XML Processing and Web Services

Chapter 17

Randy Connolly and Ricardo Hoar

Section 1 of 7

XML OVERVIEW

XML Overview

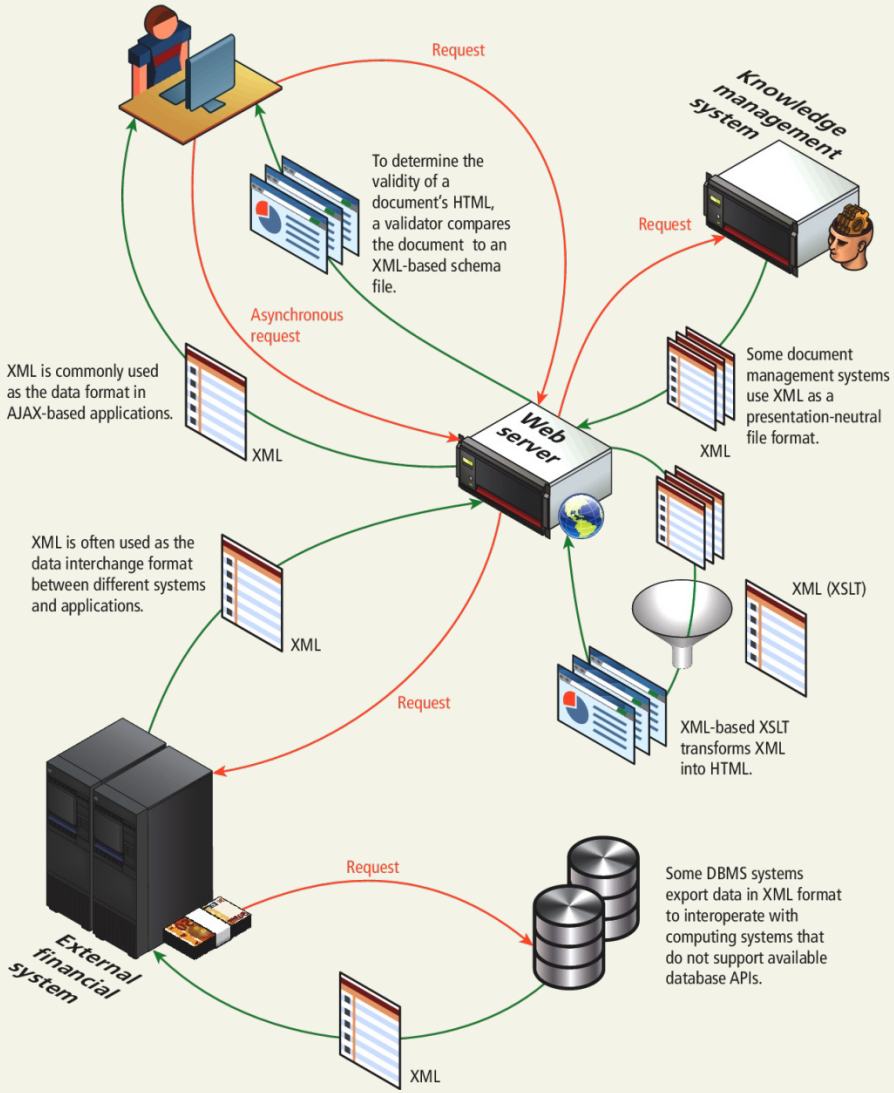
XML is a markup language, but unlike HTML, XML can be used to mark up any type of data.

One of the key benefits of XML data is that as plain text, it can be read and transferred between applications and different operating systems as well as being human-readable and understandable as well.

XML is used as a data interchange format for moving information between systems

XML Overview

Used in many systems



Well Formed XML

For a document to be **well-formed XML**, it must follow the syntax rules for XML:

- Element names are composed of any of the valid characters (most punctuation symbols and spaces are not allowed) in XML.
- Element names can't start with a number.
- There must be a single-root element. A **root element** is one that contains all the other elements; for instance, in an HTML document, the root element is <html>.
- All elements must have a closing element (or be self-closing).
- Elements must be properly nested.
- Elements can contain attributes.
- Attribute values must always be within quotes.
- Element and attribute names are case sensitive.

Well Formed XML

Sample Document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<art>
  <painting id="290">
    <title>Balcony</title>
    <artist>
      <name>Manet</name>
      <nationality>France</nationality>
    </artist>
    <year>1868</year>
    <medium>Oil on canvas</medium>
  </painting>
  <painting id="192">
    <title>The Kiss</title>
    <artist>
      <name>Klimt</name>
      <nationality>Austria</nationality>
    </artist>
    <year>1907</year>
    <medium>Oil and gold on canvas</medium>
  </painting>
  <painting id="139">
    <title>The Oath of the Horatii</title>
    <artist>
      <name>David</name>
      <nationality>France</nationality>
    </artist>
    <year>1784</year>
    <medium>Oil on canvas</medium>
  </painting>
</art>
```

LISTING 17.1 Sample XML document

Valid XML

Requires a DTD

A **valid XML** document is one that is well formed and whose element and content conform to the rules of either its document type definition (DTD) or its schema.

A DTD tells the XML parser which elements and attributes to expect in the document as well as the order and nesting of those elements.

A DTD can be defined within an XML document or within an external file.

Data Type Definition

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE art [
<!ELEMENT art (painting*)>
<!ELEMENT painting (title,artist,year,medium)>
<!ATTLIST painting id CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT artist (name,nationality)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT nationality (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT medium (#PCDATA)>
]>
<art>
...
</art>
```

LISTING 17.2 Example DTD

Data Type Definition

Example

The main drawback with DTDs is that they can only validate the existence and ordering of elements. They provide no way to validate the values of attributes or the textual content of elements.

For this type of validation, one must instead use XML schemas, which have the added advantage of using XML syntax. Unfortunately, schemas have the corresponding disadvantage of being long-winded and harder for humans to read and comprehend; for this reason, they are typically created with tools.

XML Schema

Just one example

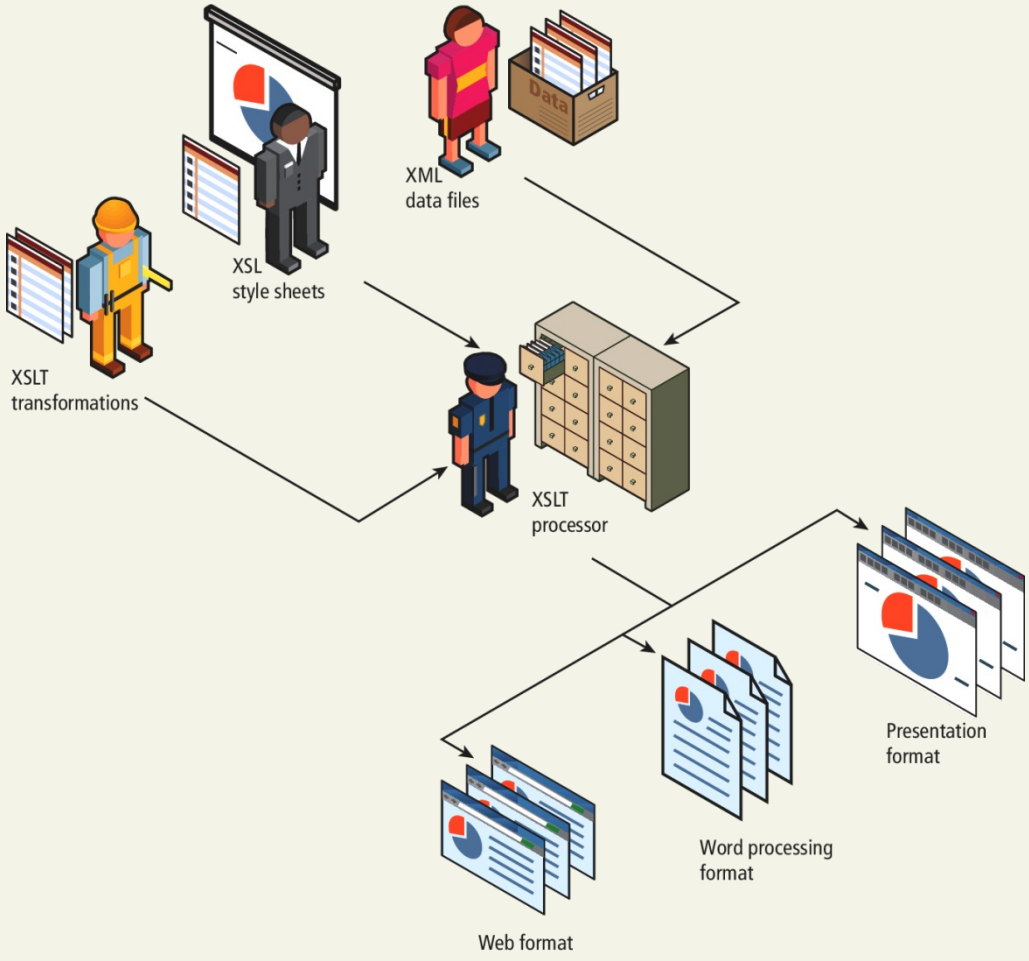
```
<xs:schema attributeFormDefault="unqualified"
     elementFormDefault="qualified"
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="art">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="painting" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="title"/>
              <xs:element name="artist">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="name"/>
                    <xs:element type="xs:string" name="nationality"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element type="xs:short" name="year" />
              <xs:element type="xs:string" name="medium"/>
            </xs:sequence>
            <xs:attribute type="xs:short" name="id" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

LISTING 17.3 Example schema

XSLT

XML Stylesheet Transformations

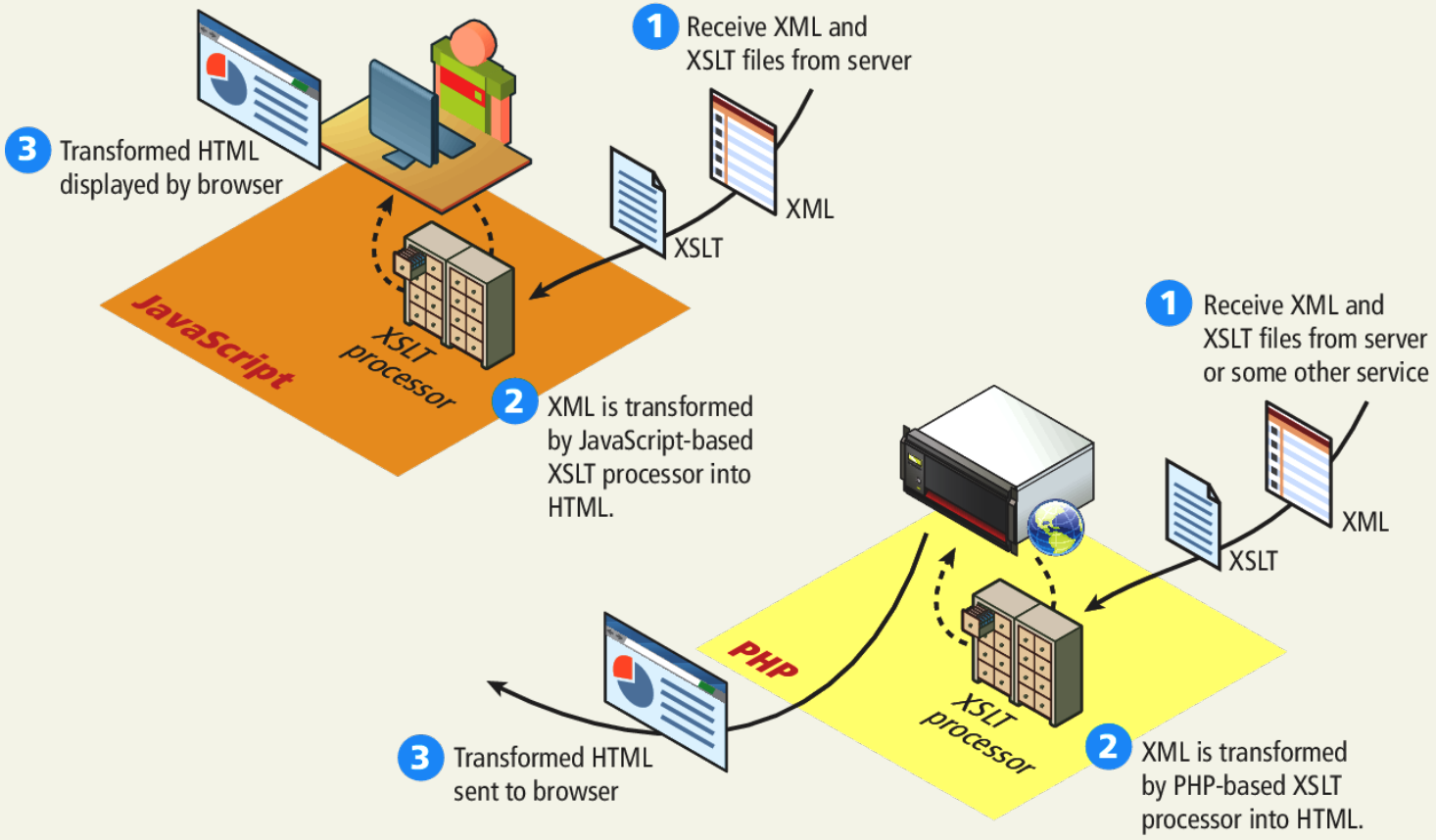
XSLT is an XML-based programming language that is used for transforming XML into other document formats



XSLT

Another usage

XSLT is also used on the server side and within JavaScript



XSLT

Example XSLT document that converts the XML from Listing 17.1 into an HTML list

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xsl:version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns="http://www.w3.org/1999/xhtml">
<body>
   <h1>Catalog</h1>
   <ul>
    <xsl:for-each select="/art/painting">
      <li>
        <h2><xsl:value-of select="title"/></h2>
        <p>By: <xsl:value-of select="artist/name"/><br/>
           Year: <xsl:value-of select="year"/>
           [<xsl:value-of select="medium"/>]</p>
      </li>
    </xsl:for-each>
   </ul>
</body>
</html>
```

LISTING 17.4 An example XSLT document

XSLT

An XML parser is still needed to perform the actual transformation


```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<h1>Catalog</h1>
<ul>
  <li>
    <h2>Balcony</h2>
    <p>By: Manet<br/>
    Year: 1868 [Oil on canvas]</p>
  </li>
  <li>
    <h2>The Kiss</h2>
    <p>By: Klimt<br/>
    Year: 1907 [Oil and gold on canvas]</p>
  </li>
  <li>
    <h2>The Oath of the Horatii</h2>
    <p>By: David<br/>Year: 1784 [Oil on canvas]</p>
  </li>
</ul>
</body>
</html>
```

XPath

Another XML Technology

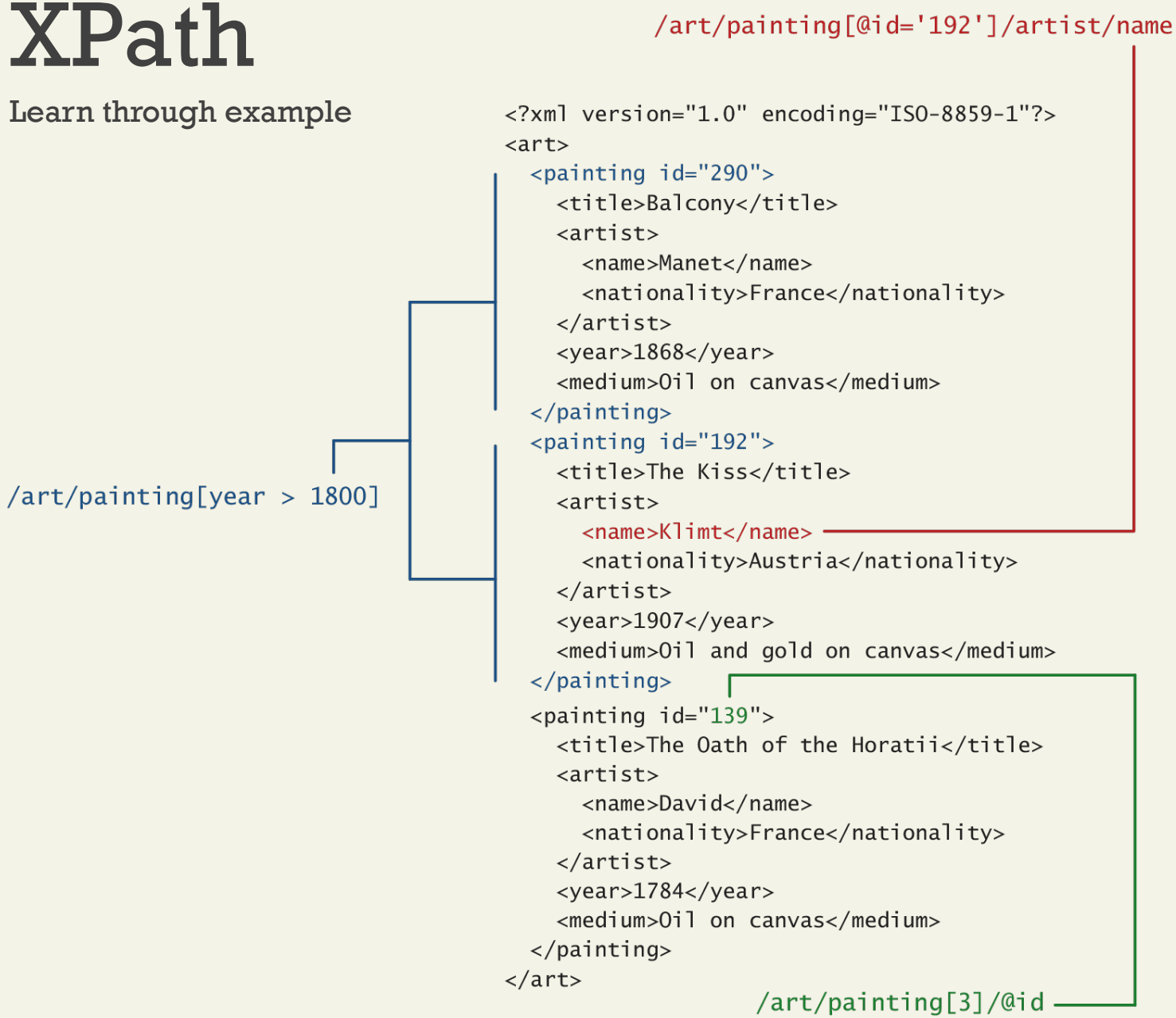
XPath is a standardized syntax for searching an XML document and for navigating to elements within the XML document

XPath is typically used as part of the programmatic manipulation of an XML document in PHP and other languages

XPath uses a syntax that is similar to the one used in most operating systems to access directories.

XPath

Learn through example



Section 2 of 7

XML PROCESSING

XML Processing

Two types

XML processing in PHP, JavaScript, and other modern development environments is divided into two basic styles:

- The **in-memory approach**, which involves reading the entire XML file into memory into some type of data structure with functions for accessing and manipulating the data.
- The **event or pull approach**, which lets you pull in just a few elements or lines at a time, thereby avoiding the memory load of large XML files.

XML Processing

In JavaScript

All modern browsers have a built-in XML parser and their JavaScript implementations support an in-memory XML DOM API.

You can use the already familiar DOM functions such as getElementById(), getElementsByTagName(), and createElement() to access and manipulate the data.

XML Processing

```
<script>
if (window.XMLHttpRequest)  {
  // code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
}
else  {
  // code for old versions of IE (optional you might just decide to
  // ignore these)
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
// load the external XML file
xmlhttp.open("GET","art.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

// now extract a node list of all <painting> elements
paintings = xmlDoc.getElementsByTagName("painting");
if (paintings) {
   // loop through each painting element
   for (var i = 0; i < paintings.length; i++)
   {
      // display its id attribute
      alert("id="+paintings[i].getAttribute("id"));

      // find its <title> element
      title = paintings[i].getElementsByTagName("title");
      if (title) {
         // display the text content of the <title> element
         alert("title="+title[0].textContent);
      }
   }
}
</script>
```

LISTING 17.5 Loading and processing an XML document via JavaScript

XML Processing

With PHP

PHP provides several extensions or APIs for working with XML including:

- The **SimpleXML** extension which loads the data into an object that allows the developer to access the data via array properties and modifying the data via methods.
- The **XMLReader** is a read-only pull-type extension that uses a cursor-like approach similar to that used with database processing

XML Processing

With PHP using Simple XML

```php
<?php

$filename = 'art.xml';
if (file_exists($filename)) {
   $art = simplexml_load_file($filename);

   // access a single element
   $painting = $art->painting[0];
   echo '<h2>' . $painting->title . '</h2>';
   echo '<p>By ' . $painting->artist->name . '</p>';
   // display id attribute
   echo '<p>id=' . $painting["id"] . '</p>';

   // loop through all the paintings
   echo "<ul>";
   foreach ($art->painting as $p)
   {
      echo '<li>' . $p->title . '</li>';
   }
   echo '</ul>';
} else {
   exit('Failed to open ' . $filename);
}

?>
```

Variable and attribute names taken from xml

LISTING 17.8 Using simple XML

XML Processing

With PHP using Simple XML and XPath

```
$art = simplexml_load_file($filename);

$titles = $art->xpath('/art/painting/title');
foreach ($titles as $t) {
   echo $t . '<br/>';
}

$names = $art->xpath('/art/painting[year>1800]/artist/name');
foreach ($names as $n) {
   echo $n . '<br/>';
}
```

LISTING 17.9 Using XPath with SimpleXML

XML Processing

With PHP using XMLReader

```
$filename = 'art.xml';
if (file_exists($filename)) {

   // create and open the reader
   $reader = new XMLReader();
   $reader->open($filename);

   // loop through the XML file
   while ( $reader->read() ) {
     $nodeName = $reader->name;

     // since all sorts of different XML nodes we must check
     // node type
     if ($reader->nodeType == XMLREADER::ELEMENT
         && $nodeName == 'painting') {
         $id = $reader->getAttribute('id');
         echo '<p>id=' . $id . '</p>';
     }

     if ($reader->nodeType == XMLREADER::ELEMENT
         && $nodeName =='title') {
         // read the next node to get at the text node
         $reader->read();
         echo '<p>' . $reader->value . '</p>';
     }
   }
} else {
   exit('Failed to open ' . $filename);
}
```

Less "automatic"

More Verbose

LISTING 17.10 Using XMLReader

XML Processing

Why choose when you can use both

```
// create and open the reader
$reader = new XMLReader();
$reader->open($filename);

// loop through the XML file
while($reader->read()) {
  $nodeName = $reader->name;
  if ($reader->nodeType == XMLREADER::ELEMENT
      && $nodeName =='painting') {
      // create a SimpleXML object from the current painting node
      $doc = new DOMDocument('1.0', 'UTF-8');
      $painting = simplexml_import_dom($doc->importNode
                  ($reader- >expand(),true));
      // now have a single painting as an object so can output it
      echo '<h2>' . $painting->title . '</h2>';
      echo '<p>By ' . $painting->artist->name . '</p>';
  }
}
```

LISTING 17.11 Combining XMLReader and SimpleXML

Section 3 of 7

JSON

JSON

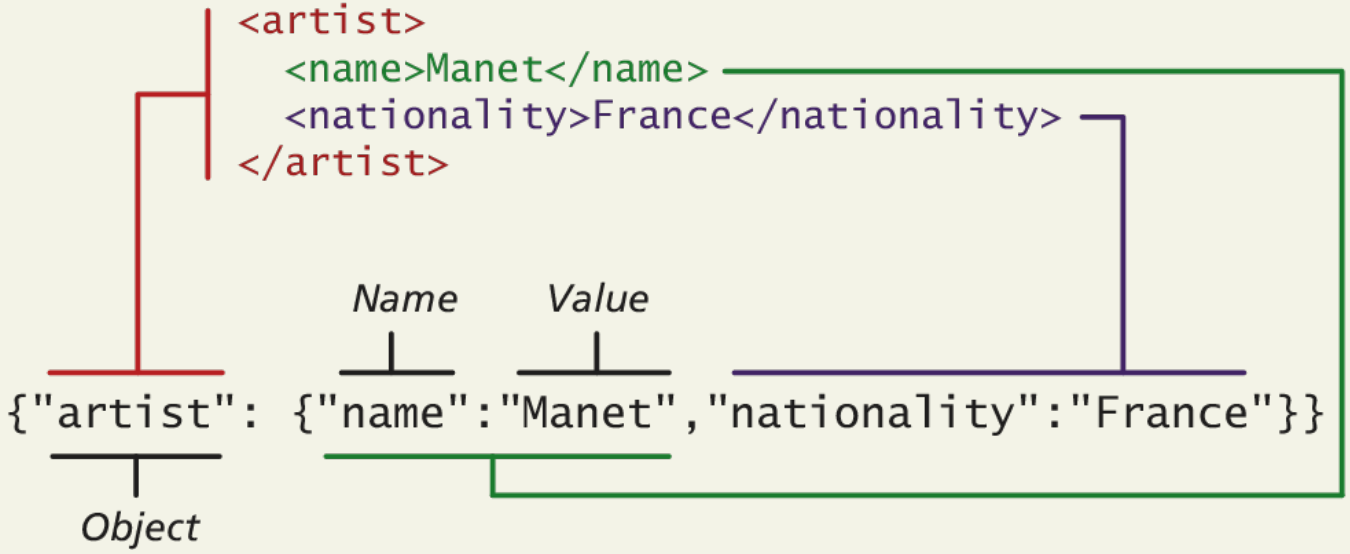
JSON stands for JavaScript Object Notation (though its use is not limited to JavaScript)

Like XML, JSON is a data serialization format. It provides a more concise format than XML.

Many REST web services encode their returned data in the JSON data format instead of XML.

JSON

An example XML object in JSON



JSON

An example XML object in JSON

```
{
   "paintings": [
      {
         "id":290,
         "title":"Balcony",
         "artist":{
            "name":"Manet",
            "nationality":"France"
         },
         "year":1868,
         "medium":"Oil on canvas"
      },
      {
         "id":192,
         "title":"The Kiss",
         "artist":{
            "name":"Klimt",
            "nationality":"Austria"
         },
         "year":1907,
         "medium":"Oil and gold on canvas"
      },
      {
         "id":139,
         "title":"The Oath of the Horatii",
         "artist":{
            "name":"David",
            "nationality":"France"
         },
         "year":1784,
         "medium":"Oil on canvas"
      }
   ]
}
```

LISTING 17.12 JSON representation of XML data from Listing 17.1

Using JSON in JavaScript

Creating JSON JavaScript objects

it is easy to make use of the JSON format in JavaScript:

```
var a = {"artist": {"name":"Manet","nationality":"France"}};

alert(a.artist.name + " " + a.artist.nationality);
```

When the JSON information will be contained within a string (say when downloading) the JSON.parse() function can be used to transform the string containing into a JavaScript object

Using JSON in JavaScript

Convert string to JSON object and vice versa

```
var text = '{"artist": {"name":"Manet","nationality":"France"}}';

var a = JSON.parse(text);

alert(a.artist.nationality);
```

JavaScript also provides a mechanism to translate a JavaScript object into a JSON string:

```
var text = JSON.stringify(artist);
```

Using JSON in PHP

JSON on the server

Converting a JSON string into a PHP object is quite straightforward:

```
<?php
  // convert JSON string into PHP object
  $text = '{"artist": {"name":"Manet","nationality":"France"}}';
  $anObject = json_decode($text);
  echo $anObject->artist->nationality;

  // convert JSON string into PHP associative array
  $anArray = json_decode($text, true);
  echo $anArray['artist']['nationality'];
?>
```

Notice that the **json_decode()** function can return either a PHP object or anassociative array.

Using JSON in PHP

Since JSON data is often coming from an external source, always check for parse errors before using it, via the **json_last_error()** function.

```php
<?php
  // convert JSON string into PHP object
  $text = '{"artist": {"name":"Manet","nationality":"France"}}';
  $anObject = json_decode($text);
  // check for parse errors
  if (json_last_error() == JSON_ERROR_NONE) {
    echo $anObject->artist->nationality;
  }
?>
```

Using JSON in PHP

JSON on the server

To go the other direction (i.e., to convert a PHP object into a JSON string), you can use the **json_encode()** function.

```
// convert PHP object into a JSON string
$text = json_encode($anObject);
```

Section 4 of 7

OVERVIEW OF WEB SERVICES

Web Services

An overview

Web services are the most common example of a computing paradigm commonly referred to as **service-oriented computing** (SOC).

A **service** is a piece of software with a platform-independent interface that can be dynamically located and invoked.

Web services are a relatively standardized mechanism by which one software application can connect to and communicate with another software application using web protocols.

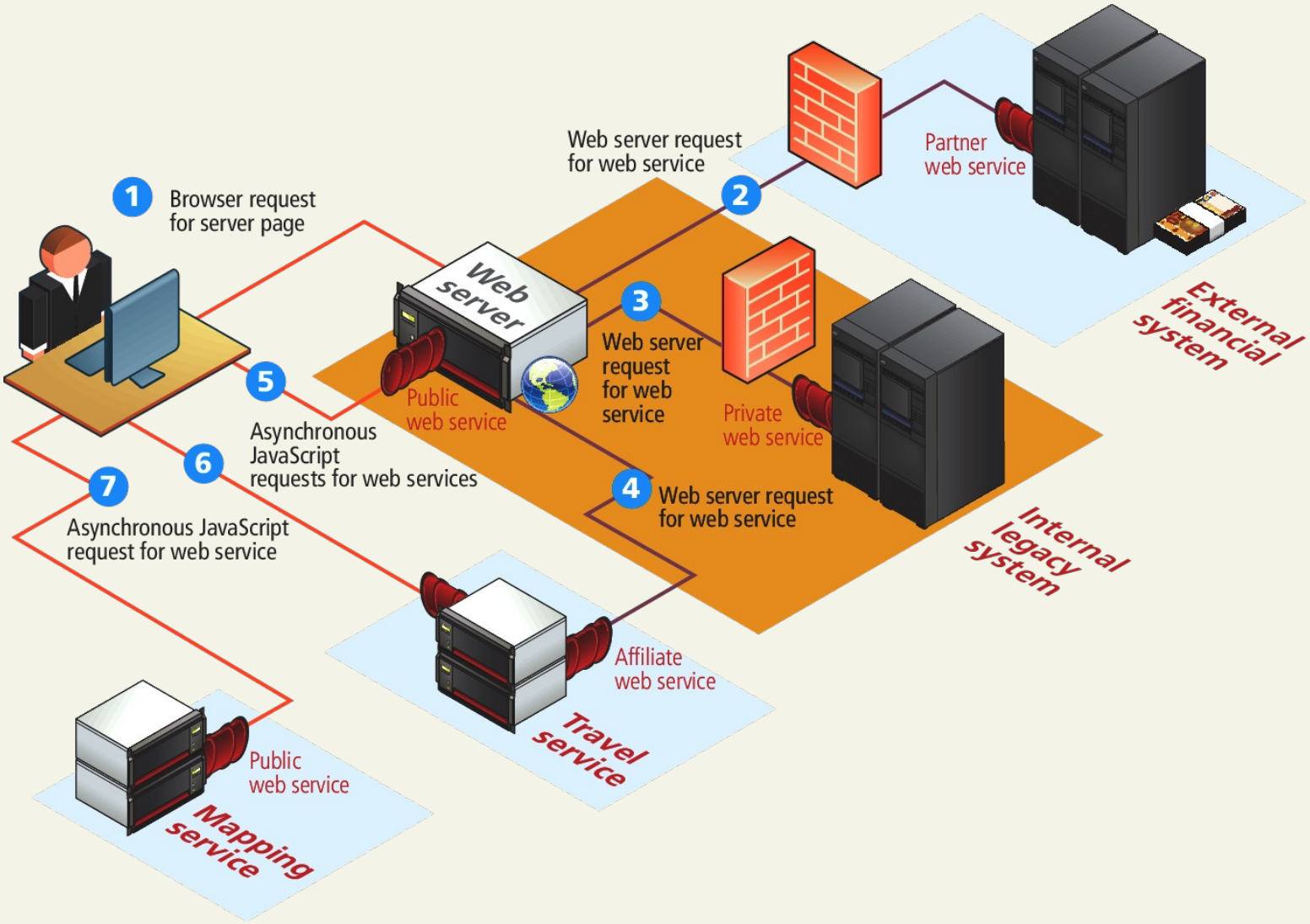
Web Services

Benefits

- They potentially provide interoperability between different software applications running on different platforms
- They can be used to implement something called a **service-oriented architecture** (SOA)
- They can be offered by different systems within an organization as well as by different organizations

Web Services

Visual Overview



Web Services

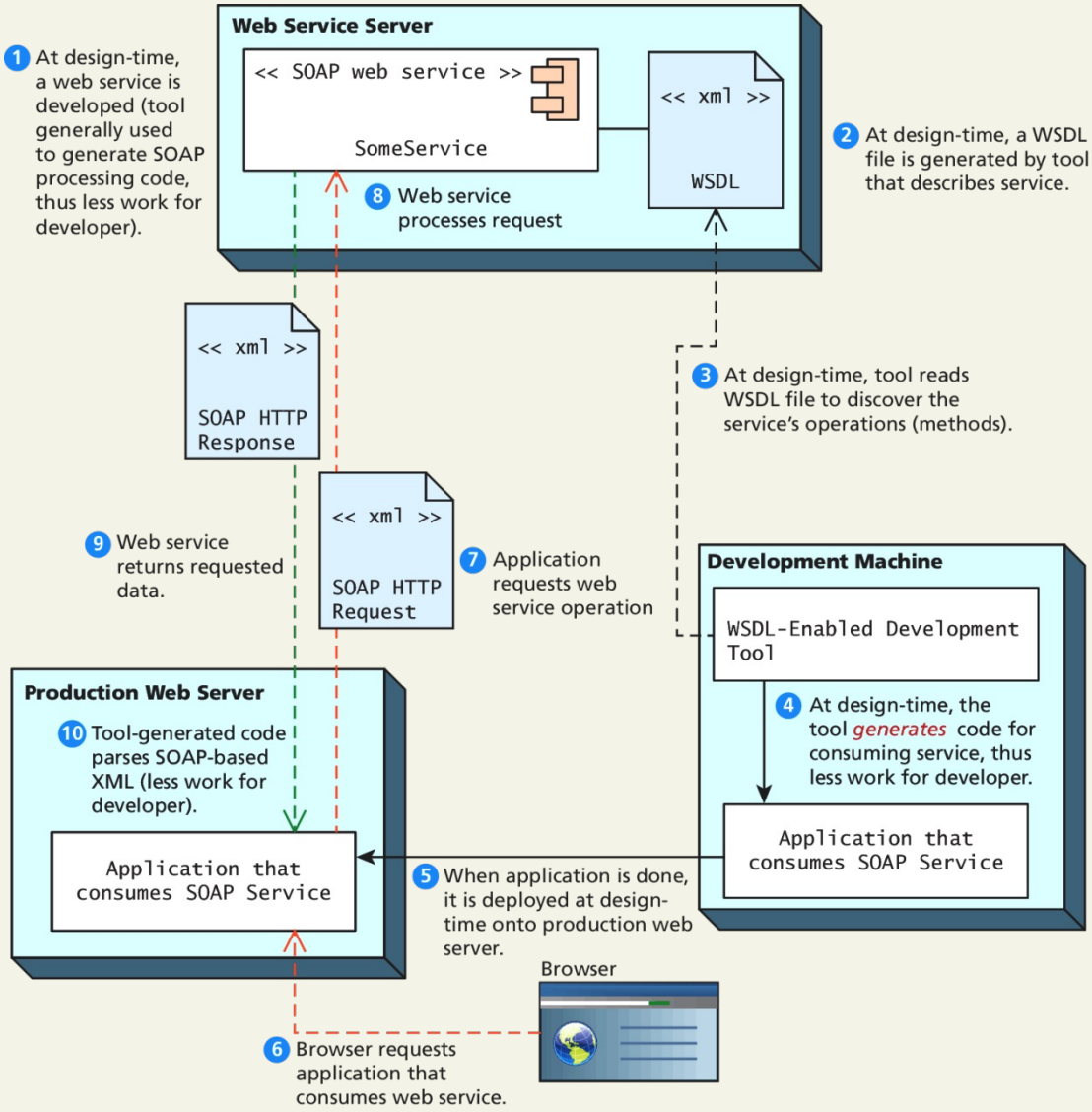
SOAP Services

SOAP is the message protocol used to encode the service invocations and their return values via XML within the HTTP header.

- SOAP and WSDL are complex XML schemas
- akin to using a compiler: its output may be complicated to understand

Web Services

SOAP Services



Web Services

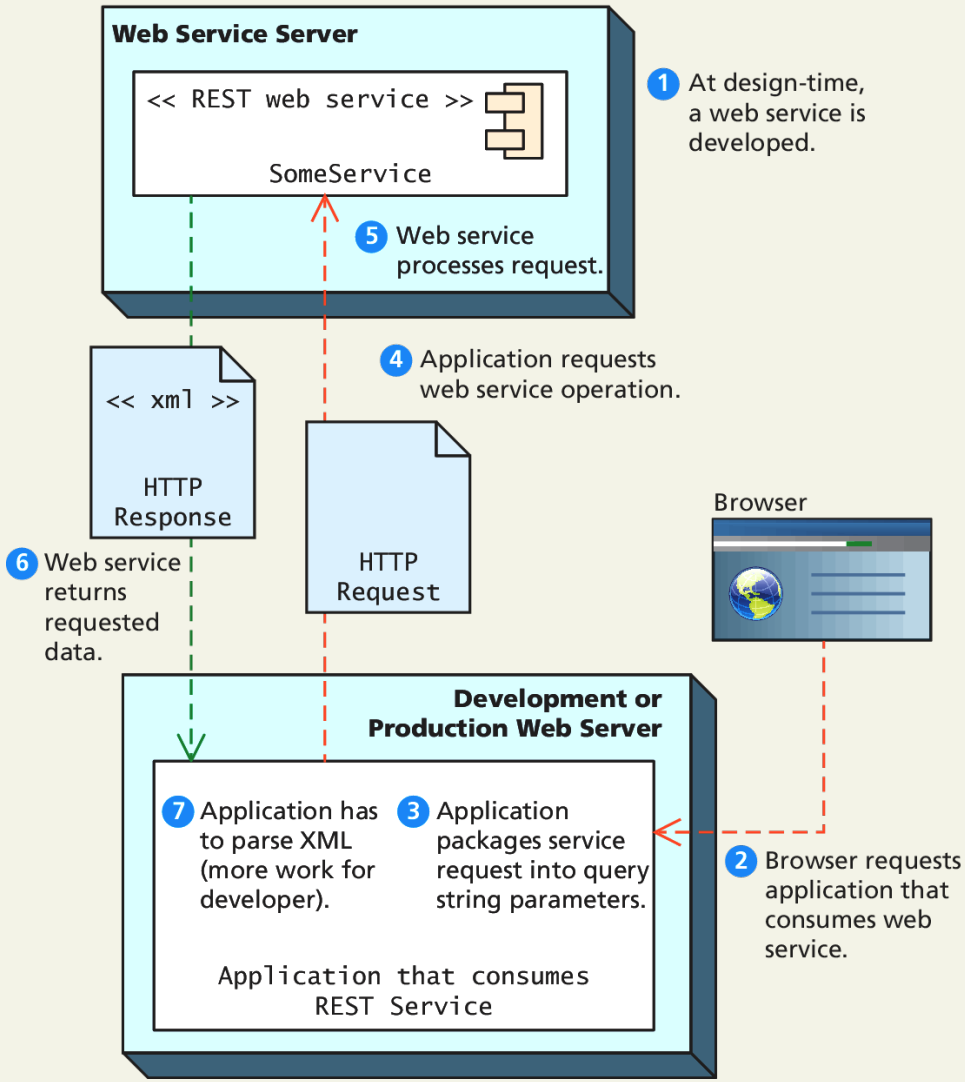
REST Services

REST stands for Representational State Transfer.

- RESTful web service does away with the service description layer, and needs no separate protocol for encoding message requests and responses.
- It simply uses HTTP URLs for requesting a resource/object (and for encoding input parameters).
- The serialized representation of this object, usually an XML or JSON stream, is then returned to the requestor as a normal HTTP response.

Web Services

REST Services



An Example Web Service

We will only use REST from here on in

Consider the Google Geocoding API.

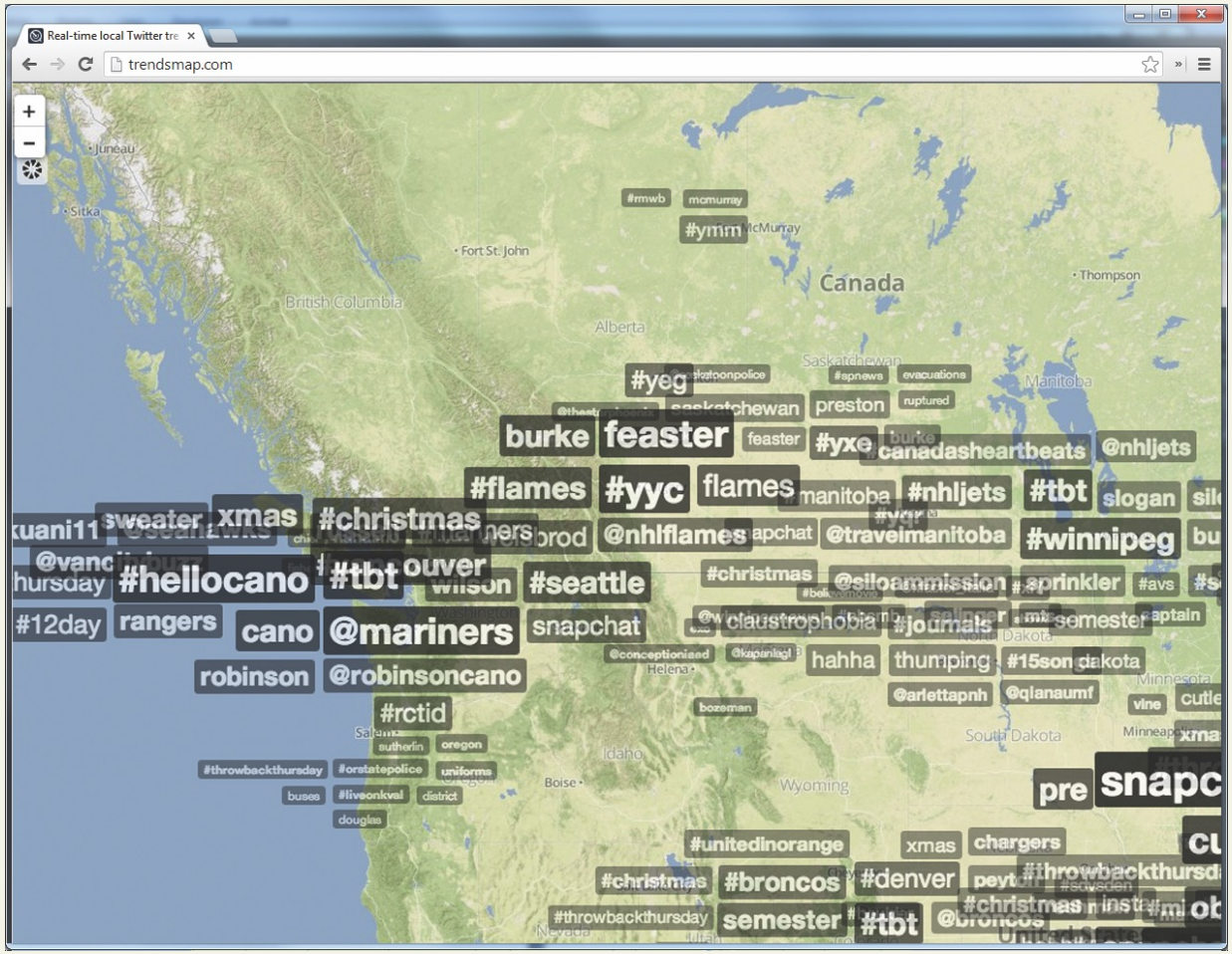
The Google Geocoding API provides a way to perform geocoding operations via an HTTP GET request, and thus is an especially useful example of a RESTful web service.

Geocoding typically refers to the process of turning a real-world address into geographic coordinates, which are usually latitude and longitude values

Reverse geocoding is the process of converting geographic coordinates into a human-readable address.

An Example Web Service

Mashups abound with web services



From trendsmap.com

An Example Web Service

More details

In this case the request will take the following form:

http://maps.googleapis.com/maps/api/geocode/xml?***address***

An example geocode request would look like the following:

http://maps.googleapis.com/maps/api/geocode/xml?address=British%20Museum,+Great+Russell+Street,+London,+WC1B+3DG

From trendsmap.com

An Example Web Service

The Response

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8
Date: Fri, 19 Jul 2013 19:15:54 GMT
Expires: Sat, 20 Jul 2013 19:15:54 GMT
Cache-Control: public, max-age=86400
Vary: Accept-Language
Content-Encoding: gzip
Server: mafe
Content-Length: 512
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
   <status>OK</status>
   <result>
      <type>route</type>
      <formatted_address>
         Great Russell Street, London Borough of Camden, London, UK
      </formatted_address>
      <address_component>
         <long_name>Great Russell Street</long_name>
         <short_name>Great Russell St</short_name>
         <type>route</type>
      </address_component>
      <address_component>
         <long_name>London</long_name>
         <short_name>London</short_name>
         <type>locality</type>
         <type>political</type>
      </address_component>
      ...
      <geometry>
         <location>
            <lat>51.5179231</lat>
            <lng>-0.1271022</lng>
         </location>
         <location_type>GEOMETRIC_CENTER</location_type>
         ...
      </geometry>
   </result>
</GeocodeResponse>
```

LISTING 17.13 HTTP response from web service

The response is a standard HTTP response with headers

This response is XML

The lat/lng is in there somewhere

Identifying and Authenticating Service Requests

Most web services are not open. Instead, they typically employ one of the following techniques:

- **Identity**. Each web service request must identify who is making the request.
- **Authentication**. Each web service request must provide additional evidence that they are who they say they are.

Identity examples

Real World ways of limiting service

Web services that make use of an API key typically require the user (i.e., the developer) to register online with the service for an API key. This API key is then added to the GET request as a query string parameter.

For instance, to request to the Microsoft Bing Maps web service will look like the following :

http://dev.virtualearth.net/REST/v1/Locations?o=xml&query=British%20Museum,+Great+Russell+Street,+London,+WC1B+3DG,+UK&**key=[*BING API KEY HERE*]**

Authentication

Real World ways of limiting service

Some web services are providing private/proprietary information or are involving financial transactions.

In this case, these services not only may require an API key, but they also require some type of user name and password in order to perform an authorization.

Many of the most well-known web services instead make use of the OAuth standard.

Section 5 of 7

CONSUMING WEB SERVICES IN PHP

Consuming Web Services in PHP

There are three usual approaches in PHP for making a REST request:

- Using the *file_get_contents()* function.
- Using functions contained within the *curl* library.
- Using a custom library for the specific web service. Many of the most popular web services have free and proprietary PHP libraries available.

Consuming Web Services in PHP

The *file_get_contents()* function is simple but doesn't allow POST requests.

Services that require authentication will have to use the *curl* extension library, which allows significantly more control over requests. You may need to configure your server to include curl support.

A Flickr Example

The Flickr web service provides a photo search service. The basic format for this service method is:

http://api.flickr.com/services/rest/method=flickr.photos.search&api_key=[*enter your flickr api key here*]&tags=[*search values here*]&format=rest

The service will return its standard XML photo list

A Flickr Example

Some Code using file_get_contents()

```php
<?php

function constructFlickrSearchRequest($search)
{
   $serviceDomain = 'http://api.flickr.com/services/rest/?';
   $method = 'method=flickr.photos.search';
   $api_key = 'api_key=' . 'your Flickr api key here';
   $searchFor = 'tags=' . $search;
   $format = 'format=rest';
   // only 12 results for now
   $options = 'per_page=12';
   // due to copyright, we will use only the author's Flickr images
   $options .= '&user_id=31790027%40N04';

   return $serviceDomain . $method . '&' . $api_key .'&'
       . $searchFor . '&' . $format . '&' . $options;
}

?>
```

LISTING 17.14 Function to construct Flickr search request

```
$request = constructFlickrSearchRequest('Athens');
$response = file_get_contents($request);
```

A Flickr Example

Use curl (and actually do something)

```
$request = constructFlickrSearchRequest('Athens');
echo '<p><small>' . $request . '</small></p>';

$http = curl_init($request);
// set curl options
curl_setopt($http, CURLOPT_HEADER, false);
curl_setopt($http, CURLOPT_RETURNTRANSFER, true);
// make the request
$response = curl_exec($http);
// get the status code
$status_code = curl_getinfo($http, CURLINFO_HTTP_CODE);
// close the curl session
curl_close($http);

if ($status_code == 200) {
   // create simpleXML object by loading string
   $xml = simplexml_load_string($response);
   // iterate through each <photo> element
   foreach ($xml->photos->photo as $p) {
      // construct URLs for image and for link
      $pageURL = "http://www.flickr.com/photos/" . $p['owner'] . "/"
                 .$p['id'];
      $imgURL = "http://farm" .$p["farm"] . ".staticflickr.com/"
        . $p["server"] . "/" . $p["id"] . "_" . $p["secret"] . "_q.jpg";
      // output links and image tags
      echo "<a href='" . $pageURL . "'>";
      echo "<img src='" . $imgURL . "' />";
      echo "</a>";
}
}
else {
   die("Your call to web service failed -- code=" . $status_code);
}
```

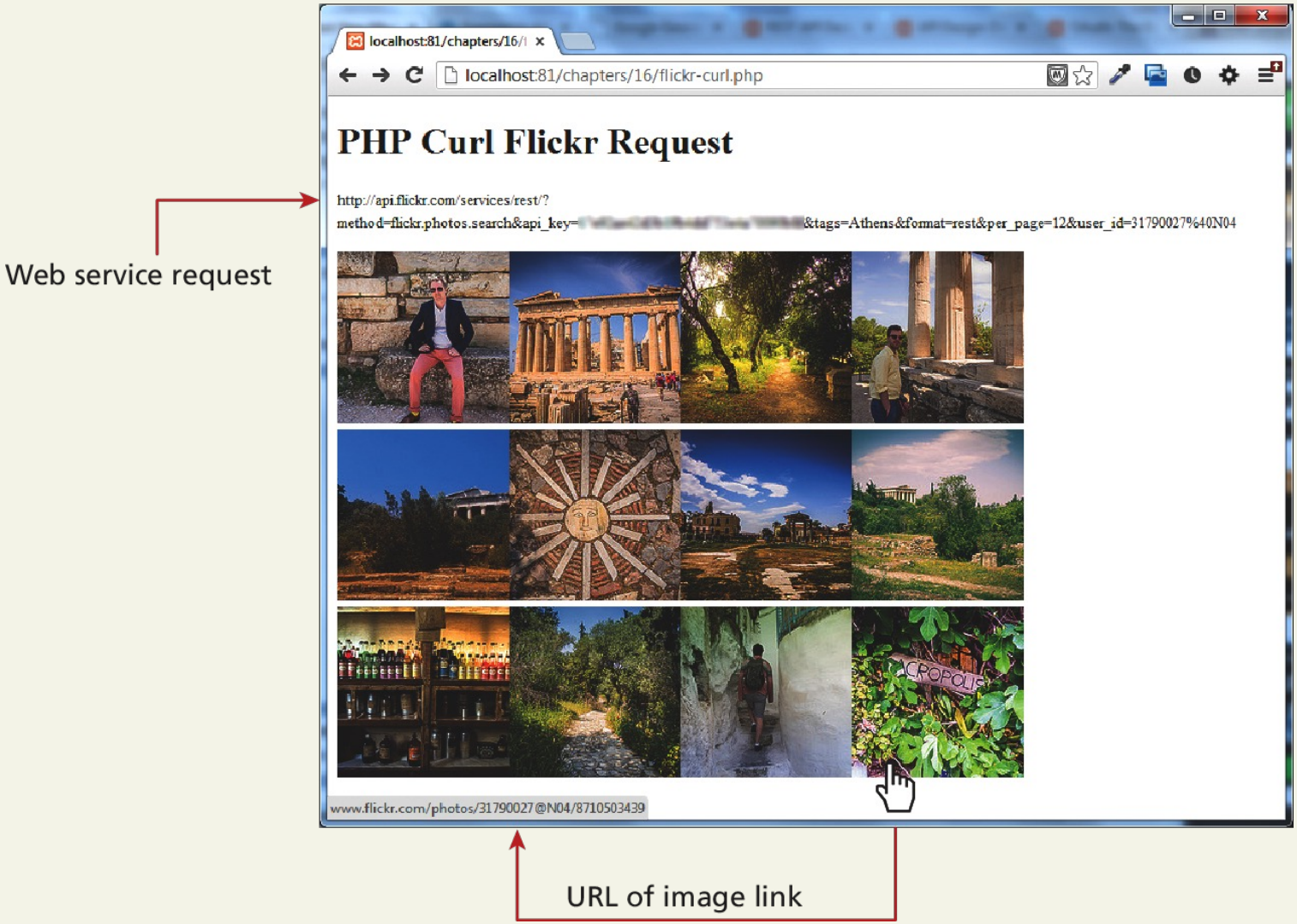
Make the request

Parse the XML

LISTING 17.15 Querying web service and processing the results

A Flickr Example

What that last code actually built



Consuming JSON Services

Consuming a JSON web service requires almost the same type of PHP coding as consuming an XML web service.

But rather than using *SimpleXML* to extract the information one needs, one instead uses the **json_decode()** function.

Consuming JSON Services

Combine 2 services (using JSON)

To extract the latitude and longitude from the JSON string returned from the mapping web service, you would need code similar to the following:

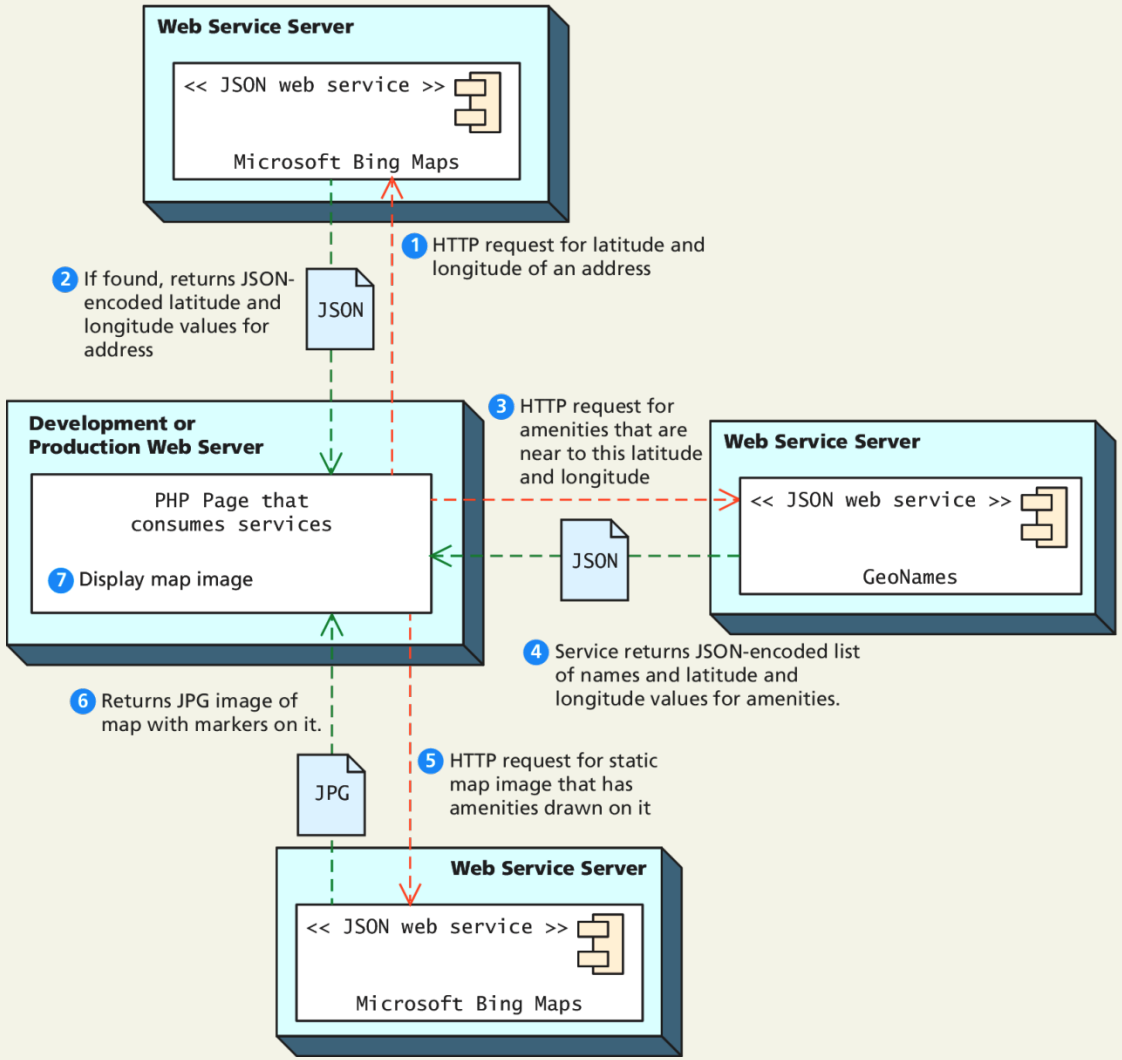
```
// decode JSON and extract latitude and longitude
$json = json_decode($response);
if (json_last_error() == JSON_ERROR_NONE) {
    $lat = $json->resourceSets[0]->resources[0]->point
         ->coordinates[0];
    $long = $json->resourceSets[0]->resources[0]->point
          ->coordinates[1];
}
```

Once our program has retrieved the latitude and longitude, the program then will use the *GeoNames* web service's. This request will take the following form:

http://api.geonames.org/findNearbyPOIsOSMJSON?lat=43.6520004&lng=-79.4082336&username=*your-username-here*

Consuming JSON Services

A complicated example with 2 services



Consuming JSON Services

More examples

URL of service request for static road map image

Zoom level (between 1 and 21)

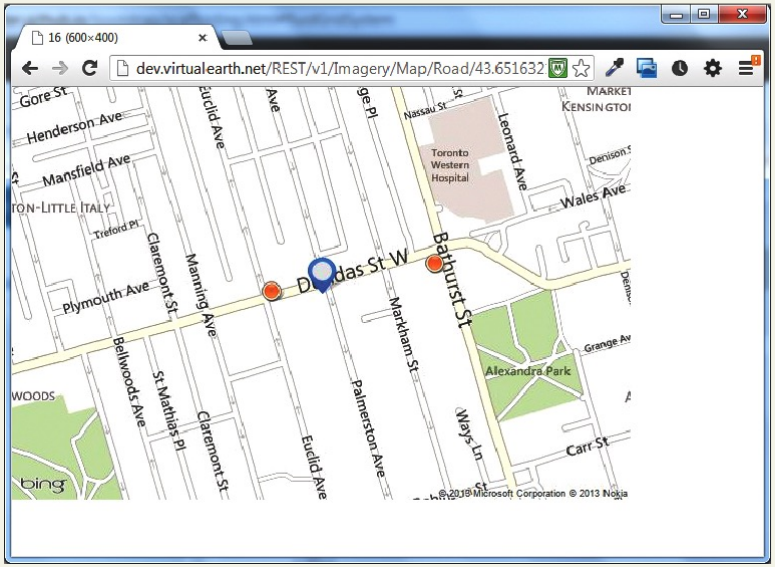
```
http://dev.virtualearth.net/REST/v1/Imagery/Map/Road/43.6516321,-79.4085317/16?
key=[your api key]
&mapSize=600,400
&pp=43.6516321,-79.4085317;66
&pp=43.6520854,-79.4061892;34
&pp=43.6516601,-79.4095859;34
```

Width and height of map in pixels

Location (latitude and longitude) of center of map

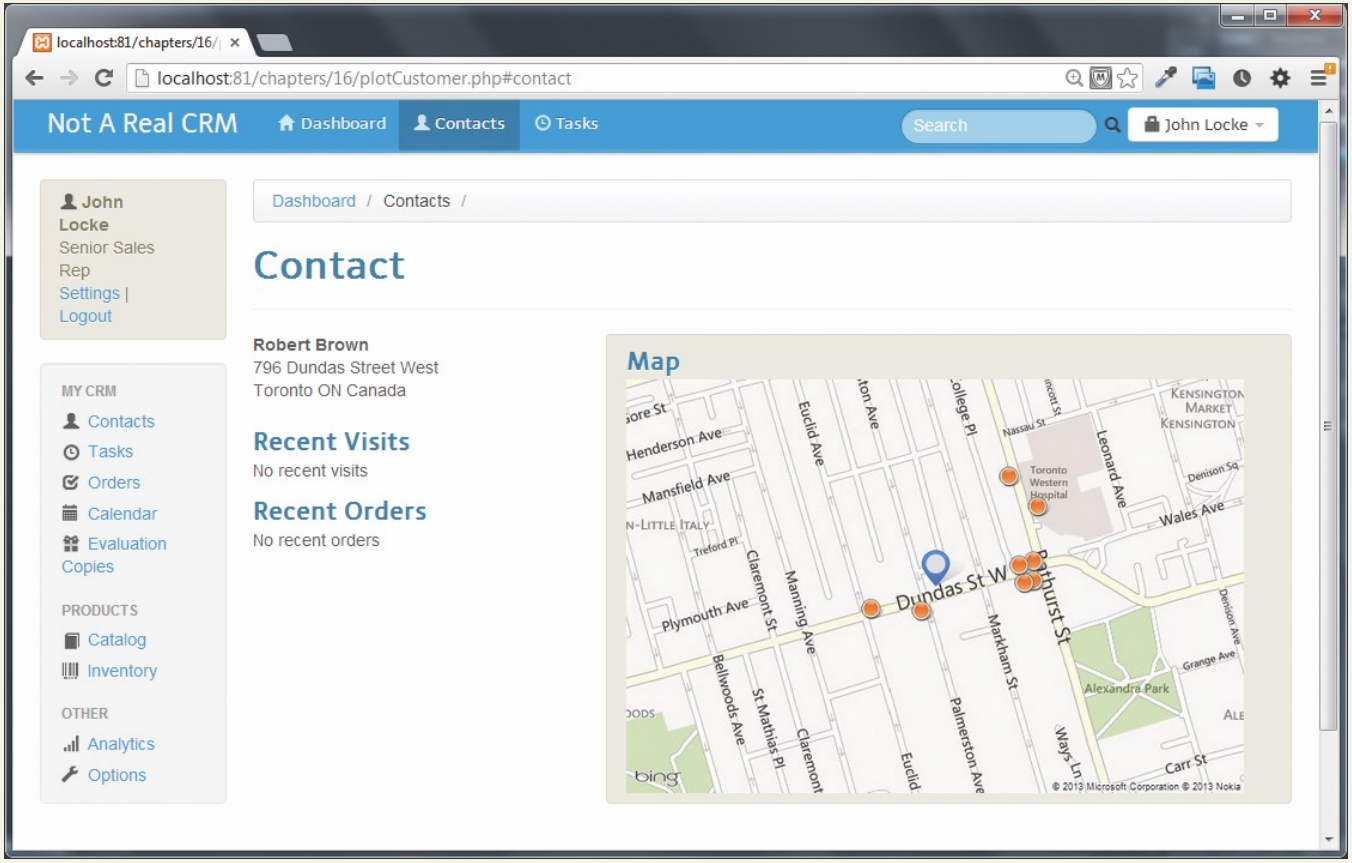
Location of marker (marker 66 = blue circle)

Location of other markers (amenities) with marker 34 = orange circle



Consuming JSON Services

More examples



Section 6 of 7

CREATING WEB SERVICES

Creating Your Own Services

Web Services that is

Since REST services simply respond to HTTP requests, creating a PHP web service is only a matter of creating a page that responds to query string parameters and instead of returning HTML, it returns XML or JSON.

Our PHP page must also modify the *Content-type* header

It is important to recognize that not all web services are intended to be used by external clients. Many web services are intended to be consumed asynchronously by their own web pages via JavaScript

Creating an XML/JSON Service

Web Service, that is

The first service we will create will be one that returns data from our Book Customer Relations Management database.

To begin, we should determine the methods our service will support and the format of the requests.

crmServiceSearchBooks.php?**criteria**=yyy&**look**=zzz

- The **criteria** parameter will be used to specify what type of criteria we will use for the book search. This exercise will only support four values: imprint, category, look, and subcategory.
- The **look** parameter will be used to specify the actual value to search.

Creating an XML/JSON Service

Web Service, that is

For instance, if we had the following request:

crmServiceSearchBooks.php?criteria=subcategory&look=finance

It would be equivalent to the SQL search:

SELECT * FROM Books WHERE SubCategoryID=5

Creating a JSON Service

Web Service

Creating a JSON web service:

- creating a JSON representation of an object
- setting the *Content-type* header to indicate the content will be JSON,
- and then outputting the JSON object

PHP *json_encode()* function does most of the work for us

Creating a JSON Service

Web Service

```php
<?php
require_once('includes/setup.inc.php');
require_once('includes/funcFindTitles.inc.php');

// Tell the browser to expect JSON rather than HTML
header('Content-type: application/json');

if ( isCorrectQueryStringInfo() ) {
   outputJSON($dbAdapter);
}
else {
   // put error message in JSON format
   echo '{"error": {"message":"Incorrect query string values"}}';
}

function outputJSON($dbAdapter) {
   // get query string values and set up search criteria
   $whereClause = 'Title Like ?';
   $look = $_GET['term'] . '%';

   // get the data from the database
   $bookGate = new BookTableGateway($dbAdapter);
   $results = $bookGate->findByFromJoins($whereClause, Array($look) );

   // output the JSON for the retrieved book data
   echo json_encode($results);

   $dbAdapter->closeConnection();
}
```

LISTING 17.24 JSON crmServiceFindTitleMatches service

Output headers

Function to create JSON From the Database, based on query

Creating a JSON Service

Web Service

For this function to work, the class of the custom object being converted must provide its own implementation of the *JsonSerializable* interface.

This interface contains only the single method *jsonSerialize()*.

In this web service, we are outputting JSON for objects of the *Book* class, so this class will need to implement this method

Creating a JSON Service

Web Service

```
class Book extends DomainObject implements JsonSerializable
{
    ...
    /*
     This method is called by the json_encode() function that is
     part of PHP
    */
    public function jsonSerialize() {
       return ['id' => $this->ID, 'value' => $this->Title];
    }
}
```

LISTING 17.25 Adding jsonSerializable() to Book class

Creating an JSON Service

Testing our service in the browser