Harvesting the Web: Automated Datagathering with PowerShell

Presented by Mark Minasi
mark@minasi.com
twitter @mminasi
Newsletter at www.minasi.com

So, I live near the water sometimes…

- … and I have been a real body surfing fan since I was four years old
- But now I'm a bit pickier about when I go into the water, and know when the conditions are best:
  - Water temperature over 74 degrees F
  - Within 90 minutes of low tide either way
  - Plus or minus two or three days around full or new Moons
- And clearly I'd love to enlist PowerShell in my data quest

Before PowerShell, I Would…

- Fire up a web browser
- Find the water temperature near the "Duck research pier" in Duck, NC
- Phase of the moon can either be estimated
  - (Time between full moons is 29.5 days)
  - There are phase-of-moon sites
- Low tides can be found on many sites as well
- I'd like to collect that all in one place, maybe a web page
- If only I could automate that…

PowerShell's Tools

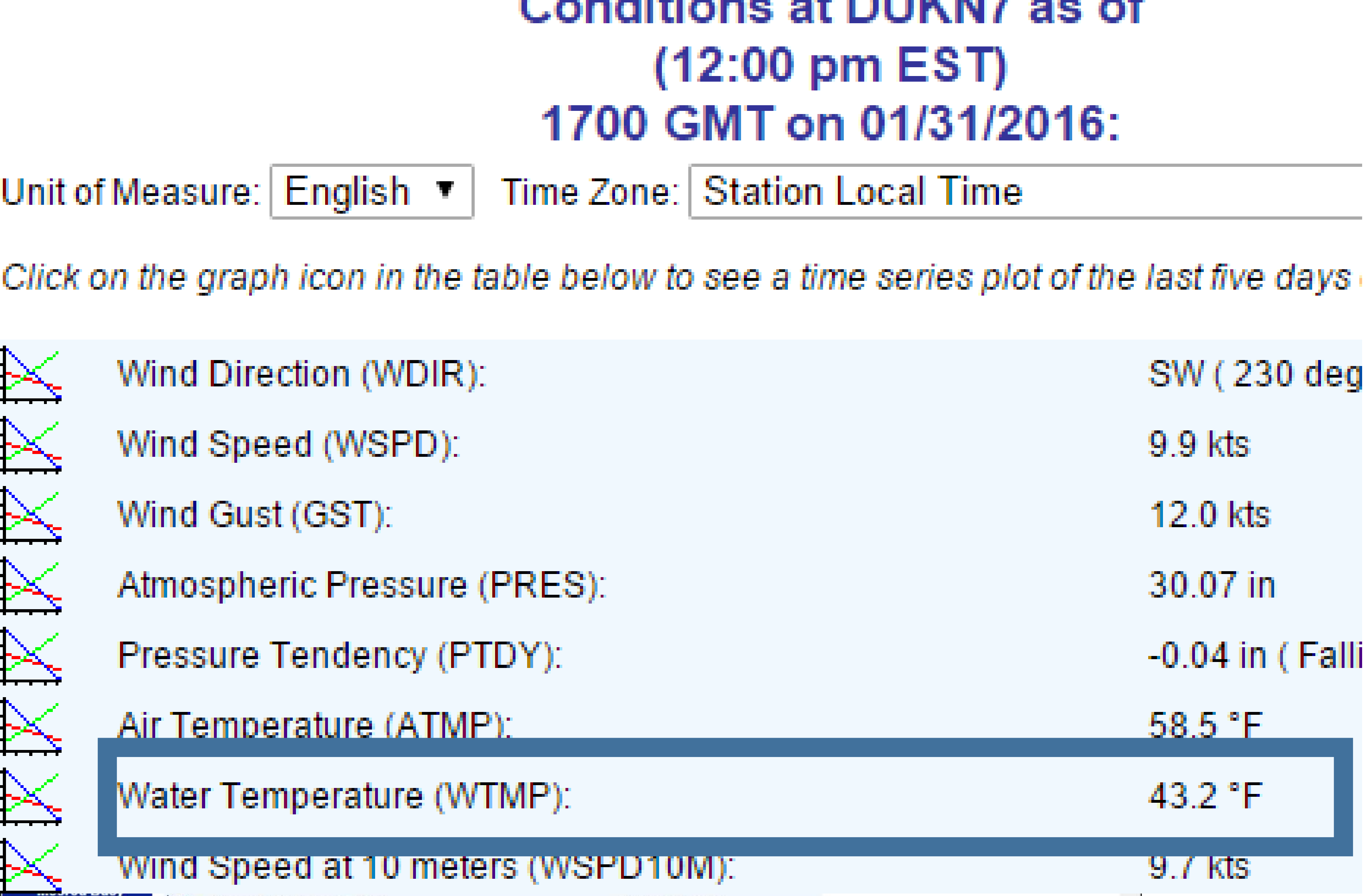
- Two cmdlets are essentially web browsers
  - Invoke-WebRequest ("IWR")
  - Invoke-RESTMethod ("IRM")
- One is a pathway to "SOAP" web services
  - New-WebServiceProxy
- All Powershell 3.0 and later

Four Approaches to Collecting Data

- Screen scraping
- Tell PowerShell to download a file, like a photo
- Query SOAP-type web services
- Query RESTful services
- And a sort of "fifth" – filling in forms

Where Can We Find Duck's Ocean Temperature?

- My go-to location is NOAA's array of sensor buoys
- Duck's site ID has an ID value of "DUKN7"
- It's a research pier with several instrument packages
- Google DUKN7, or browse to
- www.ndbc.noaa.gov/station_page.php?station=dukn7

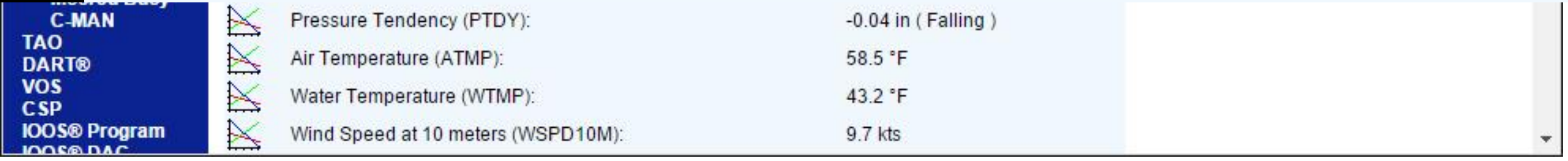


Conditions at DUKN7 as of
(12:00 pm EST)
1700 GMT on 01/31/2016:

Unit of Measure: English ▼ Time Zone: Station Local Time

Click on the graph icon in the table below to see a time series plot of the last five days

	Wind Direction (WDIR):	SW (230 deg
	Wind Speed (WSPD):	9.9 kts
	Wind Gust (GST):	12.0 kts
	Atmospheric Pressure (PRES):	30.07 in
	Pressure Tendency (PTDY):	-0.04 in (Falli
	Air Temperature (ATMP):	58.5 °F
	Water Temperature (WTMP):	43.2 °F
	Wind Speed at 10 meters (WSPD10M):	9.7 kts



C-MAN
TAO
DART®
VOS
CSP
IOOS® Program

	Pressure Tendency (PTDY):	-0.04 in (Falling)
	Air Temperature (ATMP):	58.5 °F
	Water Temperature (WTMP):	43.2 °F
	Wind Speed at 10 meters (WSPD10M):	9.7 kts

So Our First Goal…

- Use PowerShell to simulate pointing a browser at the NOAA site
- Save its (ugly) raw HTML to a "variable," something that lets us temporarily store data; this will be "string" data
- Use a "regular expression capture" to find and extract just the digits of the water temperature by building a "regex pattern"
- The capture will end up with just the digits of the water temperature

PowerShell's Browser: Invoke-WebRequest

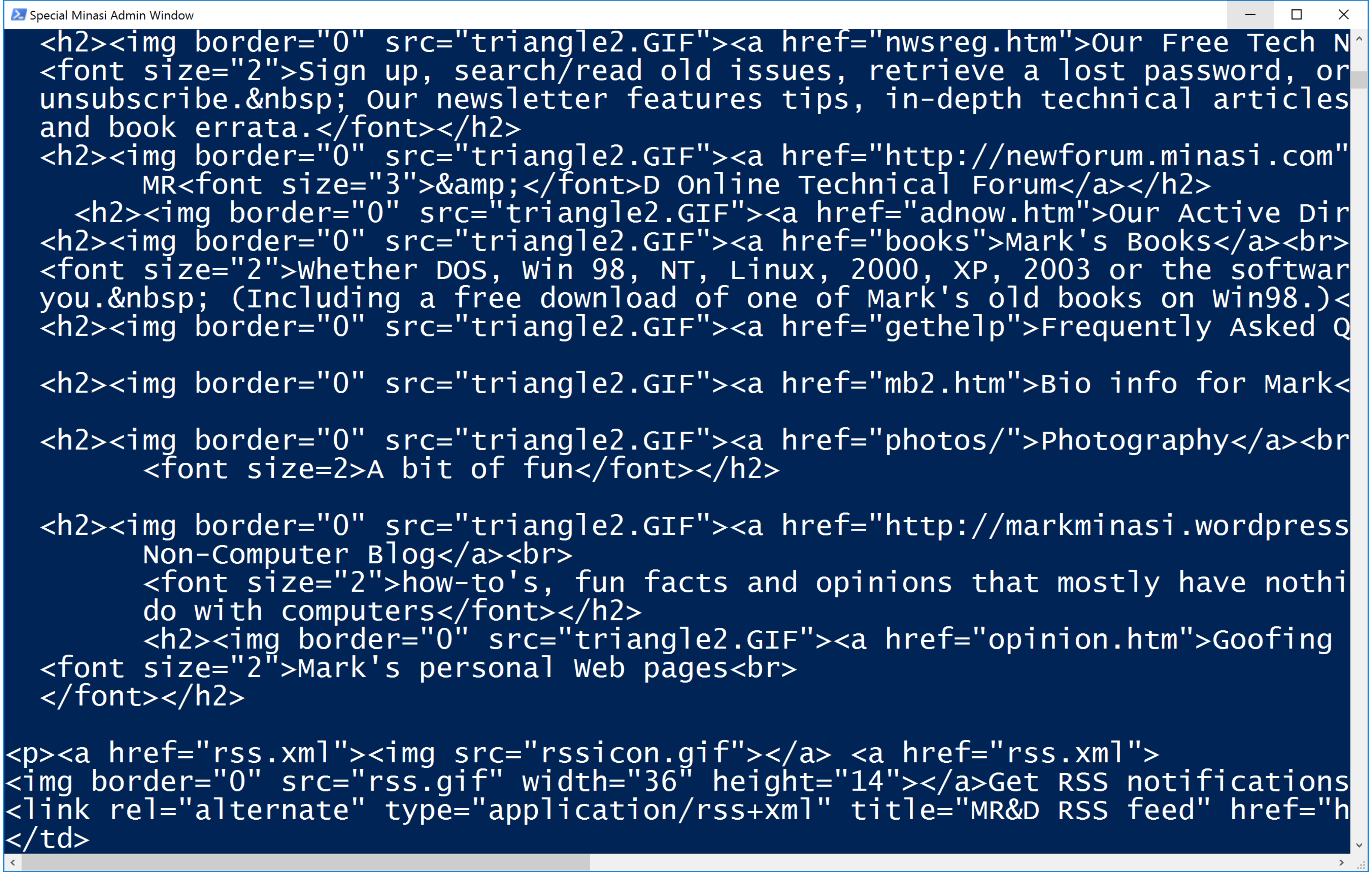
- We'll use Invoke-WebRequest, alias IWR… example:
- **Invoke-WebRequest 'http://www.minasi.com'**
- The result is a bit of a text mess, so we'll want to parse it
- Normally we retrieve the command's output to a PowerShell "variable," which is just a name that starts with a dollar sign
- **$R = Invoke-WebRequest 'http://www.minasi.com'**
- You could just see the output of the command by typing
- **$R**

What's in There?

- With PowerShell, data is structured with a "document model," and has parts and attributes
- $R | Get-Member shows what's in there, like
  - **$R.StatusCode** returns things like 200 (success), 404 etc
  - **$R.headers** dumps headers
  - **$R.Scripts** shows scripts, .forms shows forms, there is .rawcontent, .content, .images and more
  - **$R.Content** basically looks like doing a "view source" on a web page

Next Step: Analyze the Output

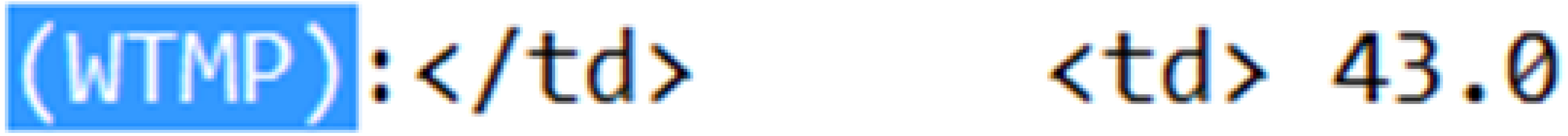
- Get the pier's home page:
- **$R = IWR 'www.ndbc.noaa.gov/station_page.php?station=dukn7'**
- We're going to want to take a close look at the raw HTML text that the site returns so we can give PowerShell a pattern to use when extracting the temperature
- That HTML is in $R.content… type it once and you'll see:
- **$R.content**
- That's a mess, so put it in the clipboard:
- **$R.content | Set-Clipboard**
- Open Notepad and paste it in so we can look at it and figure out how to pull out the temperature number / numbers


Special Minasi Admin Window

```
  <h2><img border="0" src="triangle2.GIF"><a href="nwsreg.htm">Our Free Tech N
  <font size="2">Sign up, search/read old issues, retrieve a lost password, or
  unsubscribe.&nbsp; Our newsletter features tips, in-depth technical articles
  and book errata.</font></h2>
  <h2><img border="0" src="triangle2.GIF"><a href="http://newforum.minasi.com"
        MR<font size="3">&amp;</font>D Online Technical Forum</a></h2>
    <h2><img border="0" src="triangle2.GIF"><a href="adnow.htm">Our Active Dir
  <h2><img border="0" src="triangle2.GIF"><a href="books">Mark's Books</a><br>
  <font size="2">Whether DOS, Win 98, NT, Linux, 2000, XP, 2003 or the softwar
  you.&nbsp; (Including a free download of one of Mark's old books on Win98.)<
  <h2><img border="0" src="triangle2.GIF"><a href="gethelp">Frequently Asked Q

  <h2><img border="0" src="triangle2.GIF"><a href="mb2.htm">Bio info for Mark<

  <h2><img border="0" src="triangle2.GIF"><a href="photos/">Photography</a><br
        <font size=2>A bit of fun</font></h2>

  <h2><img border="0" src="triangle2.GIF"><a href="http://markminasi.wordpress
        Non-Computer Blog</a><br>
        <font size="2">how-to's, fun facts and opinions that mostly have nothi
        do with computers</font></h2>
        <h2><img border="0" src="triangle2.GIF"><a href="opinion.htm">Goofing
  <font size="2">Mark's personal Web pages<br>
  </font></h2>

<p><a href="rss.xml"><img src="rssicon.gif"></a> <a href="rss.xml">
<img border="0" src="rss.gif" width="36" height="14"></a>Get RSS notifications
<link rel="alternate" type="application/rss+xml" title="MR&D RSS feed" href="h
</td>
```

Analyze the Raw HTML at the Duck Pier

In Notepad, search for "WTMP)" (the label) and we find this:



```
(WTMP):</td>          <td> 43.0
```

- Another search shows no other instance of "(WTMP)"
- The Fahrenheit values will always be two digits, a period and a digit… right?
- So we need a plan to pluck out the dd.d with the data
- We can do this with a "**regular expression capture**" or "**regex**"
- Here's how it works…

Extract ("Capture") Only the Temperature

- A regex *capture* requires that I build a sort description of the entire page – yes, it could be lengthy – and inside that description, I point regex at the specific data that I want
- Again, I don't care about 99% of the HTML from the page, just the useful data
- Even if I successfully point out what I want, the capture will fail if I have not described the HTML page completely
- So here's my plan (in English, not regex):

The Plan: Regions of the HTML

We have no interest before "WTMP)," so the pattern in the beginning is "match/skip anything and keep doing it until you…"

…Match "WTMP)" and stop

From there, there's only white space and text until we hit the temperature, so we'll tell our pattern matcher to just "match/skip any non-digit until you…"

…. Match "[digit][digit].[digit]" and then save ("capture") that dd.d text into a variable called $1

From there, match/skip everything until you get to the end of the text

Result: our temperature will be in the variable

Regex Patterns: Reference

- Don't sweat the regex, but if you want to read along…
- [\S\s]+ means "any character, and at least one of them"
- WTMP\) means "WTMP)" but the ")" must be "escaped" by a \ prefixed to it
- [\D]+ means an uninterrupted string of "non-digit" characters
- (\d\d\.\d) means "two digits, period and digit" and capture them

The Regular Expression in Pieces

The parentheses mean, "extract ('capture') this text." And, you can have as many "capture groups" as you like.

```
[\S\s]+   WTMP\)  [\D]+  (\d\d\.\d)  [\S\s]+
```

[\S\s]+	WTMP\)	[\D]+	(\d\d\.\d)	[\S\s]+
Match/skip anything	Match an exact pattern of WTMP), skip to next	Match/skip past all non-digits	Match nn.n and capture it to $1	Match/skip past anything

Where the Capture Goes: Regex "Replace"

- In a regular expression, the first capture goes into a space called $1 and, yes, that looks like a PowerShell variable, but it isn't… it's just a coincidence
- If there were a second capture group, it'd go into $2, the third in $3 and so on
- The tool is –replace, and a simple (non-regex) example is
- **`PS C:\> "High seas" -replace "seas","time"`**
- **`High time`**

Doing it

- The regular expression in a variable is
- **$p='[\S\s]+WTMP\)[\D]+(\d\d\.\d)[\S\s]+'**
- Basic PoSH syntax looks like
- $result = $string –Replace PATTERN, '$1'
- In our case,
  - **$TempVal = $r.content -replace $p,'$1'**
- $TempVal is a string, but we'd like a number. Convert it like this with a "cast:"
  - **[double]$Mercury = $TempVal**

Collected…

- $URI = 'http://www.ndbc.noaa.gov/station_page.php?station=dukn7'
- $R = IWR $URI
- $p='[\S\s]+WTMP\)[\D]+(\d\d\.\d)[\S\s]+'
- $TempVal = $r.content -replace $p,'$1'
- [double]$Mercury = $TempVal
- $Mercury

Result

- Use the regex tools, you needn't become an expert
- I prototype everything with regexr.com, regex101.com
- Lots of other sites do the same thing
- The point is that after an hour of experimentation I was able to create a regex pattern that pulled out the F temperature
- That let me create a module with a cmdlet "get-duck"
- This method of getting data is called "screen scraping"

Harvest Job Two: Downloading Files

File Downloads with IWR

- We can do more than just grabbing text
- Invoke-WebRequest (alias: IWR) can download files
- Just add the -outfile parameter
- Example:
- iwr "www.minasi.com/picture.jpg" -OutFile C:\scripts\testit.jpg
- Remember, you can get a list of the images on a web page as an attribute, as in
- (iwr www.minasi.com).images.src

Harvest Job Three: Web Services

(boy, Job Two didn't take all that long, did it?)

Thus far's been nice, but...

Screen scraping is a pain in the neck. And lots of folks in need of web data don't feel like screwing around with regex patterns.

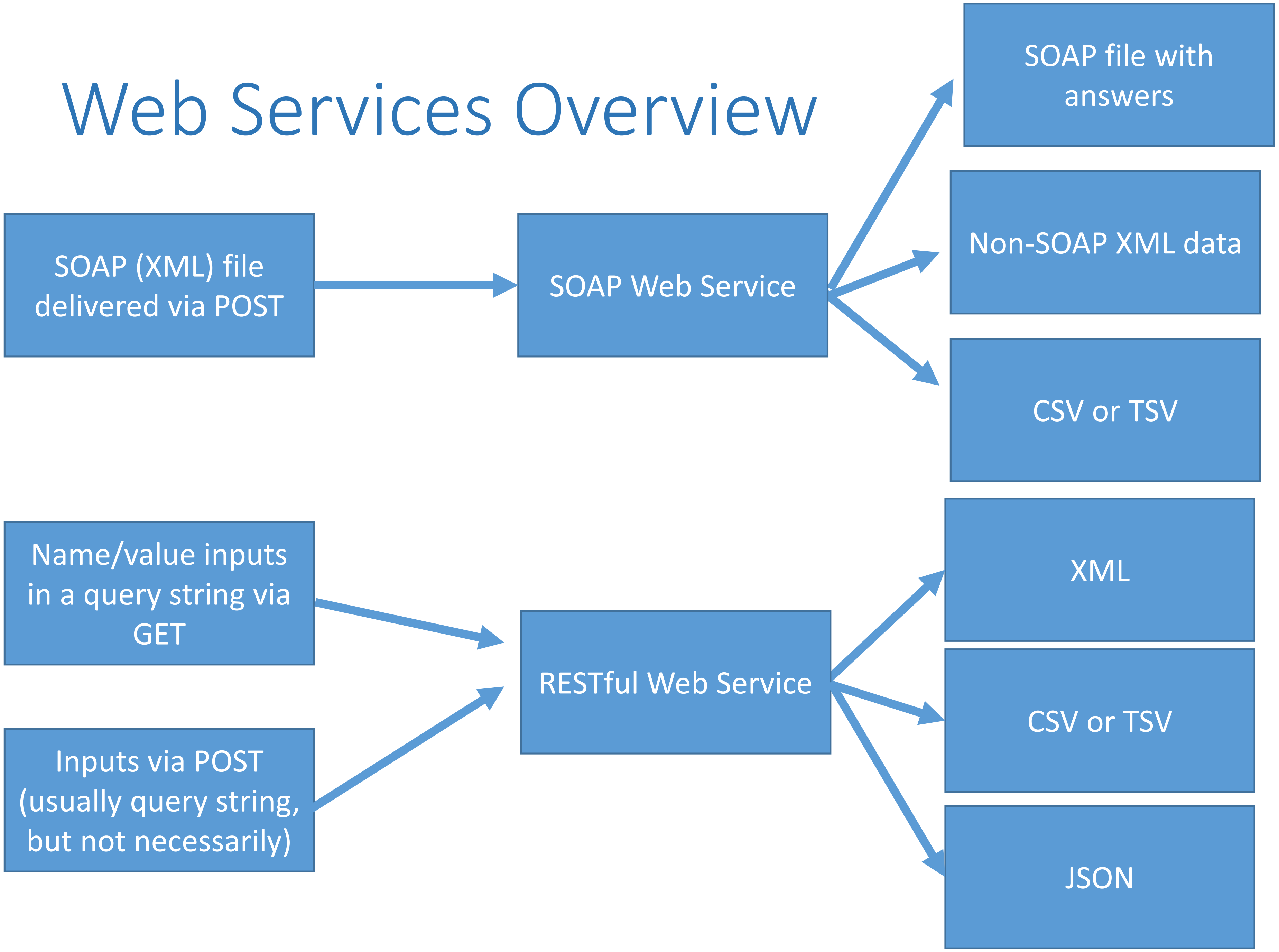
So around 2000, coders started working on a better way.

(That said, understand that sometimes screen scraping is the only way.)

Web Services: the Better Way

- We're used to getting data from a web site interactively with a client tool, a browser showing a home page or the like
- But some web sites alternatively cater to code that isn't a human-friendly browser, but instead some automated tool looking for a given chunk of data without the pictures and other stuff
- Such a site is delivering what's called a "web service"
- There are two main schools of web services… SOAP and REST

Web Services Overview


SOAP (XML) file delivered via POST
SOAP Web Service
SOAP file with answers
Non-SOAP XML data
CSV or TSV
Name/value inputs in a query string via GET
Inputs via POST (usually query string, but not necessarily)
RESTful Web Service
XML
CSV or TSV
JSON

XML and JSON Fundamentals

- Web services often deliver **structured, hierarchical** information -- an object with attributes that may be objects themselves
- We'll often want to express that data in a text form
- In the late 90s, XML caught on as a popular format
- More recently, JSON is gaining popularity
- Web services often deliver their data in XML or JSON
- Here's a quick look at them

XML Components and Terms

Element, text, attribute

- Created 1998-ish
- Looks like HTML, but it's case sensitive
- <datanow>67.4</datanow>
- That whole thing is called an *element*, and that would be called a "datanow" element
- "67.4" is called the element's *text*, which in XML-ese is its #text, which as you might guess could give PowerShell a bit of heartburn
- An alternative way to convey a data value in an element would be an *attribute*, like
- <datanow temperature="67.4"></datanow>

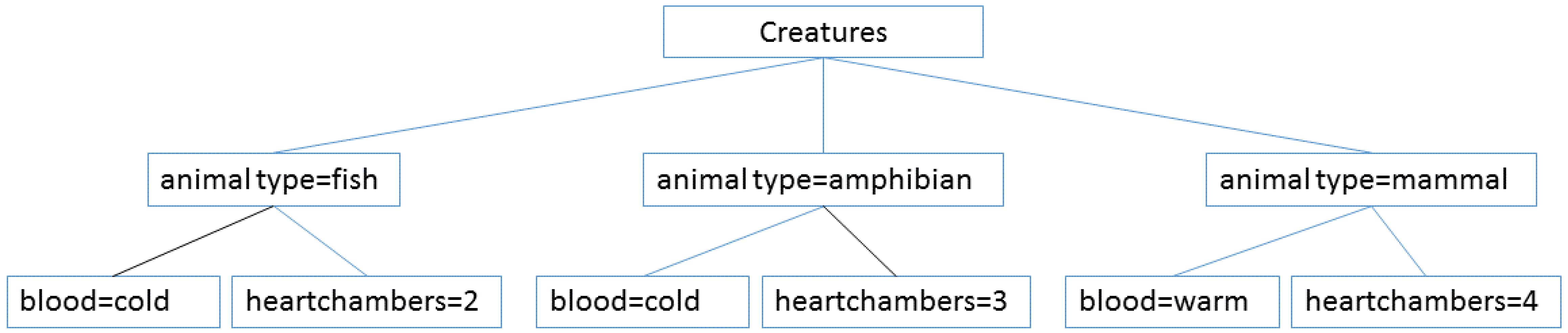
More XML Notes

- XML's hierarchy exists via elements insides elements; example:
- <datanow>
  - <readingday>Tuesday</readingday>
  - <ValueRead>67.4</ValueRead>
- </datanow>
- As everyone gets to define their own XML elements and collisions could be troublesome, XML element names can have prefixes that refer to "XML namespaces" and look like this:
- <mmspace:datanow>67.4</mmspace:datanow>
- First line in an XML object is often
- <?xml version="1.0" encoding="UTF-8"?>

JSON

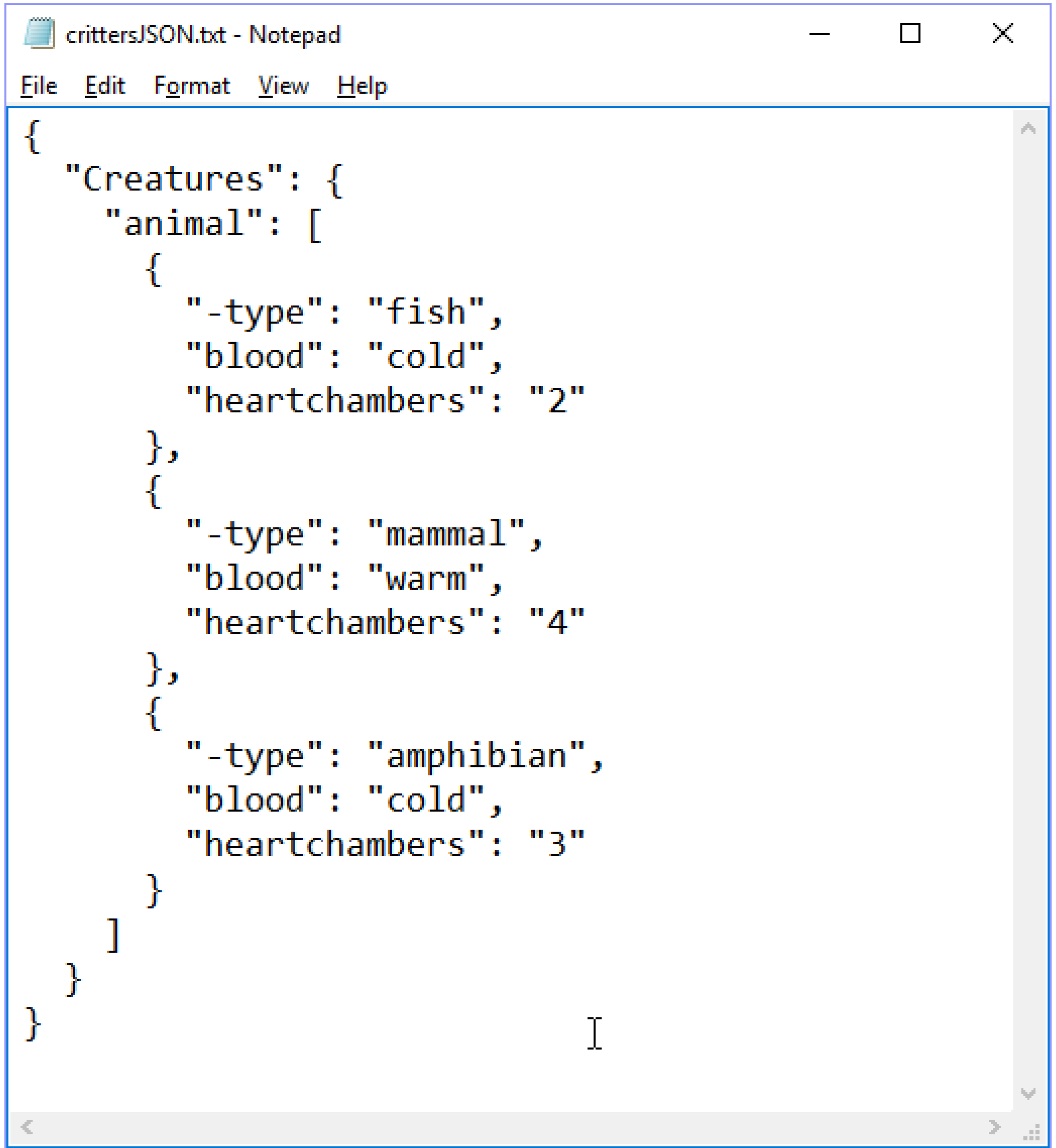
- Javascript Structured Object Notation
- Sort of like XML in its structured-ness
- The look is quite different
- A bit newer, 2006-ish
- So let's look at a small set of hierarchical data and render it in XML and JSON

Example Hierarchy: Animal Classification



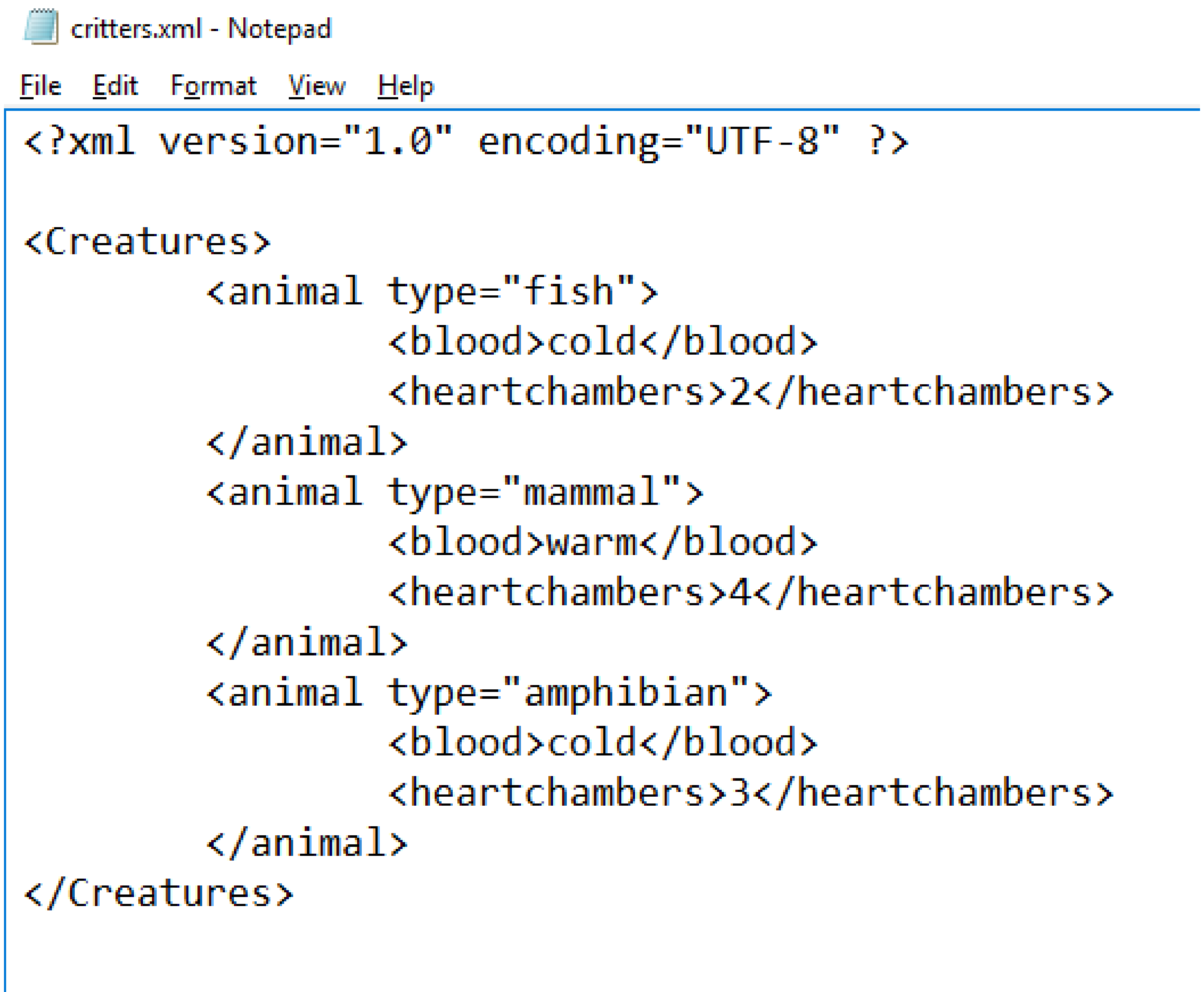
Three instances, each with two characteristics -- blood type and number of heart chambers

In JSON



```
{
  "Creatures": {
    "animal": [
      {
        "-type": "fish",
        "blood": "cold",
        "heartchambers": "2"
      },
      {
        "-type": "mammal",
        "blood": "warm",
        "heartchambers": "4"
      },
      {
        "-type": "amphibian",
        "blood": "cold",
        "heartchambers": "3"
      }
    ]
  }
}
```

In XML



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

<Creatures>
	<animal type="fish">
		<blood>cold</blood>
		<heartchambers>2</heartchambers>
	</animal>
	<animal type="mammal">
		<blood>warm</blood>
		<heartchambers>4</heartchambers>
	</animal>
	<animal type="amphibian">
		<blood>cold</blood>
		<heartchambers>3</heartchambers>
	</animal>
</Creatures>
```

Web Service Types: SOAP

- System Object Access Protocol = SOAP
- The notion was invented around 2000
- The idea was to POST a structured XML message (in "SOAP" format) with inputs for the request
- The response shows up as another XML SOAP message with the answer delivered in the SOAP XML
- This has largely been replaced by the alternative model, REST, but there are many sites on the Web that still deliver via SOAP

Web Services: RESTful Services

- REST="representational state transformation"
- Yup, just about as useful as SOA or SMB acronym-wise
- More simply, it means that you basically just send URLs with "field/value pairs" as a "query string"
- Data is then delivered as some kind of structured text, either CSV, XML or the JSON
- XML and JSON let you deliver hierarchical data, like objects -- things with attributes
- Although SOAP is older, we'll start from REST, as it's far more common

How High is the Rainbow River?

a REST example

- The USGS maintains a lot of geological and hydrological data from sensors around the country
- One is a height gauge on the Rainbow River in Florida
- (It's an unusual river in that it is fed solely by springs, not runoff or snowmelt -- the springs produce a half-billion gallons of water every day)
- It's pretty stable, but with enough rainfall the springs do get a bit more active, albeit delayed by a month
- If it gets to a depth of about seven feet at the sensor, things would be messy for me, so it's nice to be able to get status info from anywhere

A Sample Rainbow Query

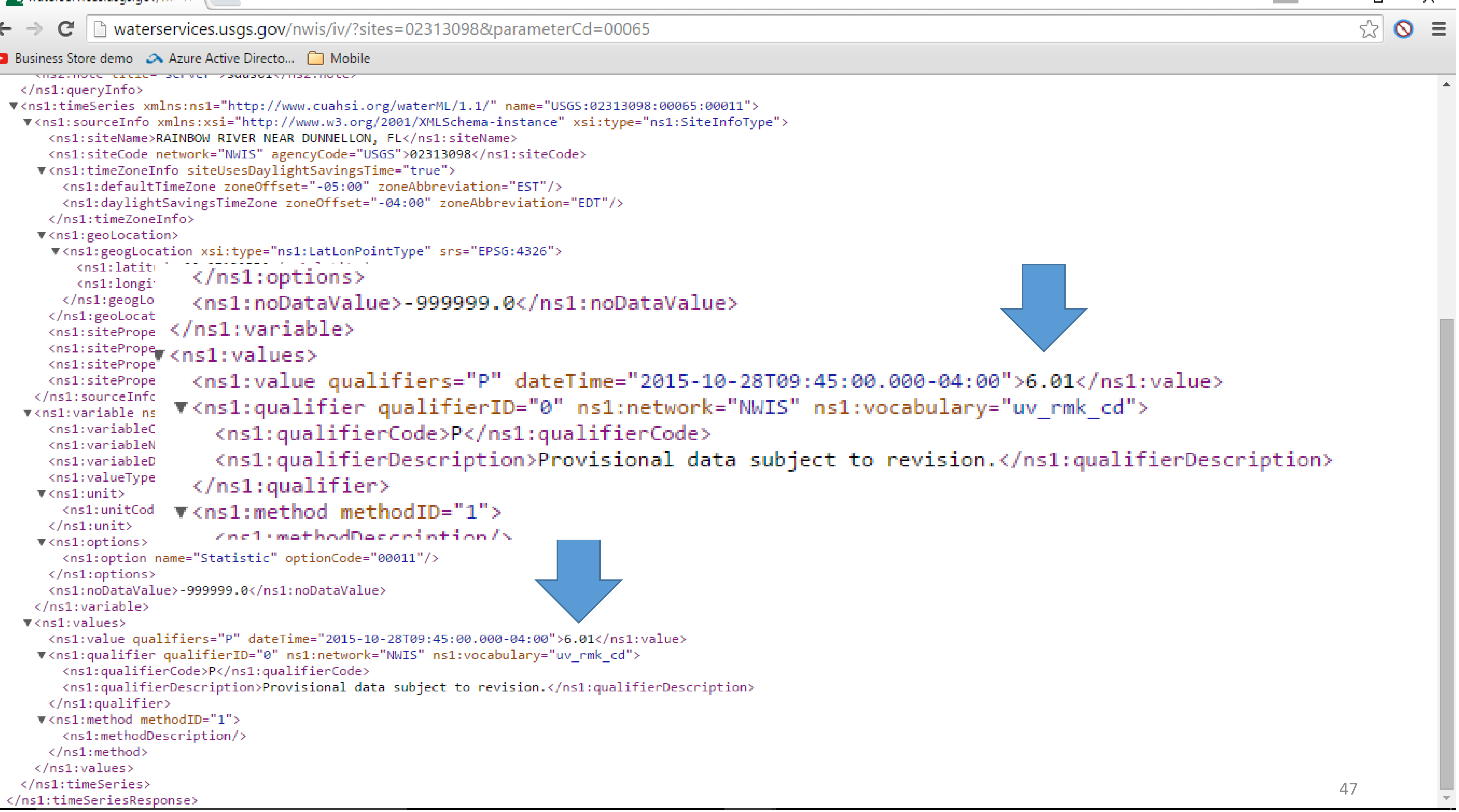
- This will return the depth of the river at the last gauge (they write it "gage") reading:
- http://waterservices.usgs.gov/nwis/iv/?sites=02313098&parameterCd=00065
- But before we look at the results of that, let's do a quick sidebar on this thing:
- ?sites=02313098&parameterCd=00065
- Notice that this "RESTful" query is just a long URL
- How did I know? They have a nice tool to build URLs

Web Review: Query Strings

- Sometimes we need to pass parameters to a Web site, as in the previous case -- 02313098 was the sensor's name and 00065 said, "I want the river depth," as opposed to its speed, temperature or nitrogen content
- That URL was a "query string" and has two parts: a base URL (waterservices.usgs.gov/nwis/iv/) and field/name pairs formatted as a question mark, a pair, an ampersand ("&"), another pair, etc, as in
- xyz.com/data/?username=jack&query=ssn&format=XML

Time to Play, "Find the Element in the XML"

- Your data comes back in XML
- There will be piles of other junk in there besides your simple "how high is the friggin' water?" query
- First, I look for a version of the web site that is browser-friendly so I can get the current height, like 6.01
- Then I run the web service, get its XML and either put it into Chrome or XML Notepad
- Then I search on the current data value to find where it sits in the XML hierarchy



waterservices.usgs.gov/nwis/iv/?sites=02313098&parameterCd=00065

```
  </ns1:queryInfo>
▼<ns1:timeSeries xmlns:ns1="http://www.cuahsi.org/waterML/1.1/" name="USGS:02313098:00065:00011">
  ▼<ns1:sourceInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns1:SiteInfoType">
     <ns1:siteName>RAINBOW RIVER NEAR DUNNELLON, FL</ns1:siteName>
     <ns1:siteCode network="NWIS" agencyCode="USGS">02313098</ns1:siteCode>
   ▼<ns1:timeZoneInfo siteUsesDaylightSavingsTime="true">
       <ns1:defaultTimeZone zoneOffset="-05:00" zoneAbbreviation="EST"/>
       <ns1:daylightSavingsTimeZone zoneOffset="-04:00" zoneAbbreviation="EDT"/>
     </ns1:timeZoneInfo>
   ▼<ns1:geoLocation>
     ▼<ns1:geogLocation xsi:type="ns1:LatLonPointType" srs="EPSG:4326">
         <ns1:latit
         <ns1:longi
       </ns1:geogLo
     </ns1:geoLocat
     <ns1:sitePrope
     <ns1:sitePrope
     <ns1:sitePrope
     <ns1:sitePrope
   </ns1:sourceInfc
  ▼<ns1:variable ns
     <ns1:variableC
     <ns1:variableN
     <ns1:variableD
     <ns1:valueType
   ▼<ns1:unit>
       <ns1:unitCod
     </ns1:unit>
   ▼<ns1:options>
       <ns1:option name="Statistic" optionCode="00011"/>
     </ns1:options>
     <ns1:noDataValue>-999999.0</ns1:noDataValue>
   </ns1:variable>
  ▼<ns1:values>
     <ns1:value qualifiers="P" dateTime="2015-10-28T09:45:00.000-04:00">6.01</ns1:value>
   ▼<ns1:qualifier qualifierID="0" ns1:network="NWIS" ns1:vocabulary="uv_rmk_cd">
       <ns1:qualifierCode>P</ns1:qualifierCode>
       <ns1:qualifierDescription>Provisional data subject to revision.</ns1:qualifierDescription>
     </ns1:qualifier>
   ▼<ns1:method methodID="1">
       <ns1:methodDescription/>
     </ns1:method>
   </ns1:values>
 </ns1:timeSeries>
</ns1:timeSeriesResponse>
```

```
   </ns1:options>
   <ns1:noDataValue>-999999.0</ns1:noDataValue>
 </ns1:variable>
▼<ns1:values>
   <ns1:value qualifiers="P" dateTime="2015-10-28T09:45:00.000-04:00">6.01</ns1:value>
 ▼<ns1:qualifier qualifierID="0" ns1:network="NWIS" ns1:vocabulary="uv_rmk_cd">
     <ns1:qualifierCode>P</ns1:qualifierCode>
     <ns1:qualifierDescription>Provisional data subject to revision.</ns1:qualifierDescription>
   </ns1:qualifier>
 ▼<ns1:method methodID="1">
```

Oh, So It's Just a Matter Of...

- Picking out the object in the result
- And the answer was:
- $wtreq.CompositeObservation.result.Composite.valueComponents.Array.valueComponents.Composite.valueComponents.Array.valueComponents.Composite.valueComponents.CompositeValue.valueComponents.quantity
- Yuk. Time for a better way

So How Do I Extract That?

- You could figure out the structure and pluck out just the 6.01… but trust me, you don't want to do that
- Instead, note that the XML **element** with the data we want is named "value" in an XML namespace called "ns1:"
- <ns1:value qualifiers="P" dateTime="2015-10-28T09:45:00.000-04:00">6.01</ns1:value>
- (Remember, XML is -- unlike HTML -- case-sensitive)
- That lets us do something called an "XPath query" to pluck out that data

First, Capture the Output

- $URI = "http://waterservices.usgs.gov/nwis/iv/?sites=02313098&parameterCd=00065"
  [string]$MyResult=(Invoke-WebRequest $URI)
- Notice the [string]... that is a "cast," which forces PowerShell to deliver that not as XML but as a string
- The next cmdlet needs the data as a string
- It'll extract the 6.01 number

How XPath Delivers the Number

- XPath is a query language (yes, you heard that right, *another* query language) that delivers parts of an XML document based on a query
- Here, we want to say
  - Find the **element** named <value>, regardless of its namespace
  - Extract from the element its "**#text**," the actual 6.01 value in this case (it'll vary every time we call it, of course)
  - By the way, the time of the measurement is an XML **attribute**

I'll Spare You the XPath Lecture

- Here's the query string:
- "//*[local-name()='value']")
- //* means search the XML all over
- [local-name()='value'] means find me an element whose name is "value," irrespective of any namespaces
- That finds the node -- then we'll tell PowerShell to give us the text (the 6.01) via ".node.'#'"

This Gets the Water Height

```
$URI =
'http://waterservices.usgs.gov/nwis/iv/?sites=02313098&parameterCd=000
65'

[string]$MyResult=(Invoke-WebRequest $URI)

$DesiredNode = (Select-XML -content $MyResult -XPath "//*[local-
name()='value']")

$WaterHeight = $desiredNode.node.'#text'
$waterHeight
```

Reviewing…

- Get your URL for your REST service
- Have it dump its data out in XML
- Look at the XML, find the name of the element that contains the data you want
- Use my Select-XML command and change "value" to whatever your element name is

Deflecting JSON: Invoke-RestMethod

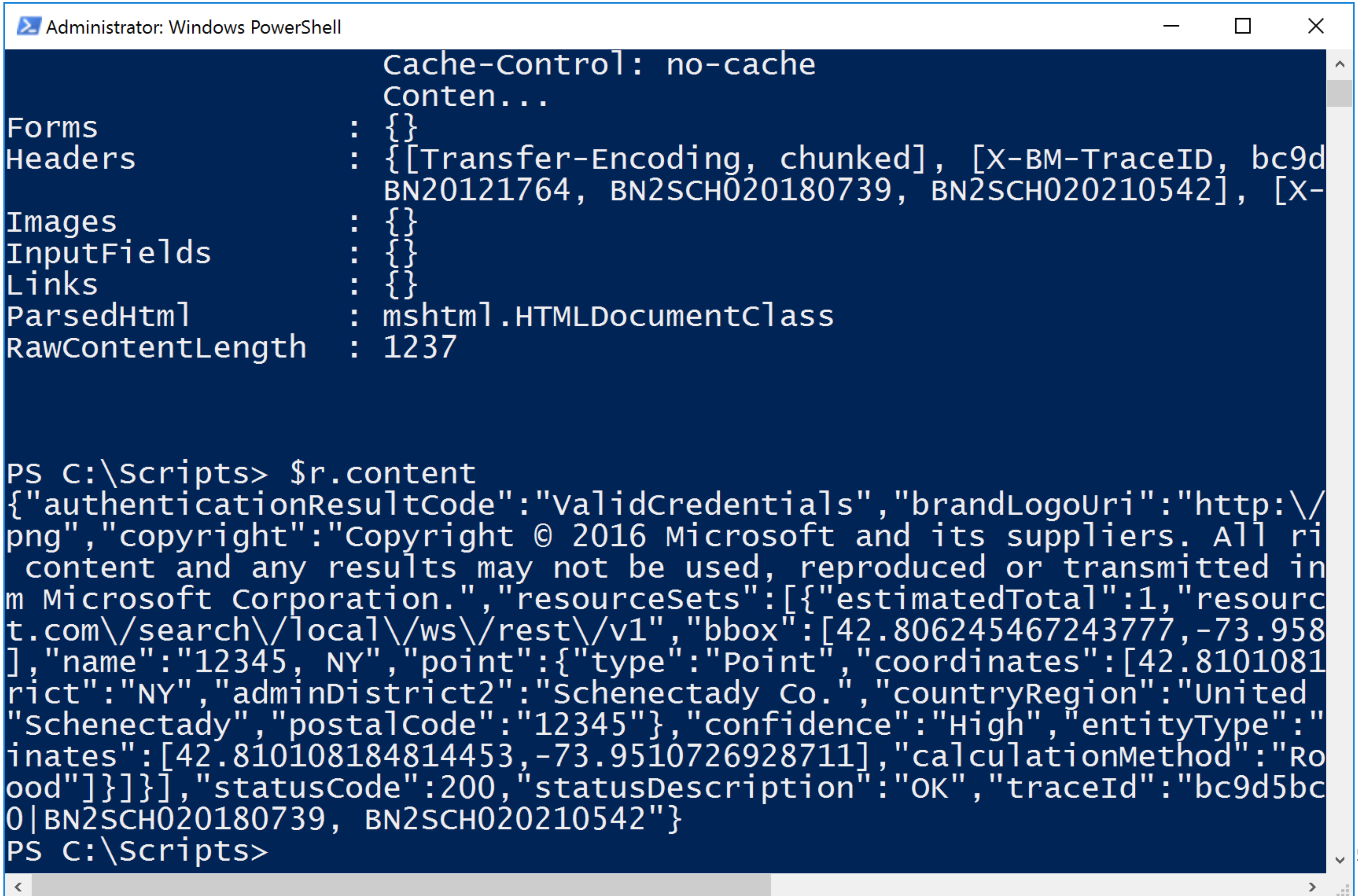
- Next let's tackle a RESTful service that returns its responses in JSON
- (Unfortunately, there's no Xpath for JSON)
- We'll try the BING geolocation service which can take a postal code as input and return the zip code's country, county, state and city

Get a Key

- Bing lets you do 125,000 lookups a year free, but you need a key
- Just go to at https://msdn.microsoft.com/en-us/library/gg650598.aspx and punch in your MSA
- Looks like “P8xPNZEHQTY9fKoDRw0NyOjysoCN3zV-Tbh-AEQBNkRSk8fsPg9916a9gOL7cQ” http://dev.virtualearth.net/REST/v1/Locations?q=12345,%20USA&key=BingKey
- I'm putting mine in $BingKey for these examples

Where is Zip 12345?

- Just build an URL looking like
- http://dev.virtualearth.net/REST/v1/Locations?q=12345,USA&key=*KEY*
- This works:
- $URI="http://dev.virtualearth.net/REST/v1/Locations?q=12345,USA&key=" + $BingKey
- $r=IWR $URI
- Look at it:
- $R.Content


Administrator: Windows PowerShell

```
                    Cache-Control: no-cache
                    Conten...
Forms             : {}
Headers           : {[Transfer-Encoding, chunked], [X-BM-TraceID, bc9d
                    BN20121764, BN2SCH020180739, BN2SCH020210542], [X-
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 1237



PS C:\Scripts> $r.content
{"authenticationResultCode":"ValidCredentials","brandLogoUri":"http:\/
png","copyright":"Copyright © 2016 Microsoft and its suppliers. All ri
 content and any results may not be used, reproduced or transmitted in
m Microsoft Corporation.","resourceSets":[{"estimatedTotal":1,"resourc
t.com\/search\/local\/ws\/rest\/v1","bbox":[42.80624546724377,-73.958
],"name":"12345, NY","point":{"type":"Point","coordinates":[42.8101081
rict":"NY","adminDistrict2":"Schenectady Co.","countryRegion":"United
"Schenectady","postalCode":"12345"},"confidence":"High","entityType":"
inates":[42.81010818481445 3,-73.951072692871 1],"calculationMethod":"Ro
ood"]}]}],"statusCode":200,"statusDescription":"OK","traceId":"bc9d5bc
0|BN2SCH020180739, BN2SCH020210542"}
PS C:\Scripts>
```

Hmmm… It Appears JSON Is…

- … wearing his hockey mask
- Let's just use JSON-Path (like Xpath)… oops, there isn't one
- Fortunately, there is a tool that will take JSON-y data and make it a PowerShell object, ConvertFrom-JSON
- But there's an easier way: Invoke-Restmethod
- Replaces IWR when you expect JSON output
- $R = Invoke-Restmethod $URI
- "IRM" is its alias

Now We Can Crack it Open

- Look at $R... looks like ResourceSets is the only useful data
- $R.ResourceSets shows what's there
- That contains "resources" so look there:
- $R.ResourceSets.resources
- Contains "addresses," so examine them:
- $R.ResourceSets.resources.addresses
- Now we're talking!
- Look also at .point, .confidence
- Then try it on zip 90000

A SOAP Example

- Most SOAP services are on the way out, or, worse, not maintained and return useless answers, but here's a mortgage calculator:
- $URI='http://www.webservicex.net/mortgage.asmx?wsdl'
- $Proxy = New-WebserviceProxy $URI –Namespace X
- $Proxy | get-member -MemberType Method Get*
- Look at the docs at www.webservicex.net/mortgage.asmx and you see the desired inputs, which will give us clues about how to call the service

Using $Proxy

- It contains "methods," functions that can query the web service to calculate for you
- Note the one named GetMortgagePayment
- So try this:
- $Proxy.GetMortgagePayment(30,1.0,360000,0,0)

Retrieving 400/500 Errors

- You don't always get a lot of "what caused this?" from Invoke-RestMethod or Invoke-WebRequest, but you can…

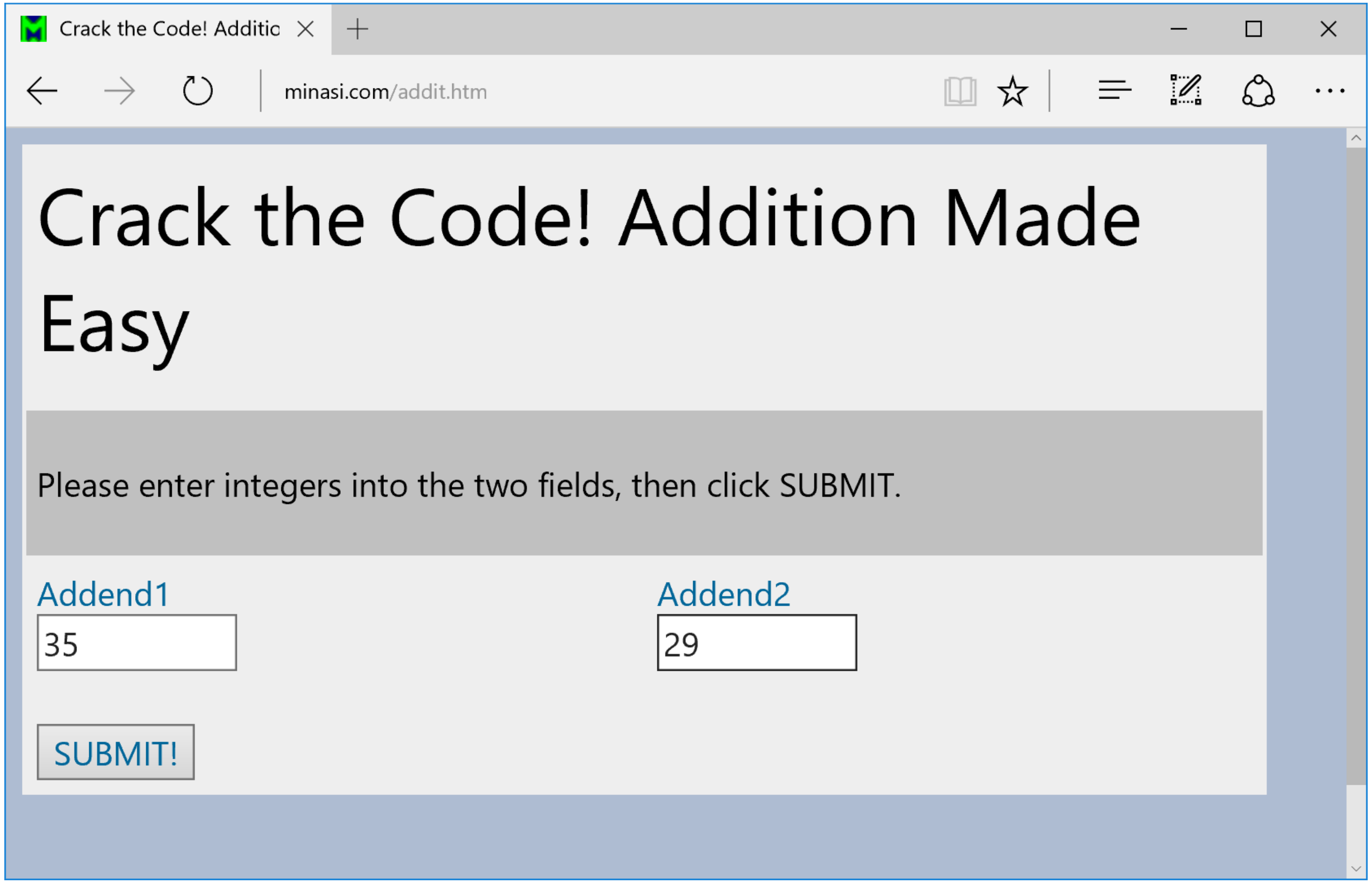
```
Try {$R = IWR $URI -infile T1.XML -Method POST -ContentType "text/xml"}
Catch {
	$result = $_.Exception.Response.GetResponseStream()
	$reader = New-Object System.IO.StreamReader($result)
	$responseBody = $reader.ReadToEnd()
	$responsebody
}
```

Harvest Task Four: Forms

Forms and PowerShell

- Web services are great, but the fact is that you can get more web sites to cough up their data after you fill in a form (and then do some screen scraping)
- Here's an example I built at http://www.minasi.com/addit.htm
- You fill in two numeric fields, click the SUBMIT! button and the system returns a page with the sum of the two numbers
- Result page is http://addit-worker.asp

Fill in the Form



Crack the Code! Additic

minasi.com/addit.htm

Crack the Code! Addition Made Easy

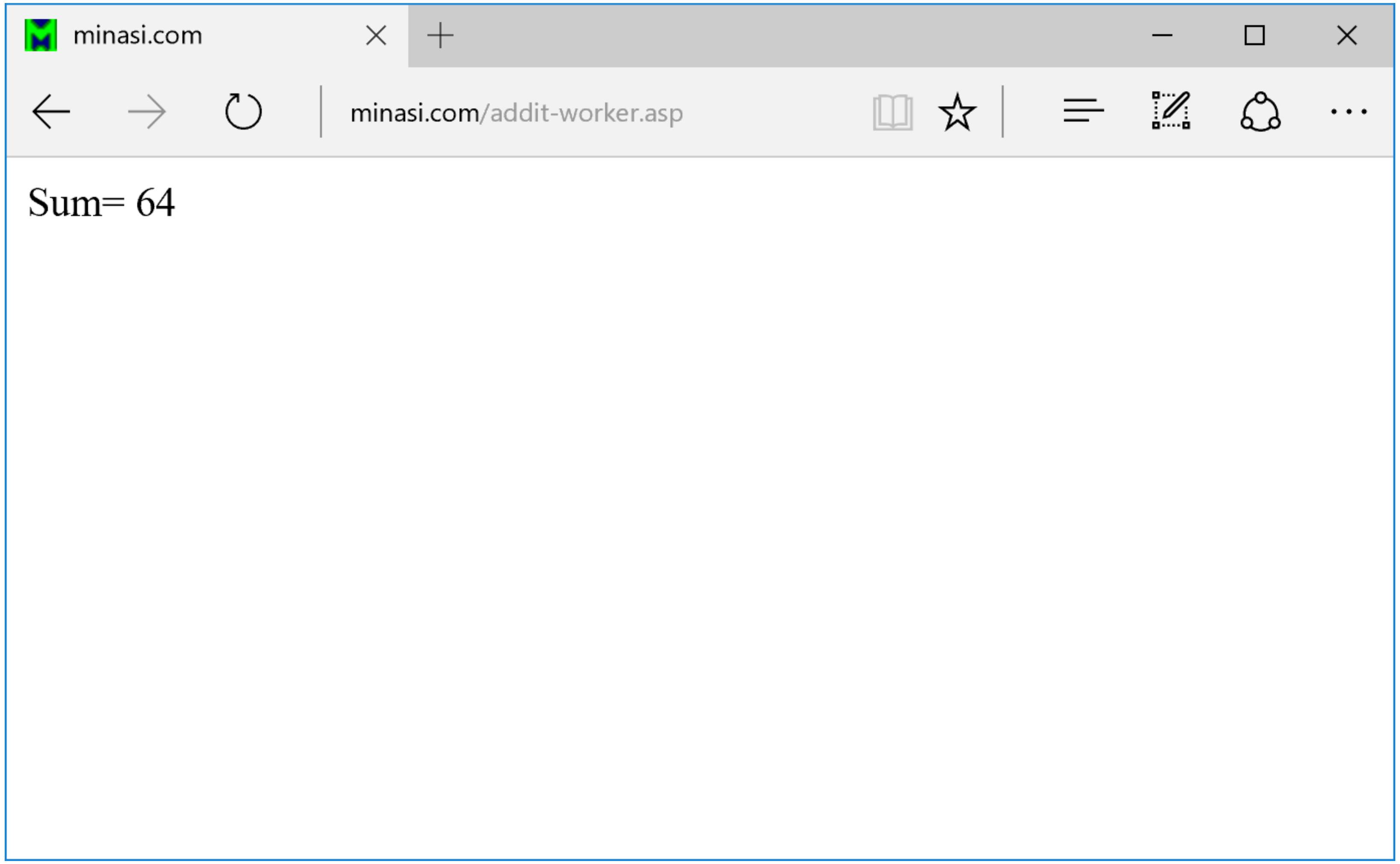
Please enter integers into the two fields, then click SUBMIT.

Addend1: 35

Addend2: 29

SUBMIT!

Result



```
Sum= 64
```

Now Let's Let PowerShell Do It

- Gather data:

```
$uri="http://minasi.com/addit.htm"
$r = iwr $uri
$r.Forms
```

- Result:

```
Fields
------
{[addend1, ], [addend2, ], [B1, SUBMIT!]}
```

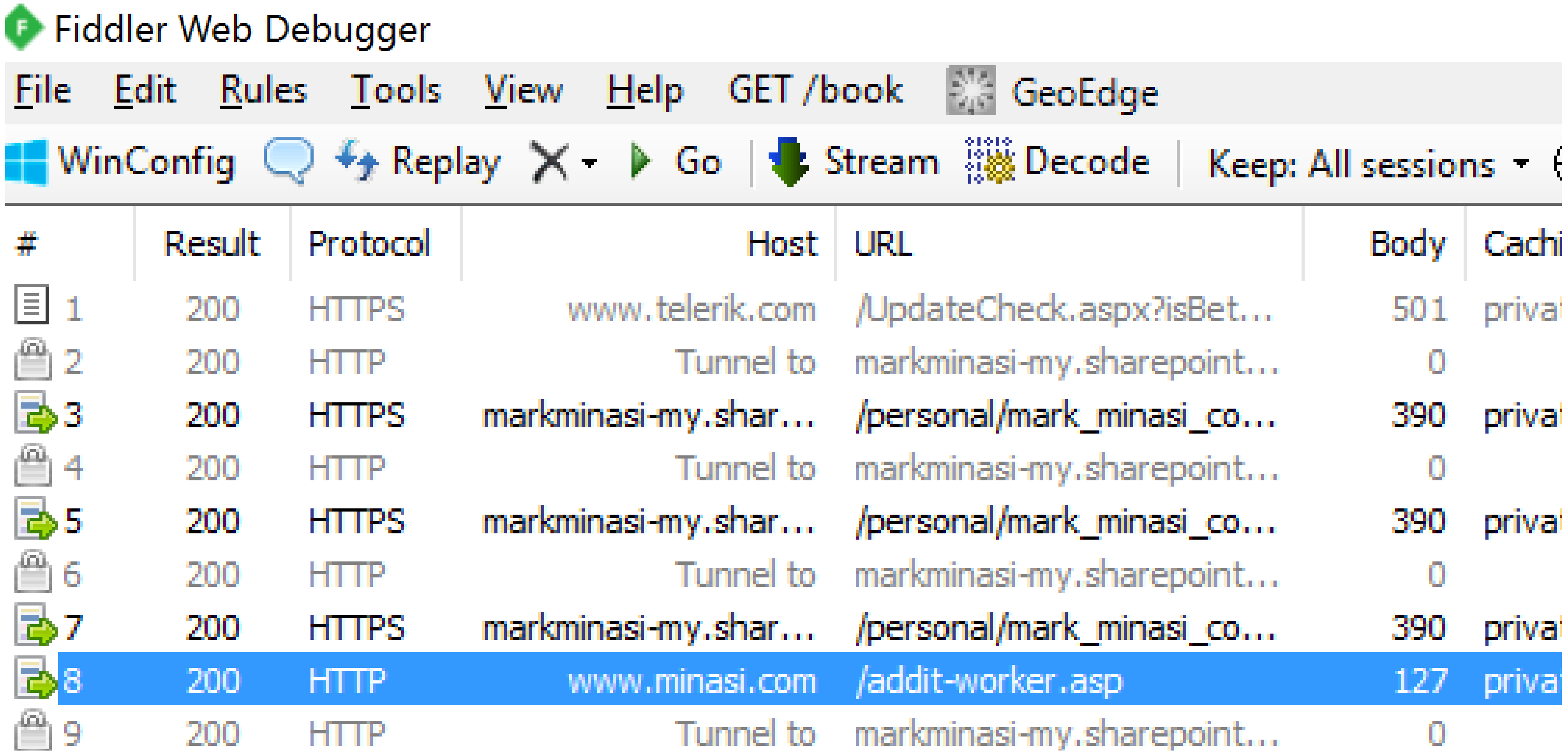
My Next Question…

- … how to push the button?
- Actually, you don't… run a network sniffer or Fiddler or something like that, and you'll see that when the page comes back, it turns out not to be addit.htm but addit-worker.asp and a querystring:
- addend1=55&addend2=82&B1=SUBMIT%21
- Remember the –body parameters? The querystring goes in there

An Example Tool: Fiddler

- Download it from telerik.com
- Start it up
- Open your browser, fill in the fields, click SUBMIT!
- You'll see in the left-hand pane a reference to addit-helper.asp – click that, look the right and choose "TextView"
- In the text field below it, you see the querystring
- Again, now we have the pieces

Fiddler Left Pane

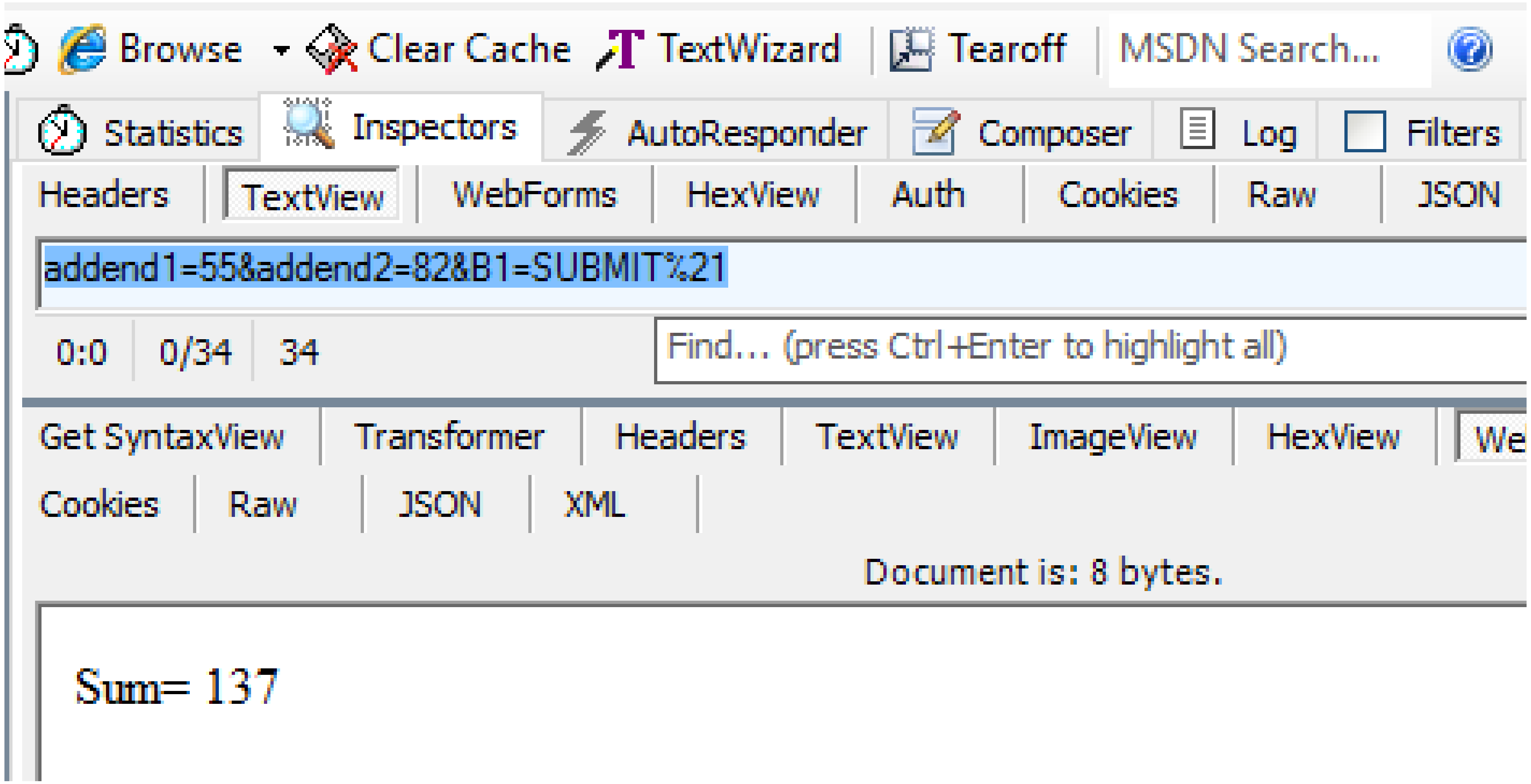

Fiddler Web Debugger

File Edit Rules Tools View Help GET /book GeoEdge

WinConfig Replay X Go Stream Decode Keep: All sessions

#	Result	Protocol	Host	URL	Body	Cachi
1	200	HTTPS	www.telerik.com	/UpdateCheck.aspx?isBet...	501	priva
2	200	HTTP	Tunnel to	markminasi-my.sharepoint...	0	
3	200	HTTPS	markminasi-my.shar...	/personal/mark_minasi_co...	390	priva
4	200	HTTP	Tunnel to	markminasi-my.sharepoint...	0	
5	200	HTTPS	markminasi-my.shar...	/personal/mark_minasi_co...	390	priva
6	200	HTTP	Tunnel to	markminasi-my.sharepoint...	0	
7	200	HTTPS	markminasi-my.shar...	/personal/mark_minasi_co...	390	priva
8	200	HTTP	www.minasi.com	/addit-worker.asp	127	priva
9	200	HTTP	Tunnel to	markminasi-my.sharepoint...	0	

Fiddler Right Pane



```
addend1=55&addend2=82&B1=SUBMIT%21
```

Document is: 8 bytes.

Sum= 137

Assembling the Query

- $uri = "http://minasi.com/addit-worker.asp"
- $body = "addend1=55&addend2=82&B1=SUBMIT%21"
- $R = IWR $URI -method POST -body $body
- Type "$R.content," and you see
- Sum= 137

That's All We Have Time For

- Thank you for attending
- I tweet at mminasi
- Please do an evaluation, thanks again for coming!