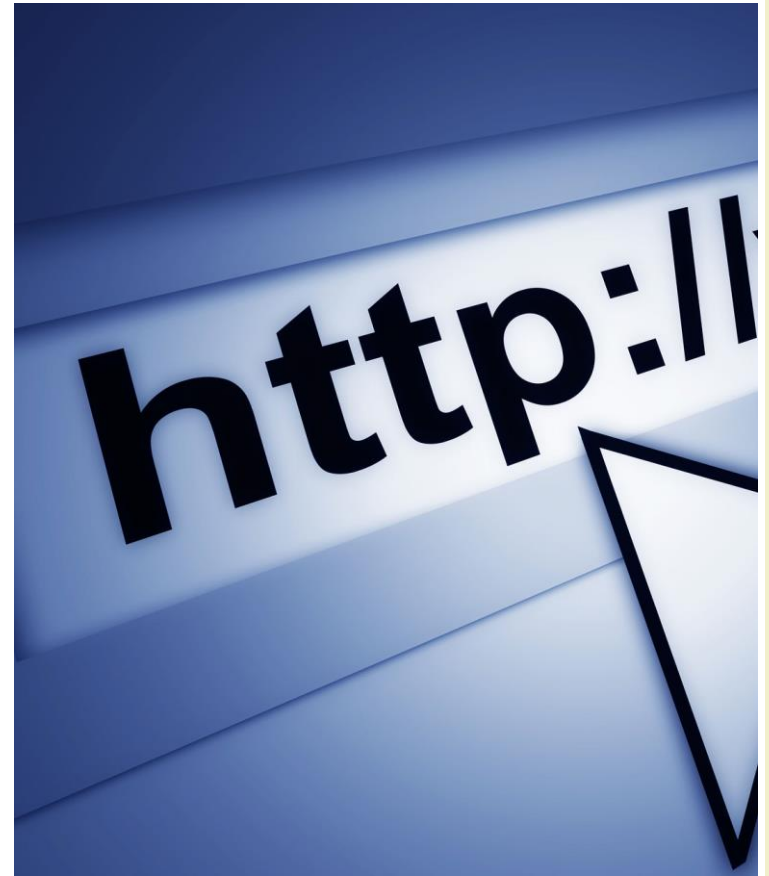


REST over HTTP

Ambient intelligence

Fulvio Corno

Politecnico di Torino, 2015/2016



POLITECNICO
DI TORINO



Goal

- Understanding main communication protocol (http)
- How to use REST architectures to integrate (call and/or offer) remote services

Summary

- HTTP (Hypertext Transfer Protocol)
- REST (Representational State Transfer)
- JSON (JavaScript Object Notation)



Web Architecture and Technologies

HTTP

HYPertext TRANSFER PROTOCOL

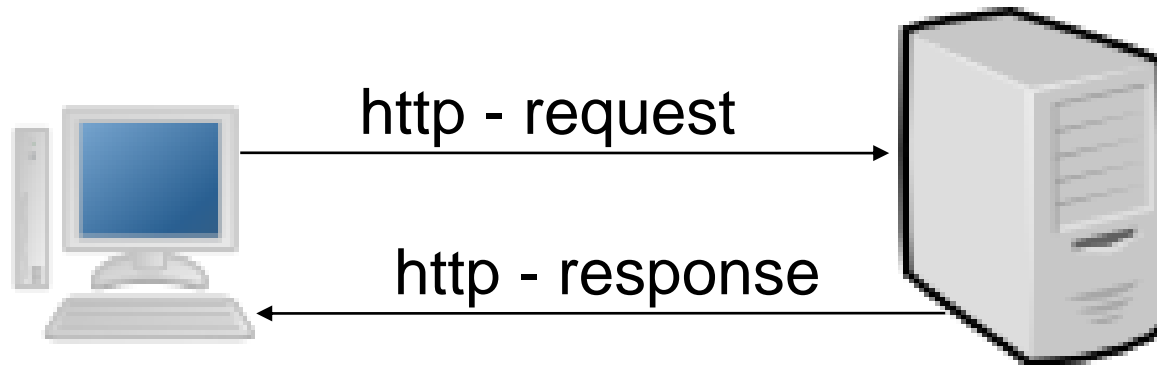
What is HTTP?

- HTTP stands for Hypertext Transfer Protocol
- It is the network protocol used to delivery virtually all data over the WWW:
 - Images
 - HTML files
 - Query results
 - Etc.
- HTTP takes places over TCP/IP connections

<http://www.ietf.org/rfc/rfc2616.txt>

HTTP clients and servers

- A browser is an HTTP client because it sends requests to an HTTP server, which then sends responses back to the client.
- The standard port for HTTP servers to listen on is 80, though they can use any port.



HTTP messages

- The format of the request and response messages are similar.
 - An initial line
 - Zero or more header lines
 - A blank line (CRLF)
 - An optional message body

```
Initial line  
header1: value1  
header2: value2  
header3: value3  
  
message body...
```

Header Example

```
HEAD /index.html HTTP/1.1  
Host: www.example.com
```

Request



Response



```
HTTP/1.1 200 OK  
Date: Mon, 23 May 2005 22:38:34 GMT  
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)  
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT  
Etag: "3f80f-1b6-3e1cb03b"  
Accept-Ranges: bytes  
Content-Length: 438  
Connection: close  
Content-Type: text/html; charset=UTF-8
```


HTTP request – initial line

- The initial line is different for the request and the response.
- A **request** initial line has three parts separated by white spaces:
 - A method name
 - The local path of the requested resource
 - The version of the HTTP being used
- `GET /path/to/file/index.html HTTP/1.0`

HTTP request – initial line

- The method name is always in upper case.
- There are several methods for a HTTP request
 - GET (most commonly used)
 - POST (used for sending form data)
 - HEAD
 - ...
- The path is the part of the URL after the host name
 - `http://www.tryme.com/examples/example1.html`

HTTP Method Basics

HEAD	Gets just the HTTP header
GET	Gets HTTP head & body
POST	Submits data in the body to the server
PUT	Uploads a resource
DELETE	Deletes a resource
TRACE	Echo's back the request
OPTIONS	Gets a list of supported methods
CONNECT	Converts to a TCP/IP tunnel for HTTPS
PATCH	Apply partial modifications to a resource

HTTP request – initial line

- The HTTP version is always in the form
 - HTTP/x.x (uppercase)
- The versions currently in use are:
 - HTTP/1.0
 - HTTP/1.1

HTTP response – initial line

- The **response** initial line is usually called status line and has also 3 parts separated by spaces:
 - The HTTP version
 - The response status code
 - An English phrase describing the status code
- Example:
 - HTTP/1.0 200 OK
 - HTTP/1.0 404 Not Found

Response Status Codes

- 1xx – Informational
- 2xx – Success
- 3xx – Redirection
- 4xx – Client Error
- 5xx – Server Error

Response Status Codes

- 1xx – Informational
 - 2xx – Success
 - 3xx – Redirection
 - 4xx – Client Error
 - 5xx – Server Error
- 100 = Continue
 - 102 = Processing
 - 200 = OK
 - 201 = Created
 - 204 = No Content
 - 206 = Partial Content
 - 301 = Moved Permanently
 - 302 = Found (Moved Temp)
 - 307 = Temp Redirect
 - 400 = Bad Request
 - 401 = Unauthorised
 - 402 = Payment Required
 - 403 = Forbidden
 - 404 = Not Found
 - 405 = Method Not Allowed
 - 409 = Conflict
 - 450 = Blocked by Windows Parental Controls
 - 500 = Internal Server Error
 - 501 = Not Implemented

HTTP msg – header lines

- Header lines provide information about the request/response or about the object sent in the message body
- The header lines are in the following format:
 - One line per header
 - Form: “Header-Name: value”
- HTTP/1.0 defines 16 headers (none required); HTTP/1.1 defines 46 headers and 1 is required in requests:
 - Host:

Request headers

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language
- Authorization;
- Expect
- From
- **Host**
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Max-Forwards
- Proxy-Authorization
- Range
- Referer
- TE
- User-Agent

Response Headers

- Accept-Ranges
- Age
- Etag
- Location
- Proxy-Authenticate
- Retry-After
- Server
- Vary
- WWW-Authenticate

General (request & response) headers

- Cache-Control
- Connection
- Date
- Pragma
- Trailer
- Transfer-Encoding
- Upgrade
- Via
- Warning

Message body

- An HTTP message may have a **body** of data sent after the header lines.
- In a **response** the body contains the resource returned to the client
 - Images
 - text/plain, text/html
 - ...
- In a **request** it may contain the data entered by the user in a form or a file to upload, etc.

Content Type

- Proper name: Internet Media Type
 - Also known as MIME type
- Parts: Type, SubType, Optional Parameters
- x- prefix for nonstandard types or subtypes
- vnd. prefix for vendor specific subtypes

Content Type Examples

Content-Type	File
text/plain	Plain text
text/xml	XML
text/html	HTML
image/png	PNG image
audio/basic	Wave audio
audio/mpeg	MPEG audio (MP3)
video/quicktime	Quicktime Video
application/pdf	Adobe PDF document
application/javascript	JavaScript
application/vnd.ms-powerpoint	PowerPoint file
application/x-rar-compressed	RAR file

Message body

- Some HTTP headers are used to describe the body content:
 - Allow
 - Content-Encoding
 - Content-Language
 - Content-Length
 - Content-Location
 - Content-MD5
 - Content-Range
 - Content-Type
 - Expires
 - Last-Modified
 - extension-header n

HTTP Authentication

- Basic Authentication
 - Easy to do, but plain text. Easy to reverse engineer. Less of an issue when used with SSL.
- Digest Authentication
 - Harder to do, still plain text. Hard (impossible?) to reverse engineer because of hashing.
- NTLM Authentication
 - Hard to do, Windows specific. Hard (impossible?) to reverse engineer.
- Note: usually, authentication is dealt at the application level, and http mechanisms are not used

HTTP methods: HEAD

- The HEAD method is like the GET except it asks the server to return the **response headers, only**. Is useful for checking the characteristics of a resource without actually downloading it.
- The response to a HEAD request **never** contains a message body, only the initial line and the headers.

HTTP methods: POST

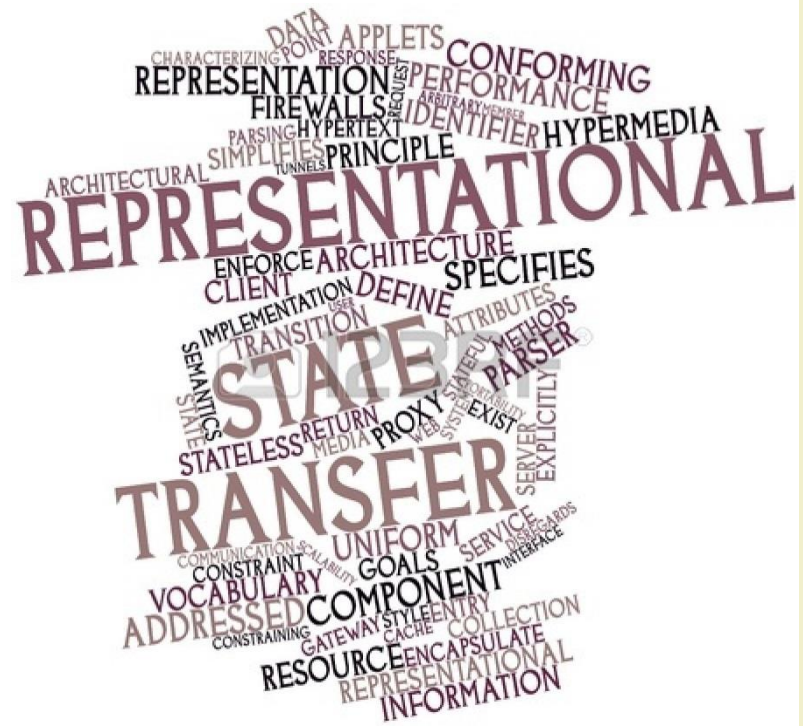
- Used to send data to the server
- A POST request is different from the GET request as:
 - There's a block of data sent with the request in the request message body
 - The request URI is not a resource to retrieve, it's usually a program or a server page that handles the sent data
 - The HTTP response is usually not-static (generated depending on the received data)

GET vs POST

- The most common use of the POST method is to submit data gathered from user forms
- Also the GET can be used to submit form data however, the data is encoded in the request URI
 - `http://www.example.com/example.html?var=This+is+a+simple+%26+short+test`
- GET requests should be **idempotent**, i.e., may be repeated without changing the state of the application

HTTP as transport layer

- HTTP is used as “transport” for many resources / protocols
- Protocols:
 - SOAP (Simple Object Access Protocol)
 - XML-RPC
 - WebDAV
- Resources:
 - Text (plain, HTML, XHTML, ...)
 - Images (gif, jpeg, ...)
 -



Web Architecture and Technologies

REST

REPRESENTATIONAL STATE TRANSFER



REST

- **Representational State Transfer**



Roy T. Fielding

Senior Principal Scientist, [Adobe](#)

Co-founder, [Apache HTTP Server Project](#)

Director, [The Apache Software Foundation](#)

Ph.D., [Information and Computer Science, UC Irvine](#)

- [@fielding](#); Blog: [Untangled](#)
- Email: fielding at (choose **one** of) gbiv.com, adobe.com, apache.org

- A style of software architecture for distributed systems
- Platform-independent (you don't care if the server is Unix, the client is a Mac, or anything else),
- Language-independent (C# can talk to Java, etc.),
- Standards-based (runs on top of HTTP)
- Can easily be used in the presence of firewalls.

What is a Resource?

- A resource can be anything that has identity
 - a document or image
 - a service, e.g., “today’s weather in Seattle”
 - a collection of other resources
 - non-networked objects (e.g., people)
- The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity that corresponds to that mapping at any particular point in time!

Main principles

- Resource: source of specific information
- Mapping: Resources \leftrightarrow URIs
- Client and server exchange *representations* of the resource
 - The same resource may have different representations
 - E.g., XML, JSON, HTML, RDF, ...
- Operations on the Resource is done by means of HTTP methods
 - GET, POST, PUT, DELETE

Main Types of Resources

- **Collection** resource

- Represents a set (or list) of items
- Format: /resource
- E.g., `http://api.polito.it/students`
`http://api.polito.it/courses`

- **Element (Item)** resource

- Represents a single item, and its properties
- Format: /resource/identifier
- E.g., `http://api.polito.it/students/s123456`
`http://api.polito.it/courses/01PRD`

Best practice

- Nouns (not verbs)
- Plural nouns
- Concrete names (not abstract)
 - /courses, not /items

Actions use HTTP Methods

- GET
 - Retrieve the representation of the resource (in http response body)
 - Collection: the list of items
 - Element: the properties of the element
- POST
 - Create a new resource (data in http request body)
 - Use a URI for a Collection
- PUT
 - Update an existing element (data in http request body)
 - Mainly for Elements' properties
- DELETE

Actions on Resources - Example

Resource	POST create	GET read	PUT update	DELETE delete
/dogs	Create a new dog	List dogs	Bulk update dogs	Delete all dogs
/dogs/1234	Error	Show Bo	If exists update Bo	Delete Bo
			If not error	

Relationships

- A given Element may have a (1:1 or 1:N) relationship with other Element(s)
- Represent with: /resource/identifier/resource
- E.g.,
`http://api.polito.it/students/s123456/courses`
`http://api.polito.it/courses/01PRD/students`

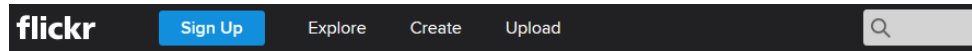
Representations

- Returned in GET, sent in PUT/POST
- Different formats are possible
- Mainly: XML, JSON
 - But also: SVG, JPEG, TXT, ...
 - In POST: URL-encoding
- Format may be specified in
 - Request headers
 - Accept: `application/json`
 - URI extension
 - `http://api.polito.it/students/s123456.json`
 - Request parameter
 - `http://api.polito.it/students/s123456?format=json`

Real Life: Flickr API

- Resource: Photos
- Where:
 - `http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}.jpg`
 - `http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}_{mstb}.jpg`
 - `http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{o-secret}_o.(jpg|gif|png)`
- What: JPEG, GIF or PNG (defined in the URL)
 - `http://farm1.static.flickr.com/2/1418878_1e92283336_m.jpg`

Real Life: flickr



The App Garden

[Create an App](#) | [API Documentation](#) | [Feeds](#) | [What is the App Garden?](#)

The Flickr API is available for non-commercial use by outside developers. Commercial use is possible by prior arrangement.

Read these first:

- [Developer Guide](#)
- [Overview](#)
- [Encoding](#)
- [User Authentication](#)

- [Dates](#)
- [Tags](#)
- [URLs](#)
- [Buddyicons](#)

- [Flickr APIs Terms of Use](#)

- [API Keys](#)
- [Developers' mailing list](#)

Photo Upload API

- [Uploading Photos](#)
- [Replacing Photos](#)
- [Example Request](#)
- [Asynchronous Uploading](#)

Request Formats

- [REST](#)
- [XML-RPC](#)
- [SOAP](#)

Response Formats

- [REST](#)
- [XML-RPC](#)
- [SOAP](#)
- [JSON](#)
- [PHP](#)

API Methods

activity

- [flickr.activity.userComments](#)
- [flickr.activity.userPhotos](#)

auth

- [flickr.auth.checkToken](#)
- [flickr.auth.getFrob](#)
- [flickr.auth.getFullToken](#)
- [flickr.auth.getToken](#)

auth.oauth

- [flickr.auth.oauth.checkToken](#)
- [flickr.auth.oauth.getAccessToken](#)

blogs

- [flickr.blogs.getList](#)
- [flickr.blogs.getServices](#)
- [flickr.blogs.postPhoto](#)

cameras

- [flickr.cameras.getBrandModels](#)
- [flickr.cameras.getBrands](#)

collections

- [flickr.collections.getInfo](#)
- [flickr.collections.getTree](#)

commons

- [flickr.common.getInstitutions](#)

contacts

- [flickr.contacts.getList](#)

<https://www.flickr.com/services/api/>

Real Life: Twitter API



[Twitter](#) / [Developers](#) / [Documentation](#) / [REST APIs](#) English

API Console Tool

Public API

- Uploading Media
- The Search API
- The Search API: Tweets by Place
- Working with Timelines
- API Rate Limits
- API Rate Limits: Chart
- GET statuses/mentions_timeline
- GET statuses/user_timeline
- GET statuses/home_timeline
- GET statuses/retweets_of_me
- GET statuses/retweets/:id
- GET statuses/show/:id
- POST statuses/destroy/:id
- POST statuses/update
- POST statuses/retweet/:id
- POST statuses/update_with_media
- GET statuses/oembed
- GET statuses/retweeters/ids
- GET statuses/lookup
- POST media/upload
- GET direct_messages/sent
- GET direct_messages/show

REST APIs

The [REST APIs](#) provide programmatic access to read and write Twitter data. Author a new Tweet, read author profile and follower data, and more. The REST API identifies Twitter applications and users using [OAuth](#); responses are available in JSON.

If your intention is to monitor or process Tweets in real-time, consider using the [Streaming API](#) instead.

Overview

Below are the documents that will help you get going with the REST APIs as quickly as possible

- [API Rate Limiting](#)
- [API Rate Limits](#)
- [Working with Timelines](#)
- [Using the Twitter Search API](#)
- [Uploading Media](#)
- [Multiple Media Entities in Statuses](#)
- [Finding Tweets about Places](#)

Latest Updates

As of version 1.1 of the Twitter API, the more recent updates to our API are highlighted below. We're excited about what it means for developers and we've captured all the meaningful changes here so you don't miss a thing.

Default entities and retweets

<https://dev.twitter.com/rest/public>

Real Life: Google Calendar API

Google Developers

Google Calendar API X Search

fulvio.corno@gmail.com Sign out

Products > Google Apps > Google Calendar API

Google Calendar API 8+1 20 Write Feedback

GUIDES

- Google Calendar API
- Get Started
- Quickstarts
- Use the Calendar API
- Calendar Gadgets
- CalDAV API Developer's Guide

REFERENCE

- Resource Summary**
- Acl
- CalendarList
- Calendars
- Channels
- Colors
- Events
- Freebusy
- Settings

Usage Limits

What's New in v3

API Reference

This API reference is organized by resource type. Each resource type has one or more data representations and one or more methods.

Resource types

- [Acl](#)
- [CalendarList](#)
- [Calendars](#)
- [Channels](#)
- [Colors](#)
- [Events](#)
- [Freebusy](#)
- [Settings](#)

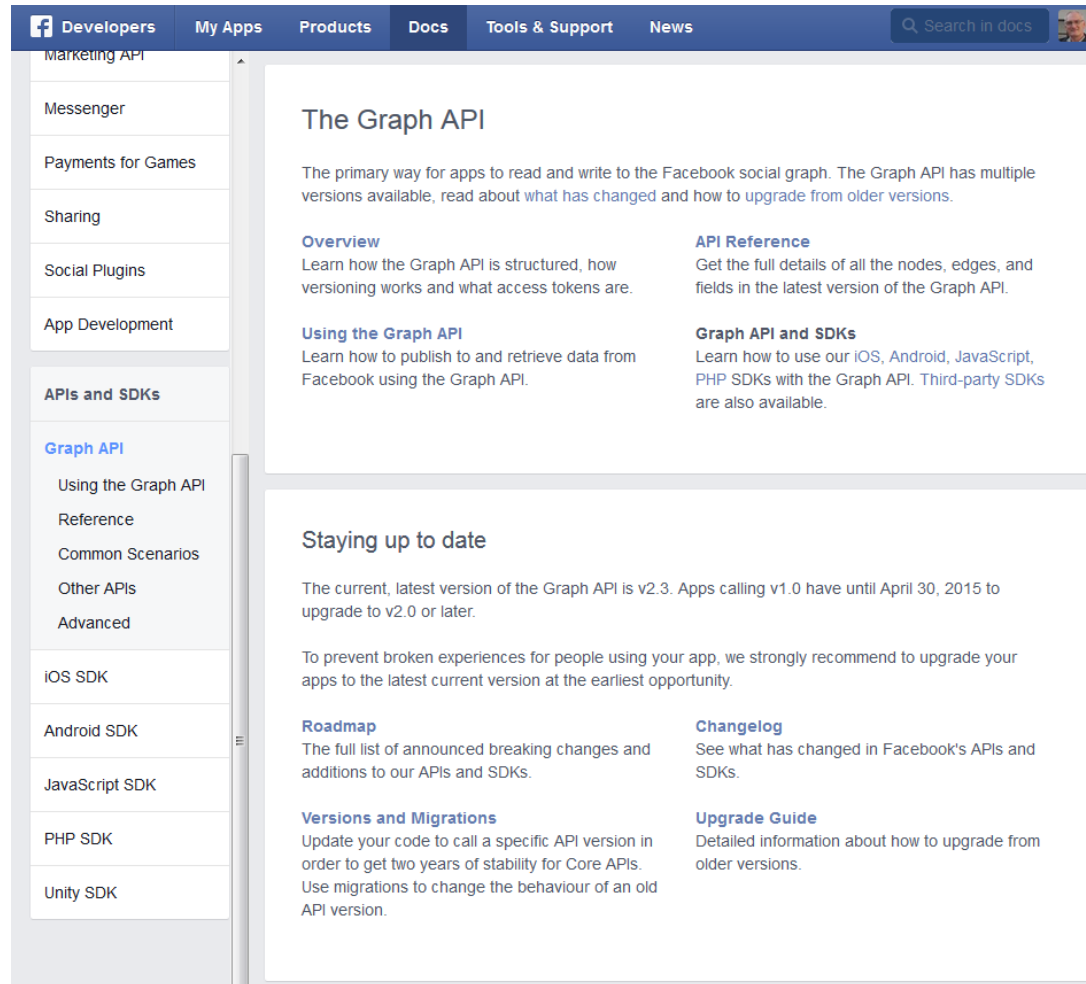
Acl

For Acl Resource details, see the [resource representation](#) page.

Method	HTTP request	Description
URIs relative to https://www.googleapis.com/calendar/v3 , unless otherwise noted		
delete	DELETE /calendars/ <i>calendarId</i> /acl/ <i>ruleId</i>	Deletes an access control rule.
get	GET /calendars/ <i>calendarId</i> /acl/ <i>ruleId</i>	Returns an access control rule.
insert	POST /calendars/ <i>calendarId</i> /acl	Creates an access control rule.
list	GET /calendars/ <i>calendarId</i> /acl	Returns the rules in the access control list for the calendar.

<https://developers.google.com/google-apps/calendar/v3/reference/>

Real life: Facebook Graph API



The screenshot shows the Facebook Developers documentation page for the Graph API. The top navigation bar includes links for Developers, My Apps, Products, Docs, Tools & Support, and News, along with a search bar. A left sidebar lists various API categories, with 'Graph API' selected and expanded to show sub-sections like 'Using the Graph API', 'Reference', 'Common Scenarios', 'Other APIs', 'Advanced', 'iOS SDK', 'Android SDK', 'JavaScript SDK', 'PHP SDK', and 'Unity SDK'. The main content area is titled 'The Graph API' and contains an introductory paragraph, an 'Overview' section, an 'API Reference' section, a 'Using the Graph API' section, and a 'Graph API and SDKs' section. Below this, there is a 'Staying up to date' section with a 'Roadmap' section, a 'Changelog' section, and an 'Upgrade Guide' section.

The Graph API

The primary way for apps to read and write to the Facebook social graph. The Graph API has multiple versions available, read about what has changed and how to upgrade from older versions.

Overview
Learn how the Graph API is structured, how versioning works and what access tokens are.

API Reference
Get the full details of all the nodes, edges, and fields in the latest version of the Graph API.

Using the Graph API
Learn how to publish to and retrieve data from Facebook using the Graph API.

Graph API and SDKs
Learn how to use our iOS, Android, JavaScript, PHP SDKs with the Graph API. Third-party SDKs are also available.

Staying up to date

The current, latest version of the Graph API is v2.3. Apps calling v1.0 have until April 30, 2015 to upgrade to v2.0 or later.

To prevent broken experiences for people using your app, we strongly recommend to upgrade your apps to the latest current version at the earliest opportunity.

Roadmap
The full list of announced breaking changes and additions to our APIs and SDKs.

Changelog
See what has changed in Facebook's APIs and SDKs.

Versions and Migrations
Update your code to call a specific API version in order to get two years of stability for Core APIs. Use migrations to change the behaviour of an old API version.

Upgrade Guide
Detailed information about how to upgrade from older versions.

<https://developers.facebook.com/docs/graph-api>

Complex resource search

- Use *?parameter=value* for more advanced resource filtering (or search)
 - E.g.,
`https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=twitterapi&count=2`

Errors

- When errors or exceptions are encountered, use meaningful HTTP Status Codes
 - The Response Body may contain additional information (e.g., informational error messages)

```
{
  "developerMessage" : "Verbose, plain language description of
the problem for the app developer with hints about how to fix
it.",
  "userMessage": "Pass this message on to the app user if
needed.",
  "errorCode" : 12345,
  "more info": "http://dev.teachdogrest.com/errors/12345"
}
```

Authentication

Twitter Streaming API

```
Authorization: Basic aWh1YXJ00mFwaXM=
```

Amazon Web Services API

```
Authorization: AWS  
AKIAIOSFODNN7EXAMPLE:frJIUNo//y11qDzg=
```

Google API

```
Authorization: Bearer 1/ffBGRNJru1FQd44AzqT3Zg
```



Guidelines

- Design with standards in mind – for example RSS & ATOM
- Create should return URIs not resources
- Use the right HTTP methods for the right actions
- You are on HTTP – use the infrastructure.
 - Proxy, Caching, Etag, Expires

URL Design

Guidelines (1/2)

Plural nouns for collections	<code>/dogs</code>
ID for entity	<code>/dogs/1234</code>
Associations	<code>/owners/5678/dogs</code>
4 HTTP Methods	<code>POST GET PUT DELETE</code>
Bias toward concrete names	<code>/dogs</code> (not <code>animals</code>)
Multiple formats in URL	<code>/dogs.json</code> <code>/dogs.xml</code>
Paginate with limit and offset	<code>?limit=10&offset=0</code>
Query params	<code>?color=red&state=running</code>
Partial selection	<code>?fields=name,state</code>
Use medial capitalization	<code>"createdAt": 1320296464</code> <code>myObject.createdAt;</code>
Use verbs for non-resource requests	<code>/convert?from=EUR&to=CNY&amount=100</code>
Search	<code>/search?q=happy%2Blabrador</code>
DNS	<code>api.foo.com</code> <code>developers.foo.com</code>

Versioning

Include version in URL

`/v1/dogs`

Keep one previous version long enough for developers to migrate

`/v1/dogs`
`/v2/dogs`

Guidelines (2/2)

Errors

8 Status Codes

200 201 304 400 401 403 404 500

Verbose messages

`{"msg": "verbose, plain language hints"}`

Client Considerations

Client does not support HTTP status codes

`?suppress_response_codes=true`

Client does not support HTTP methods

`GET /dogs?method=post`
`GET /dogs`
`GET /dogs?method=put`
`GET /dogs?method=delete`

Complement API with SDK and code libraries

1. JavaScript
2. ...
3. ...



Web Architecture and Technologies

JSON

JAVASCRIPT OBJECT NOTATION



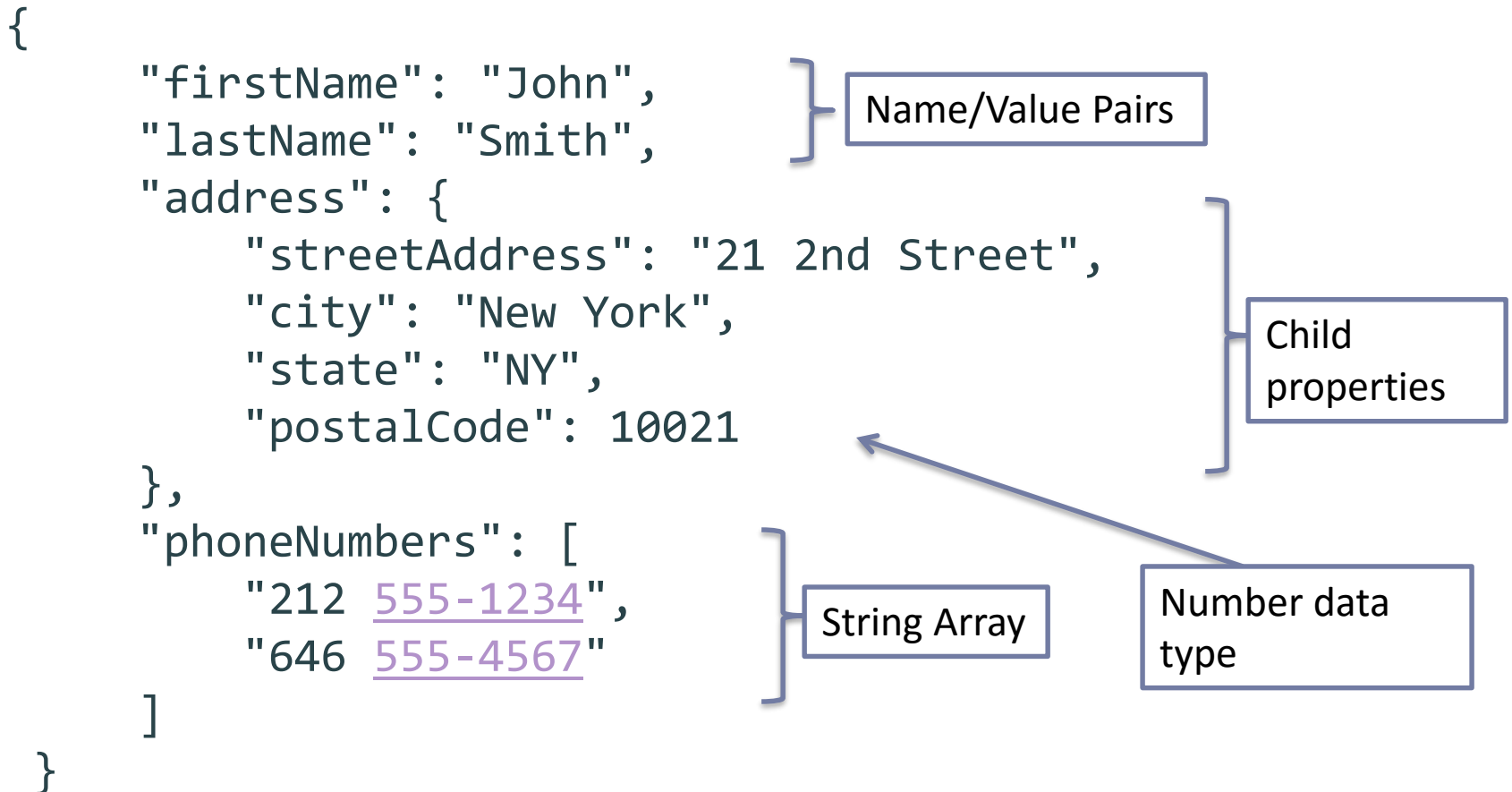
JSON – What is it?

- “JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate” – JSON.org
- Important: JSON is a subset of JavaScript

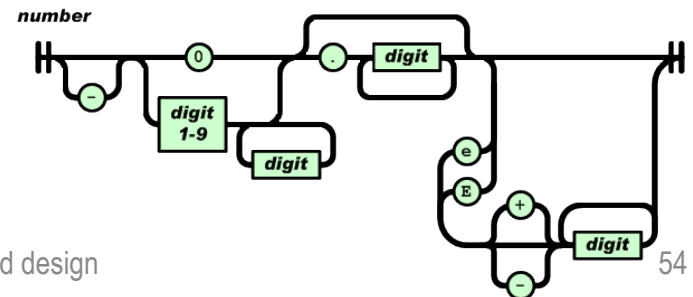
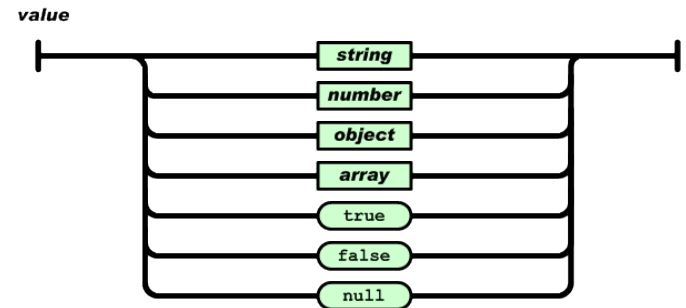
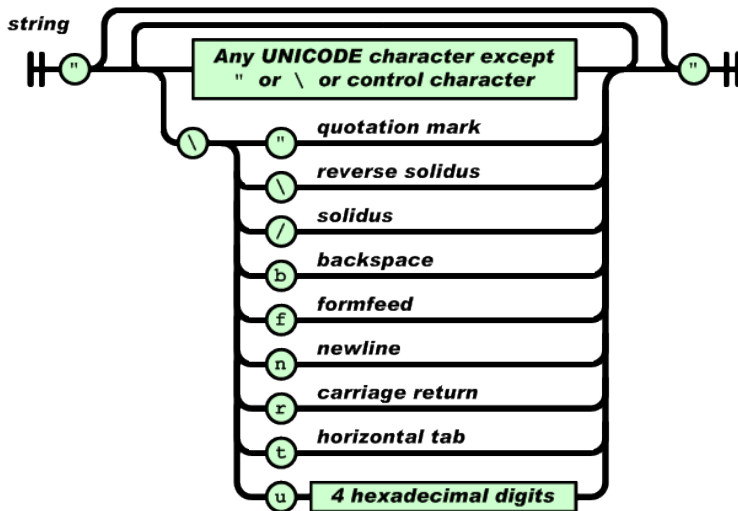
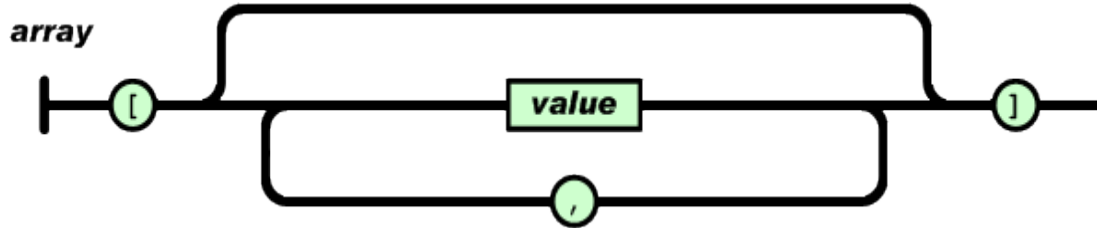
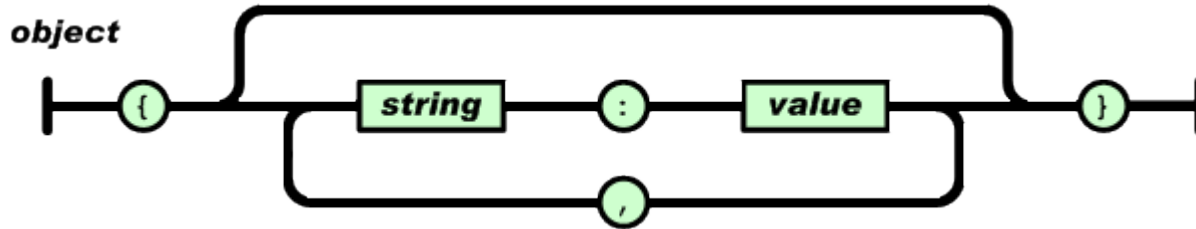
Json Logical Structure

- JSON is built on two structures:
 - A **collection** of name/value pairs. In various languages, this is realized as an ***object***, record, struct, dictionary, hash table, keyed list, or associative array. { ... }
 - An **ordered list** of values. In most languages, this is realized as an ***array***, vector, list, or sequence. [...]

JSON – What does it look like?



JSON Data Structures



Resources

- HTTP
 - <http://www.w3.org/Protocols/>
 - Hypertext Transfer Protocol -- HTTP/1.1:
<http://tools.ietf.org/html/rfc2616>
- REST
 - http://en.wikipedia.org/wiki/Representational_state_transfer
 - R. Fielding, Architectural Styles and the Design of Network-based Software Architectures,
<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
 - Learn REST: A Tutorial: <http://rest.elkstein.org/>
 - <https://pages.apigee.com/ebook-web-api-design-registration.html>
 - <http://www.slideshare.net/apigee/api-design-3rd-edition>
 - groups.google.com/group/api-craft

Resources

- JSON
 - <http://json.org>
 - ECMA-404 The JSON Data Interchange Standard.
<http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>

License



- These slides are distributed under a Creative Commons license “Attribution – NonCommercial – ShareAlike (CC BY-NC-SA) 3.0”
- You are free to:
 - Share — copy and redistribute the material in any medium or format
 - Adapt — remix, transform, and build upon the material
 - The licensor cannot revoke these freedoms as long as you follow the license terms
- Under the following terms:
 - Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - NonCommercial — You may not use the material for commercial purposes.
 - ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
 - No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- <http://creativecommons.org/licenses/by-nc-sa/3.0/>

