

IT452 Advanced Web and Internet Systems

Set 9: Web Services
(Chapter 28, loosely)

Web Services

- “any service available on the Web that has been designed for consumption by programs, independent of the technology being used”
- Two primary camps
 - REST (sometimes just “HTTP”)
 - “Representational State Transfer”
 - Exchanging documents
 - Many HTTP actions
 - SOAP
 - Exchanging messages, or RPC
 - Mainly HTTP POST

REST

- Use all of HTTP for sensible document management and caching:
 - POST – make new document
 - HEAD – get doc metadata
 - GET – retrieve document (no state change)
 - PUT – update document
 - DELETE – delete document
- Requests – all state info is in the URL
- Response format?
- In practice – often GET for everything
 - Works in browser
 - But violates “no side effects” rule

SOAP

- Originally “Simple Object Access Protocol”
- Two views
 - 1. Exchanging messages
 - 2. Performing RPC
- Request
 - Mostly POST (but need not be just HTTP!)
 - A complex XML document
 - What parameters/functions are legal??
- Response format: XML

REST Example 1

- Get the weather from wunderground.com
- <http://www.wunderground.com/weather/api/d/documentation.html>

(Ex 1) Weather XML Data

From: <http://api.wunderground.com/api/XXX/conditions/forecast/q/21409.xml>

```
<current_observation>
<display_location>
<full>Annapolis, MD</full>
<city>Annapolis</city>
<state>MD</state>
<state_name>Maryland</state_name>
<country>US</country>
<country_iso3166>US</country_iso3166>
<zip>21409</zip>
<latitude>39.02930832</latitude>
<longitude>-76.43528748</longitude>
<elevation>6.00000000</elevation>
</display_location>
<estimated></estimated>
<station_id>KMDANNAP10</station_id>
<observation_time>Last Updated on March 9, 3:08 PM EST</observation_time>
<observation_time_rfc822>Fri, 09 Mar 2012 15:08:35 -0500</observation_time_rfc822>
<observation_epoch>1331323715</observation_epoch>
<local_time_rfc822>Fri, 09 Mar 2012 15:09:00 -0500</local_time_rfc822>
<local_epoch>1331323740</local_epoch>
<local_tz_short>EST</local_tz_short>
<local_tz_long>America/New_York</local_tz_long>
<local_tz_offset>-0500</local_tz_offset>
<weather>Clear</weather>
<temperature_string>56.1 F (13.4 C)</temperature_string>
<temp_f>56.1</temp_f>
```

(Ex 1) weather.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head> <title>Web Services using XSLT</title>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.5/jquery.min.js"></script>
    <script type="text/javascript" src="transform.js"> </script>
  </head>
  <body>
    <h3>This is a webpage.</h3>
    <p>You have a lot of content on the page, and want to localize it for the user.</p>
    <p>One easy way is to provide the weather!</p>
    <p>Let's paste in your local weather using the wunderground.com web service, ask for a
 zipcode, and then use XSLT to transform the result into some nice XHTML.</p>
    <p>We'll paste the result below.</p>
    <p><b>Type your zip code</b>: <input type="text" id="zipcode" />
       <input type="button" value="Get weather!" onclick="getWeather()" />
    </p>

    <div id="planet">
      <h2>This is where the transformed XML in XHTML form will appear.</h2>
    </div>
  </body>
</html>
```

(Ex 1) transform.js

```
function getWeather() {  
    var zip = $("#zipcode").val();  
    var url = "wunderground.pl?zipcode=" + zip;  
    transform("wunderground.xsl", url);  
}  
  
function transform (xslFileName, url) {  
    // ASIDE: Why won't this work?  
    // xmlhttp.open("GET", "http://api.wunderground.com/api/XXXXXX/conditions/forecast/q/21409.xml", false);  
  
    // Get the XSLT file  
    var xslhttp = new XMLHttpRequest();  
    xslhttp.open("GET", xslFileName, false);  
    xslhttp.send('');  
  
    // Get the XML input data  
    var xmlhttp = new XMLHttpRequest();  
    xmlhttp.open("GET", url, false);  
    xmlhttp.send('');  
  
    // Transform the XML via the XSLT  
    var processor = new XSLTProcessor();  
    processor.importStylesheet(xslhttp.responseXML);  
    var newDocument = processor.transformToDocument(xmlhttp.responseXML);  
  
    // Replace part of original document with the new content  
    var o = document.getElementById("planet");  
    var n = newDocument.getElementById("planet");  
    o.parentNode.replaceChild(n, o);  
}
```

(Ex 1) wunderground.pl

```
#!/usr/bin/perl
use CGI ":standard";
use strict;

# Need to have "!head" to avoid loading the head function from LWP::Simple.
# - The above CGI module also has a head function...
use LWP::Simple "!head";
use LWP::UserAgent;
use HTTP::Request;
use HTTP::Response;

# We want to send XML back
print "Content-type: text/xml\n\n";

# Construct URL to get the weather
my $zip = param("zipcode");
my $URL = "http://api.wunderground.com/api/XXXXX/conditions/forecast/q/$zip.xml";

# Get the XML document and send it back to requestor (the browser)
my $contents = get($URL);
print $contents;
```

(Ex 1) wunderground.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" >
  <xsl:template match="/">
    <html><head>
      <title> Current weather in: <xsl:value-of select="/response/termsOfService"/> </title>
    </head>
    <body>
      <div id="planet">
        <h1>Current weather in:
          <xsl:value-of select="/response/current_observation/display_location/full"/> </h1>
          <xsl:apply-templates select="/response/current_observation"/>
          <xsl:apply-templates select="/response/forecast/simpleforecast"/>
        </div>
      </body>
    </html>
  </xsl:template>

  <!-- Handle current conditions -->
  <xsl:template match="current_observation">
    <table><tr><td></td>
    <td><xsl:value-of select=".//weather" /></td></tr></table>
    <ul>
      <li>Temperature: <xsl:value-of select=".//temperature_string" /> </li>
      <li>Wind: <xsl:value-of select=".//wind_string" /> </li>
      <li>Gusts: <xsl:value-of select=".//wind_gust_mph" /> </li>
      <li>Dew Point: <xsl:value-of select=".//dewpoint_string" /> </li>
    </ul>
  </xsl:template>
...
```

SOAP Example 2

- Search flickr.com and show photos on your page.

(Ex 2) Sample SOAP request

```
<s:Envelope
    xmlns:s='http://www.w3.org/2003/05/soap-envelope'
    xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance'
    xmlns:xsd='http://www.w3.org/1999/XMLSchema'
>
<s:Body>
    <x:FlickrRequest xmlns:x='urn:flickr'>
        <method>flickr.photos.search</method>
        <name>value</name>
        <tags>tigers</tags>
        <privacy_filter>1</privacy_filter>

        <per_page>5</per_page>
        <api_key>83ec7bab1628defd47d893288348fee5</api_key>
    </x:FlickrRequest>
</s:Body>
</s:Envelope>
```

Online API: <http://www.flickr.com/services/api/flickr.photos.search.html>

(Ex 2) Photos XML Data

```
<?xml version="1.0" encoding="utf-8" ?>
<s:Envelope
    xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema" >
    <s:Body>
        <x:FlickrResponse xmlns:x="urn:flickr">

<photos page="1" pages="20668" perpage="5" total="103339">
    <photo id="2944625312" owner="41086422@N00" secret="1975114cb7" server="3057" farm="4"
        title="Bad Mascot" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="2944368362" owner="29542413@N07" secret="0f3f076cd1" server="3020" farm="4"
        title="_MG_3447" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="2943510303" owner="29542413@N07" secret="7c04e22d9b" server="3283" farm="4"
        title="_MG_3462" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="2944369890" owner="29542413@N07" secret="fe9271a3b0" server="3035" farm="4"
        title="_MG_3454" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="2944370484" owner="29542413@N07" secret="451a349bb0" server="3184" farm="4"
        title="_MG_3456" ispublic="1" isfriend="0" isfamily="0" />

</photos>
    </x:FlickrResponse>
    </s:Body>
</s:Envelope>
```

(Ex 2) photos.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head> <title>XSLT Example with Web services</title>
    <script
      src="http://ajax.googleapis.com/ajax/libs/jquery/1.5/jquery.min.js"></script>
    <script type="text/javascript" src="transform2.js"> </script>
  </head>

  <body>
    <p>Lets now use SOAP to access photos from flickr!</p>
    <p>Tags to search for:<br/>
      <input type="text" id="tags" />
      <input type="button" value="Get photos!" onclick="getPhotos()" />
    </p>

    <div id="planet">
      <h2>This is where the transformed XML in XHTML form will appear.</h2>
    </div>
  </body></html>
```

(Ex 2) flickr.pl (part 1)

```
#!/usr/bin/perl
use CGI ":standard";

# Need this to get web pages from Perl
use LWP::Simple "!head";
use HTTP::Request;
use LWP::UserAgent;

# We want to send XML back
print "Content-type: text/xml\n\n";

my $ua = LWP::UserAgent->new();
my $method = "POST";
my $url = "http://api.flickr.com/services/soap/";

my $tags = param("tags");

my $content =
<s:Envelope
    xmlns:s='http://www.w3.org/2003/05/soap-envelope'
    xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance'
    xmlns:xsd='http://www.w3.org/1999/XMLSchema'
>
    <s:Body>
        <x:FlickrRequest xmlns:x='urn:flickr'>
            <method>flickr.photos.search</method>
            <name>value</name>
            <tags>$tags</tags>
            <privacy_filter>1</privacy_filter>
            <per_page>5</per_page>
            <api_key>83ec7bab1628defd47d893288348fee5</api_key>
        </x:FlickrRequest>
    </s:Body>
</s:Envelope>
";
```

(Ex 2) flickr.pl (part 2)

```
use HTTP::Headers;
my $header = HTTP::Headers->new();
my $request = HTTP::Request->new($method, $url, $header, $content);

use HTTP::Response;
my $response = $ua->request($request);
if($response->is_success) {
    # The Flickr response via SOAP is encoded: not recognized right away as XML.
    # So we need to decode some of the things like < " etc.
    my $the_response = $response->content;
    $the_response =~ s/</'; '<' /eg;          # convert <
    $the_response =~ s/>/'; '>' /eg;          # convert >
    $the_response =~ s/"/'; "' /eg;           # convert "

    print $the_response;
}
else {
    print $response->error_as_HTML;
}
```

(Ex 2) flickr.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:s="http://www.w3.org/2003/05/soap-envelope"
  xmlns:x="urn:flickr" >
  <xsl:template match="/">
    <html>
      <head><title> Flickr test</title></head>

      <body>
        <div id="planet">
          <!-- Define var for number of matches. @total gets the 'total' attribute of <photos> -->
          <xsl:variable name="var_total" select="/s:Envelope/s:Body/x:FlickrResponse/photos/@total" />

          <p>There were <xsl:value-of select="$var_total" /> results. Here are just some: </p>
          <ul>
            <xsl:apply-templates select="/s:Envelope/s:Body/x:FlickrResponse/photos/photo" />
          </ul>
        </div> </body>
      </html>
    </xsl:template>
    <xsl:template match="photo">
      <!-- Create a variable for the image url.
           A whole URL is like: http://farm1.staticflickr.com/2/1418878_1e92283336.jpg -->
      <xsl:variable name="url">http://farm<xsl:value-of select="@farm"
/>.staticflickr.com/<xsl:value-of select="@server" />/<xsl:value-of select="@id" />_<xsl:value-of
select="@secret" />.jpg</xsl:variable>

      <!-- show the actual image, with a link to it -->
      <p>          <li> <a href="{{$url}}>  </a> </li> </p>
    </xsl:template>
  </xsl:stylesheet>
```

(Ex 2) transform2.js

```
function getPhotos() {  
    var tags = $("#tags").val();  
    var url = "flickr.pl?tags=" + tags;  
    transform("flickr.xsl", url);  
}  
  
function transform (xslFileName, url) {  
    // Get the XSLT file  
    var xslhttp = new XMLHttpRequest();  
    xslhttp.open("GET", xslFileName, false);  
    xslhttp.send('');  
  
    // Get the XML input data  
    var xmlhttp = new XMLHttpRequest();  
    xmlhttp.open("GET", url, false);  
    xmlhttp.send('');  
  
    // Transform the XML via the XSLT  
    var processor = new XSLTProcessor();  
    processor.importStylesheet(xslhttp.responseXML);  
    var newDocument = processor.transformToDocument(xmlhttp.responseXML);  
  
    // Replace part of original document with the new content  
    var o = document.getElementById("planet");  
    var n = newDocument.getElementById("planet");  
    o.parentNode.replaceChild(n, o);  
}
```