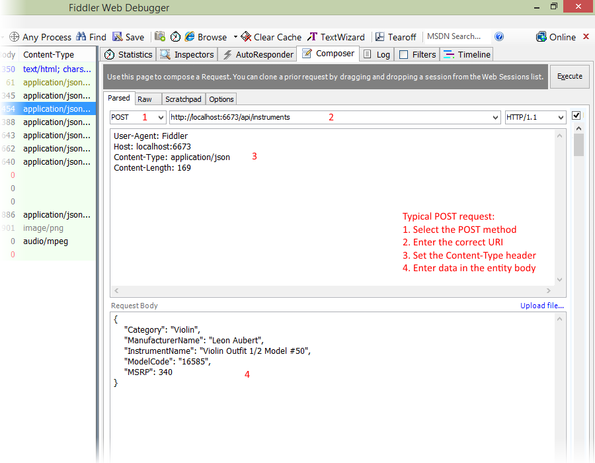
* Method is GET
* URI includes a segment that has an identifier
* Optionally, you can specify an Accept header

*Add new* request

* Method is POST
* URI is the collection URI
* Must add a Content-Type header
* Include a correctly-defined-and-formatted entity body

Usage tip: Execute a “get one” request, and use the response body as a template for entering a new entity body for the “add new” request. Copy, paste, and edit. It will save you a bit of typing.

The following is an example of a correctly-formatted “add new” request. Click to open full-size in its own tab/window.

[](https://petermcintyre.files.wordpress.com/2015/09/fiddler-how-to-post.png)

**Work on the assignment**

[Assignment 1](https://petermcintyre.com/dps907/graded-work/assign1/) is now available. It is due near the end of the day on Thursday, September 13.