How ↑↑
↓↓ To
Follow ←→ Instructions
I write stories about the future
We all write stories about the future

<a href="/pages/2" rel="next">
Next
</a>
A story about the past

Test of your ability to follow instructions

1. Read every instruction before you do anything.
2. Write your name in the top right hand corner of the paper.
3. Write today's date to the left of your name.
4. Who is your English teacher? ______________

...

18. Draw a triangle: ______________
19. Draw a circle around the triangle.
20. STOP! Do not write anything on this paper! Just turn it in blank!
A Job Application Puzzle!!

Send a POST to:

https://www.example.com/jobs/applications/

with the Content-Type set to application/json. The body should be a JSON dictionary with the following keys:

- **name**: String with your name (required)
- **email**: String with your email address (required)
- **urls**: Array of strings with links to your resume, site, or code samples (required)

You will receive a blank 200 response on success.
url = "https://www.example.com/jobs/applications"
data = dict(
    name='Leonard "Bakersfield Bruiser" Richardson',
    email='leonardr@segfault.org',
    urls=['http://www.crummy.com/self/resume.html',
          'http://www.crummy.com/writing/'])
headers = {'Content-type': 'application/json'}

import json
import httpplib2
http = httpplib2.Http()
http.request(
    url, "POST", json.dumps(data), headers=headers)
Job Application Puzzle! #2

Send a POST to:

https://www.example.com/jobs/applications/

with the Content-Type set to `application/x-www-form-urlencoded`. The body should be a form submission with the following keys:

- `name`: String with your name (required)
- `email`: String with your email address (required)
- `urls`: Array of strings with links to your resume, site, or code samples (required)

You will receive a blank 200 response on success.
Solving the second puzzle

url = "https://www.example.com/jobs/applications"
data = dict(
    name='Leonard "Bakersfield Bruiser" Richardson',
    email='leonardr@segfault.org',
    urls=[
        'http://www.crummy.com/self/resume.html',
        'http://www.crummy.com/writing/'])
Headers = {
    'Content-type':
        'application/x-www-form-urlencoded'}

import json
import http
http = http.Http()
http.request(
    url, "POST", urllib.urlencode(data, doseq=True),
    headers=headers)
Job Application Puzzle #3

<form method="POST"
action="https://www.example.com/jobs/applications/">
<ul>
<li>
<label for="name">Your name (required)</label>
<input id="name" name="name" type="text"/>
</li>
<li>
<label for="email">Your email address (required)</label>
<input id="email" name="email" type="text"/>
</li>
<li>
<label for="urls">Links to your resume, site, or code samples (required)</label>
<textarea id="urls" name="urls"></textarea>
</li>
</ul>
<input type="submit"/>
</form>
Surprise! It's boring!

- Your name (required)
- Your email address (required)
- Links to your resume, site, or code samples (required)

Submit Query
Questions resolved by the HTML form

- When I send the next HTTP request, what should it look like?

Questions not resolved

- What is my name?
- What is my email address?
- Where is my resume?
- Is my resume up to date?
- Do I want this job?
Part One: How to Recognize Instructions

1. Place 1x1 block.
2. Add 2x2 block to top.
3. Attach 1x1 block to side.
4. Insert 1x1 block into top.
5. Add 1x1 block to side.
6. Place 1x1 block on top.
7. Attach 2x2 block to side.
8. Insert 1x1 block into top.
9. Add 1x1 block to side.
10. Place 1x1 block on top.
11. Insert 10x1 block into top.
12. Add 1x1 block to side.
13. Place 1x1 block on top.

Credit: Flickr user pasukaru76
"This file is available via anonymous Internet FTP to the host ftp.cc.utexas.edu, in the directory pub/minerva."

(1995)
RFC 1738 (1994)

ftp://ftp.cc.utexas.edu/pub/minerva/paganism-faq
This file is available via anonymous Internet FTP.
Machine-readable instructions are defined in terms of human-readable instructions

- "The FTP URL scheme is used to designate files and directories on Internet hosts accessible using the FTP protocol" – RFC 1738

- "If the reader selects this text, (s)he should be presented with another document whose network address is defined by the value of the HREF attribute." - Original Internet-Draft for HTML
2. Native-language bindings
3. Service descriptors
4. Hypermedia

• Stories about the future
• Control data
• Served alongside regular data
  – Links
  – Forms
5. Code on demand

- Client-language code
- Served alongside data
- Code for making the future happen

Credit: thebittenword.com
Cheating at the job application puzzle (part 1)

<script type="text/javascript">
function send_json()
{
    form = document.forms["form"]
    data = {};
    data['name'] = form["name"].value;
    data['email'] = form["email"].value;
    data['urls'] = form["urls"].value.split("\n");

    url = "https://www.example.com/jobs/applications/";
    client = new XMLHttpRequest();
    client.open("POST", url, false);
    client.setRequestHeader("Content-type","application/json");
    client.send(JSON.stringify(data));
}
</script>
Cheating at the job application puzzle (part 2)

<form id="form">
  <li>
    <label for="name">Your name (required)</label>
    <input id="name" name="name" type="text"/>
  </li>

  <li>
    <label for="email">Your email address (required)</label>
    <input id="email" name="email" type="text"/>
  </li>

  <li>
    <label for="urls">Links to your resume, site, or code samples (required)</label>
    <textarea id="urls" name="urls"></textarea>
  </li>
</form>

<a href="#" onclick="send_json();">Submit</a>
Code on demand depends on hypermedia

<a href="" 
   onclick="send_json();">Submit</a>
Evil twins

- Service descriptors are dysfunctional hypermedia.
- Native-language bindings are dysfunctional code-on-demand.
The problem is "installation"
Summary: 5 kinds of instructions

• Human-readable:
  "The service endpoint is at http://www.example.com/."  

• Native-language client 

• Service descriptor 

• Hypermedia:
  
```html
<link rel="index" href="https://www.example.com/"/>
```

• Code on demand:

```javascript
new XMLHttpRequest().open("GET", "https://www.example.com/");
```
Part Two: How To Write Instructions

**READING PRACTICE TEST**

**Part 3: Reading Comprehension (cont.)**

**Directions:** Listen to your teacher read the sentences. Look at the pictures. Choose the sentence that matches the picture. Practice with example B. Do 3–6 the same way.

**Example**

<table>
<thead>
<tr>
<th>F</th>
<th>Todd ate cereal.</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>I love my horse.</td>
</tr>
<tr>
<td>H</td>
<td>The weather is nice.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>The boat sunk.</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>My pen does not write.</td>
</tr>
</tbody>
</table>
Remember Mike's H factors

• Link Support
  - [LE] Embedding links
  - [LO] Outbound links
  - [LT] Templated queries
  - [LN] Non-Idempotent updates
  - [LI] Idempotent updates

• Control Data Support
  - [CR] Control data for read requests
  - [CU] Control data for update requests
  - [CM] Control data for interface methods
  - [CL] Control data for links
The weigh-in: AtomPub

Hypermedia Factors

CL
CR
CU
CM
LE
LO
LT
LN
LI

Atom
The weigh-in: HTML

Hypermedia Factors

CL
CR
CU
CM
LE
LO
LT
LN
LI

HTML
The weigh-in: JSON and XML

Hypermedia Factors

CL
CR CU CM
LE LO LT LN LI
Nobody cares

"There is no single registered media type that contains all nine of these factors."

--Mike Amundsen

(REST: From Research to Practice)
Nobody cares

It seems that we currently do not understand how PUT and DELETE will be useful for HTML forms.

For DELETE, it's indeed easy to create a useful request. However, server implementations usually respond with 200 and a minimal response body ("deleted") or 204 (no content). So it's not clear how this can be used in a web application.

For PUT, it seems there's no real use case as long as the web page doesn't have full control over the payload, and also can set the content type.

Please consider removing this feature until there's a clearer understanding about what it's good for.
Nobody cares

Month

Services added to ProgrammableWeb

- JSON only
- XML+JSON
- XML only
- HTML
- Atom
What do we really want?

Javascript **Object** Notation
Simple **Object** Access Protocol
Dark matter
We make websites

**Javascript** Object Notation
Web sites as web services, 2005

• Hit the home page
• Download some hypermedia (HTML)
• Fill out a hypermedia form and click submit
• Browser page refresh
• More hypermedia!
Web sites as web services, 2012

- Hit the home page
- Download hypermedia (HTML) and code-on-demand (Javascript)
- Fill out a hypermedia form and click submit
- XMLHttpRequest request (no page refresh)
- Download some data (JSON)
- DOM update through Javascript
Web site as web service

Browser

- DOM
- XMLHttpRequest

Server

GET

HTML + Javascript

GET

JSON
The "API" part

Browser

XMLHttpRequest

GET

JSON

Server
Human-readable documentation replaces hypermedia and code-on-demand
Part 3: Why To Follow Instructions

ZERO GRAVITY TOILET

PASSENGERS ARE ADVISED TO READ INSTRUCTIONS BEFORE USE

1. The toilet is of the standard zero-gravity type. Depending on requirements, system A and/or system B can be used. Details of which are clearly marked in the toilet compartment. When operating system A, depress lever and a plastic deflector eliminator will be dispensed through the slot immediately underneath. When you have fastened the adhesive lip, attach connection marked by the large “x” outlet hole. Twist the silver coloured ring one inch below the connection point until you feel it lock.

2. The toilet is now ready for use. The slopover cleaner is activated by the pull switch on the lip. When securing, twist the ring back to its initial condition, so that the two orange lines meet. Disconnect. Place the deflector eliminator in the vacuum receptacle to the rear. Activates by pressing the blue button.

3. The controls for system B are located on the opposite wall. The red release switch places the deflector eliminator in position: it can be adjusted manually up or down by pressing the blue manual release button. The opening is self-adjusting. To secure after use, press the green button which simultaneously activates the evaporator and returns the deflector eliminator to its storage position.

4. You may leave the lavatory if the green anti-light is on over the door. If the red light is illuminated, one of the lavatory facilities is not properly secured. Press the “Slopperless” exit button to the right of the door. She will secure all facilities from her control panel outside. When green anti-light goes on you may open the door and leave. Please close door behind you.
Your choices

<form id="red" method="POST"
   action="https://www.example.com/red">
   <input type="submit" value="Red"/>
</form>

<form id="blue" method="POST"
   action="https://www.example.com/blue">
   <input type="submit" value="Blue"/>
</form>
Your choices

Red

Blue
1. Human-readable documentation

<form>
    <input type="submit" id="submitred" value="Red"/>
    <label for="submitred">
        This form controls the Interocerator.</label>
</form>

<form>
    <input type="submit" id="submitblue" value="Blue"/>
    <label for="submitblue">
        This form controls the flux capacitor.</label>
</form>
2. HTTP's uniform interface

<a href="http://www.example.com/">Click here for a representation!</a>
3. Link relations

<a href="/pages/2" rel="next">Next</a>

<table>
<thead>
<tr>
<th>monitor-group</th>
<th>monitor changes in an HTTP resource.</th>
<th>[RFC5989]</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>Indicates that the link's context is a part of a series, and that the next in the series is the link target.</td>
<td>[<a href="http://www.w3.org/TR/html5/links.html#link-type-next">http://www.w3.org/TR/html5/links.html#link-type-next</a>]</td>
</tr>
<tr>
<td>next-archive</td>
<td>Refers to the immediately following archive</td>
<td>[RFC5005]</td>
</tr>
</tbody>
</table>
4. Media types

Content-type: text/calendar

BEGIN:VCALENDAR
VERSION:2.0
BEGIN:VEVENT
DTSTART:19970714T170000Z
DTEND:19970715T035959Z
LOCATION:Conference Room - F123
SUMMARY:Bastille Day Party
END:VEVENT
END:VCALENDAR
JSON doesn't mean anything

Content-type: application/json

```json
{
  "version": "2.0",
  "events": [
    {
      "dtstart": "19970714T170000Z",
      "dtend": "19970715T035959Z",
      "location": "Conference Room - F123",
      "summary": "Bastille Day Party"
    }
  ]
}
```
Media types define semantics

Content-type: application/atomsvc+xml

<collection href="http://example.org/blog/main">
  <atom:title>My Blog Entries</atom:title>
  <categories href="http://example.com/cats/forMain.cats" />
</collection>
But they also define behavior under the uniform interface!

Content-type: application/atomsvc+xml

<collection href="http://example.org/blog/main">
  <atom:title>My Blog Entries</atom:title>
  <categories
    href="http://example.com/cats/forMain.cats" />
</collection>
Media types can be hypermedia types

17.3 The FORM element

```xml
<!-- ELEMENT FORM - - (%block; | SCRIPT)+ -(FORM) -- interactive form -->
<!ATTLIST FORM
 %attrs: -- %coreattrs, %i18n, %events --
 action %URI: #REQUIRED -- server-side form handler --
 method (GET|POST) GET -- HTTP method used to submit the form--
 enctype %ContentType; "application/x-www-form-urlencoded"
 accept %ContentType; #IMPLIED -- list of MIME types for file upload --
 name CDATA #IMPLIED -- name of form for scripting --
 onsubmit %Script; #IMPLIED -- the form was submitted --
 onreset %Script; #IMPLIED -- the form was reset --
 accept-charset %Charsets; #IMPLIED -- list of supported charsets --
 >
```

Start tag: **required**, End tag: **required**

**Attribute definitions**

**action** = **uri** [CT]

This attribute specifies a form processing agent. User agent behavior for a value other than an HTTP URI is undefined.

**method** = **get** | **post** [CI]

This attribute specifies which HTTP method will be used to submit the form data set. Possible (case-insensitive) values are "get" (the default) and "post". See the section on form submission for usage information.
5. Profiles

Content-Type: application/schema+json

{ "description": "A representation of an event",  
  "type": "object",  
  "properties": {  
    "dtstart": { "format": "date-time", "type": "string", "description": "Event starting time", "required": true },  
    "summary": { "type": "string", "required": true },  
    "location": { "type": "string" },  
    "url": { "type": "string", "format": "url" },  
    "dtend": { "format": "date-time", "type": "string", "description": "Event ending time" },  
    "duration": { "format": "date", "type": "string", "description": "Event duration" },  
    "rdate": { "format": "date-time", "type": "string", "description": "Recurrence date" },  
    "rrule": { "type": "string", "description": "Recurrence rule" },  
    "category": { "type": "string" },  
    "description": { "type": "string" },  
    "geo": "http://json-schema.org/geo"  
}
Summary: 5 reasons to follow instructions

- The human-readable docs say it'll get you what you want.
- Or the HTTP method aligns with what you want.
- Or the link relation points to the document you want.
- Or the media type semantics say it'll help.
- Or the semantics defined in a profile say it'll help.
Don't Repeat Yourself

REST API Resources

Timelines
Timelines are collections of Tweets, ordered with the most recent first.

Tutorial
Core Concepts > Open Graph > Tutorial

This tutorial will guide you through the key concepts. We will build a sample recipe app that allows for planning, but will also help speed up the app development process.
The Parable of OAuth

AuthSub for Web Applications

Important: AuthSub has been officially deprecated as of April 20, 2012. It will continue to work as per our deprecation policy, but we encourage you to migrate to OAuth 2.0 as soon as possible.

flickr.auth.getFrob

Returns a frob to be used during authentication. This method call must be signed, and is deprecated in favour of OAuth.
Don't Repeat Yourselves

An open protocol to allow secure API authorization in a simple and standard method from desktop and web applications.

Read the OAuth 2 specification »
Ripped From the Headlines

- 71 eCommerce APIs: Seatwave, Playme and eBay
- 78 Hosting APIs: Heroku, CloudMine and BuzzData
- 123 Database APIs: GeoNames, Freebase and Yahoo Query Language
- 53 Microblogging APIs: Twitter, TwitPic and What The Trend
Getting to twenty

- What if there were twenty competing standards for "database API"s?
- That would be insanely wasteful...
- but better than the 123 one-offs we have now.
"The big question in my mind is whether architectures consciously designed with REST in mind will “win” over architectures that are simple but only intermittently RESTful."

--Leonard Richardson, 2007
The lessons of 2008-2012

- JSON has won the representation war.
- So there's little point in minting more media types.
- But JSON without hypermedia means an endless proliferation of internal designs published as one-off "API"s.
- So... ?
1. JSON links, HTML forms

- JSON-LD, JSON Reference, Hypertext Application Language
- Plus HTML
2. Code on demand all the way

- You write the native-language client library.
- But the user never "installs" it.
- Just like your website!
- A stupid idea???
3. OData

- AtomPub with a JSON representation.
- And more relationships between resources.
- And rules for filtering.
- And a schema definition language.
- And everything you need to relive the glory days of SOAP/WSDL.
- Except it's RESTful...
4. Collection+JSON

- AtomPub with a JSON representation
- And that's it
- Close enough
- Good enough
5. Copy someone else

At least you won't make the problem worse!

Credit: Marshall Astor
6. Go freelance

- Create HTTP intermediaries that standardize popular interfaces.
- Then sell access to the intermediaries.
- Or give them away to demonstrate proof-of-concept.
Summary: 6 design strategies that might work

- JSON links, HTML forms
- Code on demand all the way
- OData
- Collection+JSON
- Third-party reuse
- "I fixed it for you"
Epilogue: How to agree on instructions

- We're used to writing instructions for others to follow.
- The status quo won't change until we start following instructions ourselves.
- But this requires talking to (or copying) competitors.
Generalities

- Product/Shopping cart
- Entry/Blog
- Row/Table
- Event/Calendar
- Transaction/Register
Iterative standardization

- Look for bits of the problem other people have solved, and adopt their solutions.
- Try to solve what's left in a way that's reusable.
- Write up instructions for the next person to come along.
Justice will take us millions of intricate moves

- Start reusing
- Start copying
- Start wrapping
- Start linking within your representations
- Start linking to other peoples' human-readable instructions
- Don't worry about hypermedia forms (yet)
- "I built it one piece at a time." - Johnny Cash
Progress

Credit: Marcin Wichary