
Severance, Charles

http://hdl.handle.net/2027.42/64959
Unless otherwise noted, the content of this course material is licensed under a Creative Commons Attribution 3.0 License. http://creativecommons.org/licenses/by/3.0/.

Copyright © 2009, Charles Severance.

You assume all responsibility for use and potential liability associated with any use of the material. Material contains copyrighted content, used in accordance with U.S. law. Copyright holders of content included in this material should contact open.michigan@umich.edu with any questions, corrections, or clarifications regarding the use of content. The Regents of the University of Michigan do not license the use of third party content posted to this site unless such a license is specifically granted in connection with particular content. Users of content are responsible for their compliance with applicable law. Mention of specific products in this material solely represents the opinion of the speaker and does not represent an endorsement by the University of Michigan. For more information about how to cite these materials visit http://michigan.educommons.net/about/terms-of-use.

Any medical information in this material is intended to inform and educate and is not a tool for self-diagnosis or a replacement for medical evaluation, advice, diagnosis or treatment by a healthcare professional. You should speak to your physician or make an appointment to be seen if you have questions or concerns about this information or your medical condition. Viewer discretion is advised: Material may contain medical images that may be disturbing to some viewers.
Web Services and Application Programming Interfaces

SI539 - Charles Severance
Service Oriented Approach

Service Oriented Approach

- Most non-trivial web applications are service oriented
- They use services from other applications
- Credit Card Charge
- Hotel Reservation systems
- Twitter

(Gears) Source: http://www.clker.com/clipart-2952.html
(Cloud) Source: http://www.clker.com/search/networksym/1
Multiple Systems

• Initially - two systems cooperate and split the problem

• As the data/service becomes useful - multiple applications want to use the information / application

• Standards for SOA need to be developed

http://www.youtube.com/watch?v=mj-kCFzF0ME
The API itself is largely abstract in that it specifies an interface and controls the behavior of the objects specified in that interface. The software that provides the functionality described by an API is said to be an “implementation” of the API. An API is typically defined in terms of the programming language used to build an application.

http://en.wikipedia.org/wiki/API
Application Program Interface

- An API is a layer that is “between” an application and some aspect of the environment such as a system resource.

- The API is a contract that simplifies interacting with a resource - hides detail.

http://www.youtube.com/watch?v=31bS6cUHj-U
Biz Stone (Founder of Twitter): Yeah. The API has been arguably the most important, or maybe even inarguably, the most important thing we’ve done with Twitter. It has allowed us, first of all, to keep the service very simple and create a simple API so that developers can build on top of our infrastructure and come up with ideas that are way better than our ideas, and build things like Twitrrific, which is just a beautiful elegant way to use Twitter that we wouldn’t have been able to get to, being a very small team. So, the API which has easily 10 times more traffic than the website, has been really very important to us.

Web Services

http://en.wikipedia.org/wiki/Web_services
Web Service Protocols

• Since the Service Oriented Architecture (SOA) and Application Program Interface (API) approaches are so common we have developed general-purpose infrastructure to use applications remotely and work with remote resources across the web.

Web Service Technologies

• SOAP - Simple Object Access Protocol (software)
  • Remote programs/code which we use over the network
  • Note: Chuck does not like SOAP because it is overly complex*

• REST - Representational State Transfer (resource focused)
  • Remote resources which we create, read, update and delete remotely

http://en.wikipedia.org/wiki/REST
REST
Representational State Transfer


http://en.wikipedia.org/wiki/REST

http://www.infoq.com/articles/rest-introduction

http://wiki.developer.mindtouch.com/REST/REST_for_the_Rest_of_Us
REpresentational State Transfer

- Rest goes back to the basic concepts of the Internet
- URLs = Documents
- We have HTTP Operations on URLs: GET, PUT, POST, and DELETE
- Actually there are called **URIs - Uniform Resource Identifiers**
URI versus URN versus URL

- Uniform Resource Locator - Where to get a resource
- Uniform Resource Name - More of a “look up key”
  - urn:isbn:0-486-27557-4
- Advice - For now pretend URI = URL

HTTP Methods

• HTTP Methods Operate on a URI
• GET - Retrieve a resource at a URI
• POST - Create/Modify a resource at a URL (remember <form>)
• PUT - Create/Replace/Overwrite a resource at a URI
• DELETE - Delete a resource at a URI
What does a REST Resource look like?

- It depends - the idea is that when you GET the “resource” is returned to you in the most convenient format.
- If it is an image - it is the bytes which make up the image.
- If it is an “Object” such as a person - it may be an XML representation of that person.
GET http://directory.umich.edu/users/csev

<person>
  <name>Dr. Chuck</name>
  <email>csev@umich.edu</email>
  <office>305B West Hall</office>
</person>

An XML Serialization of the resource.
Twitter - a REST Example
Twitter REST API

- A series of URLs which you retrieve which return data
- Much like the information on twitter.com
- Returns XML data in the HTTP Document

http://apiwiki.twitter.com/REST+API+Documentation
User Methods

statuses/friends

Returns the authenticating user's friends, each with current status inline. They are ordered by the order in which they were added as friends. It's also possible to request another user's recent friends list via the id parameter below.

URL: http://twitter.com/statuses/friends.format

Formats: xml, json

Method(s): GET

Parameters:

- id. Optional. The ID or screen name of the user for whom to request a list of friends. Ex:
  http://twitter.com/statuses/friends/12345.json or
  http://twitter.com/statuses/friends/bob.xml

- user_id. Optional. Specifies the ID of the user for whom to return the list of friends. Helpful for disambiguating when a valid user ID is also a valid screen name. Ex:
  http://twitter.com/statuses/friends.xml?user_id=1401881

- screen_name. Optional. Specifies the screen name of the user for whom to return the list of friends. Helpful for disambiguating when a valid screen name is also a user ID. Ex:
  http://twitter.com/statuses/friends.xml?screen_name=101010

- page. Optional. Retrieves the next 100 friends. Ex:
  http://twitter.com/statuses/friends.xml?page=2

Returns: list of basic user information elements
Basic user information element

Basic user information elements contain primary user information with nested a `<status>` element to describe the user's most current update.

```xml
<user>
  id
  name
  screen_name
  location
  description
  profile_image_url
  url
  protected
  followers_count
  <status>
    created_at
    id
    text
    source
    truncated
    in_reply_to_status_id
    in_reply_to_user_id
    favorited
    in_reply_to_screen_name
</status>
</user>
```
<xml version="1.0" encoding="UTF-8"?>
<users type="array">
  <user>
    <id>14870169</id>
    <name>gbhatnag</name>
    <screen_name>gbhatnag</screen_name>
    <location>iPhone: 42.284775,-83.732422</location>
    <profile_image_url>http://s3.amazonaws.com/twitter_production/profile_images/54535105/profile_normal.jpg</profile_image_url>
    <followers_count>29</followers_count>
    <status>
      <created_at>Sun Mar 15 17:52:44 +0000 2009</created_at>
      <id>1332217519</id>
      <text>to add to @aatorres: projects that may fall into pervasive computing, situated technologies, distributed media, would be interesting #sxsw</text>
    </status>
  </user>
  <user>
    <id>928961</id>
    <name>Rasmus Lerdorf</name>
    .......
  </user>
</users>
http://twitter.com/statuses/friends/drchuck.xml
Retrieving Twitter Data in Python

cat twpals1.py

import urllib

TWITTER_URL = 'http://twitter.com/statuses/friends/ACCT.xml'

while True:
    print ''
    acct = raw_input('Enter Twitter Account:')
    if ( len(acct) < 1 ) : break
    url = TWITTER_URL.replace('ACCT', acct)
    document = urllib.urlopen (url).read()
    print document[:250]
Parsing XML in Python

- XML Becomes a Document Object Model - DOM

The Document Object Model (DOM) is a platform- and language-independent standard object model for representing HTML or XML documents as well as an Application Programming Interface (API) for querying, traversing and manipulating such documents.
XML as a Tree

```
<a>
  <b>B</b>
  <c>
    <d>D</d>
    <e>E</e>
  </c>
</a>
```
XML Text and Attributes

Elements

Text

```xml
<a>
  <b x="5">B</b>
  <c>
    <d>D</d>
    <e>E</e>
  </c>
</a>
```
<?xml version="1.0" encoding="UTF-8"?>
<users type="array">
  <user>
    <id>14870169</id>
    <name>Gaurav Bhatnagar</name>
    <screen_name>gbhatnag</screen_name>
    <location>42.28,-83.74</location>
    <status>
      <created_at>Sun Mar 15 17:52:44</created_at>
      <text>to add to @aatorres: projects</text>
    </status>
  </user>
  <user>
    <id>928961</id>
    <name>Rasmus Lerdorf</name>
    .......
  </user>
</users>

document = urllib.urlopen(url).read()
dom = xml.dom.minidom.parseString(document)
document = urllib.urlopen(url).read()
dom = xml.dom.minidom.parseString(document)
```python
document = urllib.urlopen(url).read()
dom = xml.dom.minidom.parseString(document)
x = dom.getElementsByTagName('user')
```

getElementsByTagName pulls out a Python List of sub-trees.
url = TWITTER_URL.replace('ACCT', acct)
document = urllib.urlopen(url).read()
dom = xml.dom.minidom.parseString(document)
count = 0
for user in dom.getElementsByTagName('user'):
    count = count + 1
print count

Counting the number of user tags...
for user in dom.getElementsByTagName('user'):
    name = getTag(user, 'screen_name')
    print name
    status = user.getElementsByTagName('status')[0]
    text = getTag(status, 'text')
    if len(text) > 0:
        print " ", text[:50]
for user in dom.getElementsByTagName('user'):
    name = getTag(user, 'screen_name')
    print name
    status = user.getElementsByTagName('status')
    text = getTag(status, 'text')
    if len(text) > 0:
        print " ", text[:50]

$ python twpals2.py

Enter Twitter Account: drchuck
Gbhatnag
to add to @aatorres: projects that
Rasmus
@nine_L Which shop is that?
def getTag(node, tagname):
    if isinstance(node, list) and len(node) < 1: return ''
    if isinstance(node, list): node = node[0]

    nodelist = node.getElementsByTagName(tagname)[0].childNodes
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            rc = rc + node.data
    return rc

for user in dom.getElementsByTagName('user'):
    name = getTag(user, 'screen_name')
XML Text and Attributes

```python
rc = ''
for node in nodelist:
    if node.nodeType == node.TEXT_NODE:
        rc = rc + node.data
```
$ python twpals2.py

Enter Twitter Account: drchuck
Gbhatnag
to add to @aatorres: projects that may fall into p
Rasmus
@nine_L Which shop is that?
Ptarjan
Home sweet home
Olegliber
introducing ourselves and our interests...
Wilm
Randomness: my firewall just informed me that Skyp
Dugsong
RT @themediaisdying Chicago Tribune changes masthe
<user>
  <id>14870169</id>
  <name>Gaurav Bhatnagar</name>
  <screen_name>gbhatnag</screen_name>
  <location>42.28,-83.74</location>
  <status>
    <created_at>Sun Mar 15 17:52:44</created_at>
    <text>to add to @aatorres: projects</text>
  </status>
</user>

```python
twpals3.py
Enter Twitter Account: drchuck
Gbhatnag
42.28,-83.74
to add to @aatorres: projects that may fall into p Rasmus
Sunnyvale, California
Grr.. #lazyweb, how do I tell Thunderbird to use
```
Summary

- Service Oriented Architecture - allows an application to be broken into parts and distributed across a network - and for standards to be developed for service reuse

- An Application Program Interface (API) is a contract for interaction

- Web Services provide infrastructure for applications cooperating (an API) over a network - SOAP and REST are two styles of web services