iPhone Programming

CMSC 498i – Spring 2010

Data & Web Services

Lecture #15 – Chuck Pisula

Today's Topics

- SQLite
- WebServices – XML, JSON

- Goals
  - Understand the basics (enough for the next lab…)
  - Allow you to understand which technologies to use
  - Just enough to make you dangerous!

Web Services

Your Application & The Cloud

- Web 2.0 apps with public API are common
- Many are exposed via “RESTful” interfaces returning XML or JSON

  `http://example.com/products?color=green`

- Examples
  - Google Data, Flickr, Twitter, Last.fm, Yahoo!
  - GeoNames.org, Weather.gov
- High level overview
  - Issuing requests
  - Parsing the results – XML, and JSON

RESTful

- In general, describes architectural style
- REST – Coined by Roy Fielding, “REpresentational State Transfer”
  - Web Page as an example
- In context of web development, use HTTP protocol
  - GET – receive representations in XML, or JSON
  - PUT, POST, DELETE – modify a resource
- We will focus on parsing of XML, JSON

Example Request - XML Response

http://www.parts.com/partlist?which=bargin_bin

```
<?xml version="1.0"?>
<parts-store>
   <part category=”Auto”>
      <PartID>314</PartID>
      <Name>Flux Capacitor</Name>
      <Description>Requires 1.21 Giga-Watts</Description>
      <PhotoLink>"http://www.parts.com/parts/314.jpg”<PhotoLink/>
      <Cost currency="USD">200.10</Cost>
   </part>
   <part category=”Dirigible”>
       ....
   </part>
</parts-store>
```

Example Request - JSON Response

http://www.parts.com/partlist?which=bargin_bin

```
{
  “auto-part” : {
       “part-id” : 314,
       “name” : “Flux Capacitor”,
       “description” : “Requires 1.21 Giga-Watts”,
       “cost-usd” : 200.10,
       “manufacturers” : [ “DrBrown”, “AlienInc” ];
   },
   “dirigible-part” : {
        ...
   }
}
```

XML & JSON

XML Parsing

- **Tree-based** – instantiate tree, the dig around for what you want
  - Maps XML into internal tree structure
  - Provides facilities to navigate and manipulate the tree
    - **DOM** – Document Object Model, structure traversal
    - **XPath**, XQuery – XML Path Language for querying and handling XML
    - Others…
- **Event–driven** – streaming parser sends “events” as it encounters elements, often called SAX-based parser;
  - Extremely efficient
  - Can be very complicated to manage parsing state

XML Parsing

- libxml2
  - Provides both tree-based, and event-driven parsing
  - Also provides a “Text Reader”, a simpler alternative to SAX-based
  - Included on iPhone
- Objective–C options
  - NSXMLParser – Event-driven parser only, included on iPhone
  - TouchXML – Tree-based, MIT licensed project on code.google.com
    - http://code.google.com/p/touchcode/wiki/TouchXML

XML Parsing

Event-based (SAX)

Event-based Parsing

- Series of parser events reported to client
  - Found start of an element
  - Found an attribute of an element
  - Found text inside an element
  - Found end of an element

Event-based Parsing

- Series of parser events reported to client
  - Found start of an element
  - Found an attribute of an element
  - Found text inside an element
  - Found end of an element
- NSXMLParser events sent to delegate

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
          namespaceURI:(NSString *)URI qualifiedName:(NSString *)qName
          attributes:(NSDictionary *)attributeDict;

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
          namespaceURI:(NSString *)URI qualifiedName:(NSString *)qName;

- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string;
```

Event-based Parsing

- Create a parser

```
- (id)initWithContentsOfURL:(NSURL *)url;
- (id)initWithData:(NSData *)data;
```

- Register delegate to receive parser events / errors

```
- (void)setDelegate:(id)delegate;
```

- Start parsing

```
- (void)parse;
```

- Stop when you have what you need

```
- (void)abortParsing;
```

Example

- Setup and start parsing

```
NSXMLParser *parser = [[NSXMLParser alloc] initWithContentsOfURL:...];
[parser setDelegate: self];
[parser parse];

if ([parser parserError]) {
    NSLog(@”ERROR: parser error %@”, [parser parseError]);
}
```

Example

- Create “PartInfo” objects with category/name parsed from XML

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
         namespaceURI:(NSString *)URI qualifiedName:(NSString *)qName
           attributes:(NSDictionary *)attrs
{



    // Watch for “parts” and create new Part objects


}

- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string {
    // Gather text if in the middle the “Name” node inside a “parts” node
}

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
        namespaceURI:(NSString *)URI qualifiedName:(NSString *)qName
{


    // If done parsing the name, use it


}
```

Example

- Create “PartInfo” objects with category/name parsed from XML

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
          namespaceURI:(NSString *)URI qualifiedName:(NSString *)qName
            attributes:(NSDictionary *)attrs
{
    if ([elementName isEqual:@“part”]) {
        partInfo = [[PartInfo alloc] initWithCategory: [attrs objectForKey:@”category”]];
    }
    if (partInfo && [elementName isEqual:@”Name”]) {
        parsedName = [[NSMutableString alloc] init];
    }
}

- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string {
    // Gather text if in the middle the “Name” node inside a “parts” node
}

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
        namespaceURI:(NSString *)URI qualifiedName:(NSString *)qName
{


    // If done parsing the name, use it


}
```

Example

- Create “PartInfo” objects with category/name parsed from XML

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
          namespaceURI:(NSString *)URI qualifiedName:(NSString *)qName
            attributes:(NSDictionary *)attrs
{
    if ([elementName isEqual:@“part”]) {
        partInfo = [[PartInfo alloc] initWithCategory: [attrs objectForKey:@”category”]];
    }
    if (partInfo && [elementName isEqual:@”Name”]) {
        parsedName = [[NSMutableString alloc] init];
    }
}

- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string {
    [parsedName appendString:string];
}

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
        namespaceURI:(NSString *)URI qualifiedName:(NSString *)qName
{


    // If done parsing the name, use it


}
```

Example

- Create “PartInfo” objects with category/name parsed from XML

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
          namespaceURI:(NSString *)URI qualifiedName:(NSString *)qName
            attributes:(NSDictionary *)attrs
{
    if ([elementName isEqual:@“part”]) {
        partInfo = [[PartInfo alloc] initWithCategory: [attrs objectForKey:@”category”]];
    }
    if (partInfo && [elementName isEqual:@”Name”]) {
        parsedName = [[NSMutableString alloc] init];
    }
}

- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string {
    [parsedName appendString:string];
}

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
        namespaceURI:(NSString *)URI qualifiedName:(NSString *)qName
{
    if (partInfo && [elementName isEqual:@”Name”]) {
        [partInfo setName: parsedName];
    }
    [parsedName release];
    parsedName = nil;
}
```

Another Example

- Gather all Photo Links

Another Example

- Gather all Photo Links

```
- (void)clearParsedElementText {
    [parsedText release];
    parsedText = [[NSMutableString alloc] init];
}

- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
         namespaceURI:(NSString *)URI qualifiedName:(NSString *)qName
           attributes:(NSDictionary *)attributeDict
{
    [self clearParsedElementText];
}

- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string {
    [parsedText appendString:string];
}

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
        namespaceURI:(NSString *)URI qualifiedName:(NSString *)qName
{
    if ([elementName isEqual:@"PhotoLink"]) {
        [photoLinks addObject:[NSURL urlWithString:parsedText]];
    }
    [self clearContentsOfElement];
}
```

Demo

Simple Image Search

Network Data

Options

NSURLRequest / Response

- *Alternative to initWithContentsOfURL:*
- Download the NSData yourself
- Create a NSURLRequest

```
NSMutableURLRequest *req =
   [NSMutableURLRequest requestWithURL:url
                            cachePolicy:NSURLRequestReloadIgnoringCacheData
                        timeoutInterval:30.0];
[req setHTTPMethod:@"GET"];
```

- Get data by asking for a NSURLResponse

```
NSHTTPURLResponse *urlResponse;
NSData *responseData = [NSURLConnection sendSynchronousRequest:req
                                                returningResponse:&urlResponse
                                                            error:&requestError];
if (requestError == nil) {
     int status = [httpResponse statusCode];
     if ((status >= 200) && (status < 300)) { // if the call was okay, parse!
          [self parseDocument:responseData];
     }
}
```

NSURLConnection

- Synchronous response loading
  - Example from previous slide…
  - Simple way to get data, and info about a response
- **Asynchronous** response callbacks (events)
  - Receive just the final results
  - Option to receive data as it is downloaded
  - Authentication challenge / response hooks
  - All sorts of events associated with request / response sequence

Asynchronous Example

- Create a request and connection

```
NSString *urlStr = [NSString stringWithFormat:/* http ..... */];
NSURL *url = [NSURL URLWithString:urlStr];

NSURLRequest *request = [NSURLRequest requestWithURL:url];
NSURLConnection *conn = [NSURLConnection connectionWithRequest:request delegate:self];
```

Asynchronous Example

- Create a request and connection

```
NSString *urlStr = [NSString stringWithFormat:/* http ..... */];
NSURL *url = [NSURL URLWithString:urlStr];

NSURLRequest *request = [NSURLRequest requestWithURL:url];
NSURLConnection *conn = [NSURLConnection connectionWithRequest:request delegate:self];
```

- Get data as it comes in and start parsing when done

```
- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data {
   // Making progress...
   NSMutableData *receivedData = [self receivedData];
   [receivedData appendData:data];
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
   // Got all the data, start parsing!
   NSXMLParser *parser = [[[NSXMLParser alloc] initWithData:[self receivedData]]
                              autorelease];
   [parser setDelegate:self];
   [parser parse];
}
```

Give “Activity” Feedback

- Status bar network activity indicator

```
[[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:YES];

... DO STUFF...

[[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:NO];
```

- Use UIActivityIndicatorView

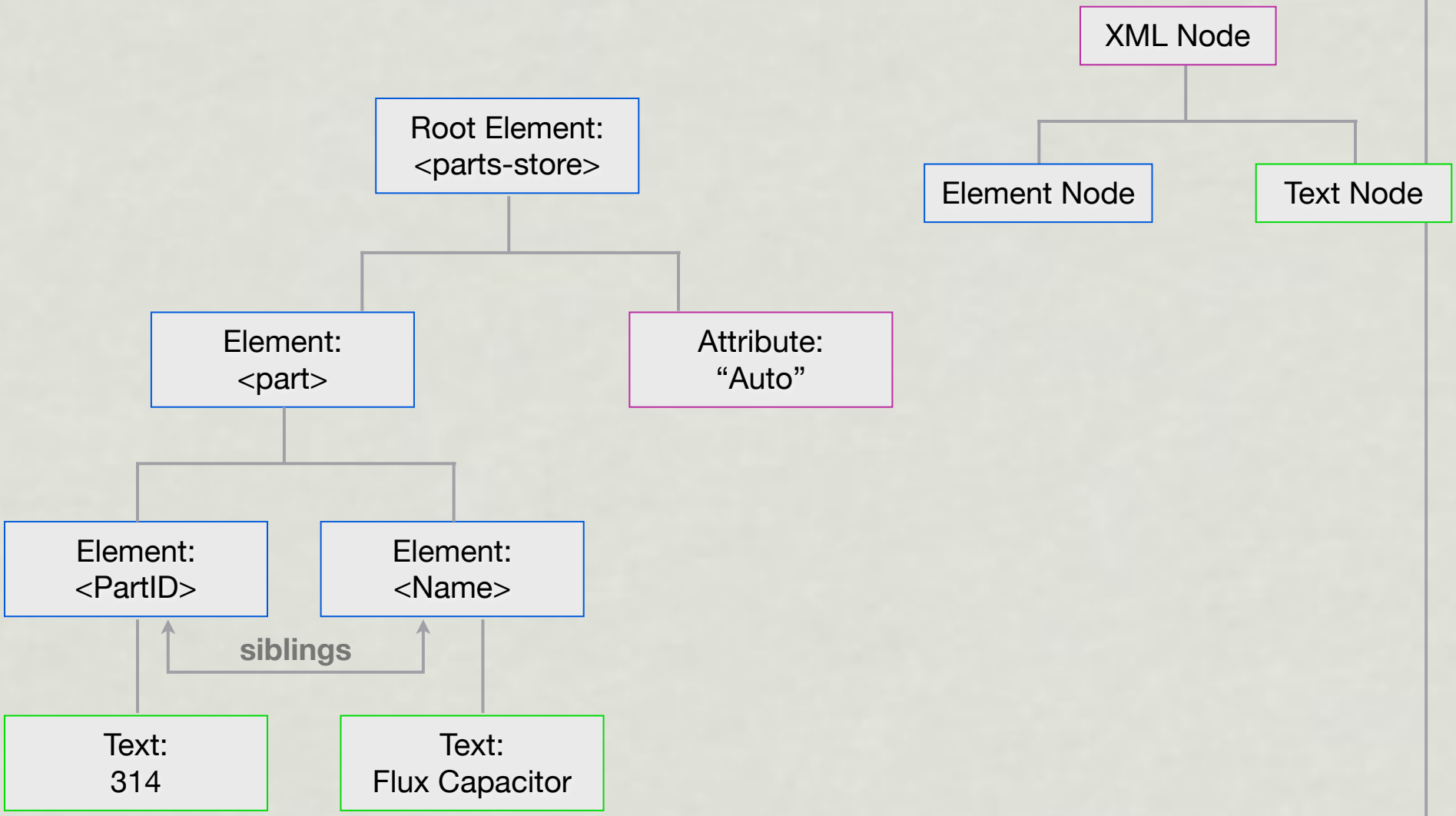
```
UIActivityIndicatorView *indicator = [[UIActivityIndicatorView alloc]
                            initWithActivityIndicatorStyle:UIActionSheetStyleDefault];

[self addSubview:indicator];
[indicator startAnimating];
```

XML Parsing

Tree-based

XML Tree Representation



Tree-based Parsing

- Tree traversal using Node properties
  - get children, parent
  - get next, previous siblings
  - access element name, attributes, string value
- XPath
  - Specify the node you want as a path
  - Example: //parts-store/part[@category=”Auto”]

```
<part category=”Auto”>
   <PartID>314</PartID>
   <Name>Flux Capacitor</Name>
   <Description>Requires 1.21 Giga-Watts</Description>
   <PhotoLink>"http://www.parts.com/parts/314.jpg”<PhotoLink/>
   <Cost currency="USD">200.10</Cost>
</part>
```

Tree-based Traversal Example

- libxml2 based example

```
#include <libxml/parser.h>
#include <libxml/tree.h>

xmlDoc *doc;
xmlNode *root_element, *curr;
```

- TouchXML probably easier to use…

Tree-based Traversal Example

- Using libxml2

```
// open and parse document, other create options (IO, Memory buffer, unix fd)
doc = xmlReadFile(argv[1], NULL, 0);

// Get the root element node
root_element = xmlDocGetRootElement(doc);
part_element = root_element->next;

while (part_element != NULL) {
    if ( 0==xmlStrcmp(part_element->name, (const xmlChar *)"part") )
    {
        const xmlChar *cat = xmlGetProp(part_element, (const xmlChar *)“category”);
        if ( 0==xmlStrcmp(cat, (const xmlChar *)”Auto”) )
        {
            parseAutoPart (doc, part_element);
        }
        else ...
    }
    part_element = part_element->next;
}

xmlFreeDoc(doc);
xmlCleanupParser(); // get rid of any global memory libxml2 created
```

Tree-based Traversal Example

- Using libxml2

```
// open and parse document, other create options (IO, Memory buffer, unix fd)
doc = xmlReadFile(argv[1], NULL, 0);

// Get the root element node
root_element = xmlDocGetRootElement(doc);
part_element = root_element->next;

while (part_element != NULL) {
    if ( 0==xmlStrcmp(part_element->name, (const xmlChar *)"part") )
    {
        const xmlChar *cat = xmlGetProp(part_element, (const xmlChar *)“category”);
        if ( 0==xmlStrcmp(cat, (const xmlChar *)”Auto”) )
        {
            parseAutoPart (doc, part_element);
        }
        else ...
    }
    part_element = part_element->next;
}

xmlFreeDoc(doc);
xmlCleanupParser(); // get rid of any global memory libxml2 created
```

Tree-based Traversal Example

- Using libxml2

```
// open and parse document, other create options (IO, Memory buffer, unix fd)
doc = xmlReadFile(argv[1], NULL, 0);

// Get the root element node
root_element = xmlDocGetRootElement(doc);
part_element = root_element->next;

while (part_element != NULL) {
    if ( 0==xmlStrcmp(part_element->name, (const xmlChar *)"part") )
    {
        const xmlChar *cat = xmlGetProp(part_element, (const xmlChar *)“category”);
        if ( 0==xmlStrcmp(cat, (const xmlChar *)”Auto”) )
        {
            parseAutoPart (doc, part_element);
        }
        else ...
    }
    part_element = part_element->next;
}

xmlFreeDoc(doc);
xmlCleanupParser(); // get rid of any global memory libxml2 created
```

Tree-based Traversal Example

- Using libxml2

```
// open and parse document, other create options (IO, Memory buffer, unix fd)
doc = xmlReadFile(argv[1], NULL, 0);

// Get the root element node
root_element = xmlDocGetRootElement(doc);
part_element = root_element->next;

while (part_element != NULL) {
    if ( 0==xmlStrcmp(part_element->name, (const xmlChar *)"part") )
    {
        const xmlChar *cat = xmlGetProp(part_element, (const xmlChar *)“category”);
        if ( 0==xmlStrcmp(cat, (const xmlChar *)”Auto”) )
        {
            parseAutoPart (doc, part_element);
        }
        else ...
    }
    part_element = part_element->next;
}

xmlFreeDoc(doc);
xmlCleanupParser(); // get rid of any global memory libxml2 created
```

Tree-based Traversal Example

- Using libxml2

```
// open and parse document, other create options (IO, Memory buffer, unix fd)
doc = xmlReadFile(argv[1], NULL, 0);

// Get the root element node
root_element = xmlDocGetRootElement(doc);
part_element = root_element->next;

while (part_element != NULL) {
    if ( 0==xmlStrcmp(part_element->name, (const xmlChar *)"part") )
    {
        const xmlChar *cat = xmlGetProp(part_element, (const xmlChar *)“category”);
        if ( 0==xmlStrcmp(cat, (const xmlChar *)”Auto”) )
        {
            parseAutoPart (doc, part_element);
        }
        else ...
    }
    part_element = part_element->next;
}

xmlFreeDoc(doc);
xmlCleanupParser(); // get rid of any global memory libxml2 created
```

XPath

- Instead of traversing, specify what you want

XPath

- Instead of traversing, specify what you want
- Examples
  - //part-store/part - will get you all the part nodes
  - //part-store/part[1] - get part at index 1
  - //part-store/part[@category=”Auto”] - get the part with a certain attribute

XPath Syntax

Expression	Result
`part-store`	selects all child nodes of “part-store” node
`/part-store`	select the root element “part-store”
`//xyz`	Select all “xyz” nodes no matter where they are…
`part-store/xyz`	select all “xyz” elements that are children of “part-store”
`part-store//xyz`	select all “xyz” elements that are descendants of “part-store”

XPath Syntax

Predicate	Result
`part[1]`	select the first part element
`part[last()]`	select the last part element
`part[position()>2]`	select all but the first two part elements
`part[@xyz='foo']`	select all part elements with xyz attribute value of 'foo'
`part[cost>200.00]`	select all part elements with cost element whose value is greater than 200
`part[cost>200.00]/name`	select title of all part elements with cost element whose value is greater than 200

- Go crazy... even more options exists…

XPath Example

- libxml also has XPath APIs (`xmlXPathEvalExpression`, ...)

XPath Example

- This example uses TouchXML

```
#import "CXMLDocument.h"
#import "CXMLElement.h"

CXMLDocument *xmlDocument =
         [[CXMLDocument alloc] initWithContentsOfURL:url options:0 error:nil];
```

- One option – specify exactly the node you want

```
CXMLElement *part = [[doc nodesForXPath:@"//part-store/part[@category=\”Auto\”]"
                                error:nil] objectAtIndex:0];
```

XPath Example

- This example uses TouchXML

```
#import "CXMLDocument.h"
#import "CXMLElement.h"

CXMLDocument *xmlDocument =
         [[CXMLDocument alloc] initWithContentsOfURL:url options:0 error:nil];
```

- Another option – find some nodes then enumerate

```
CXMLNode *store = [[xmlDocument nodesForXPath:@"//part-store" error:nil] objectAtIndex:0];
NSArray * allparts = [store nodesForXPath:@”./part”];

for (CXMLElement *part in allparts) {
    NSString *cat = [part attributeForName:@"category"];
    if ([cat isEqual:@"Auto"]) {
        // Process Auto Part...
    }
    else ...
}
```

JSON

JavaScript Object Notation

- More lightweight than XML
- Looks a lot like a property list
  - Arrays, dictionaries, strings, numbers
- Open source
  - From the makers of TouchXML ... comes TouchJSON
    - http://code.google.com/p/touchcode
  - json-framework wrapper for Objective-C
    - http://code.google.com/p/json-framework

Using JSON-framework

```
#import <JSON/JSON.h>
```

- Adds functionality to NSObject / NSString via Obj-C categories
- Convert from a plist value to JSON string

```
// NSObject.h
- (NSString *)JSONRepresentation;
```

- Convert from a JSON string to plist value `(NSArray or NSDictionary)`

```
// NSString+SBJSON.h
- (id)JSONValue;
```

JSON Example

```
{
  “auto-part” : {
       “part-id” : 314,
       “name” : “Flux Capacitor”,
        ...
```

```
#import <JSON/JSON.h>
```

JSON Example

```
{
  “auto-part” : {
       “part-id” : 314,
       “name” : “Flux Capacitor”,
        ...
```

```
#import <JSON/JSON.h>

NSString *jsonString = [NSString stringWithContentsOfURL:url];

NSDictionary *partsStore = [jsonString JSONValue];
```

JSON Example

```
{
  “auto-part” : {
       “part-id” : 314,
       “name” : “Flux Capacitor”,
        ...
```

```
#import <JSON/JSON.h>

NSString *jsonString = [NSString stringWithContentsOfURL:url];

NSDictionary *partsStore = [jsonString JSONValue];
NSDictionary *autoPart = [partsStore objectForKey:@”auto-part”];
```

JSON Example

```
{
  “auto-part” : {
       “part-id” : 314,
       “name” : “Flux Capacitor”,
        ...
```

```
#import <JSON/JSON.h>

NSString *jsonString = [NSString stringWithContentsOfURL:url];

NSDictionary *partsStore = [jsonString JSONValue];
NSDictionary *autoPart = [partsStore objectForKey:@”auto-part”];

NSString *partName = [autoPart objectForKey:@”name”];
```

SQLite

SQLite

- Complete SQL database in an ordinary file
  - No separate server process
  - A complete database contained in a single file
- Simple, Fast, Compact code size
  - Ideal for embedded devices
- Reliable – “ACID” Transactions
  - **Atomicity** – transactions are all or none
  - **Consistency** – only valid data committed, invalid data is rolled back
  - **Isolation** – concurrent transactions never see data in an inconsistent state
  - **Durability** – transactions survive system failure (e.g. transaction log)

SQLite

- Understands most of the standard SQL language
  - “SQL As Understood By SQLite” – http://www.sqlite.org/lang.html
- Interactive Command Line Tool
  - /usr/bin/sqlite3
- C/C++ APIs
- D. Richard Hipp
- Freely available source

SQLite

- Understands most of the standard SQL language
  - “SQL As Understood By SQLite” – http://www.sqlite.org/lang.html
- Interactive Command Line Tool
  - /usr/bin/sqlite3
- C/C++ APIs
- D. Richard Hipp
- Freely available source

Instead of a license, the SQLite source code offers a blessing:

May you do good and not evil
May you find forgiveness for yourself and forgive others
May you share freely, never taking more than you give.

When To Use SQLite

- Embedded devices
- Replacement for documents / file formats
- Low to medium traffic websites
- “Appropriate Uses for SQLite” – http://www.sqlite.org/whentouse.html

When Not To Use SQLite

- Multi-gigabyte databases
- High concurrency (multiple writers)
- Client-server applications
- “Appropriate Uses for SQLite” – http://www.sqlite.org/whentouse.html

Database – Basic Structure

Database – Basic Structure

- **Table** – The main element in a database – `TABLE`
- **Table Rows** – represent data entities (or records) – `ROW`
- **Table Columns** – represent attributes of an entity – `COLUMN`

Database – Basic Structure

- **Table** – The main element in a database – `TABLE`
- **Table Rows** – represent data entities (or records) – `ROW`
- **Table Columns** – represent attributes of an entity – `COLUMN`

- Visualizing The Main Structure
  - Table is like an OO Class
  - Row is like an OO instance of a class
  - Column is like an OO instance variable

Database – Basic Structure

- **Index** – Data structure that improves other operations – `INDEX`
  - B-Tree – unlike a simple “hash”, allows for fast selection of multiple contiguous items with a logarithmic maintenance costs
  - Costs memory
  - Costs time – maintaining `INDEX` leads to slower writes
- **Trigger** – Statements that automatically execute in response to certain events (insertion, deletion, updates) on a table
  - Example Uses
    - Timestamp records
    - Enhance performance – (e.g. maintain a total column that gets updated...)
    - Replicate data – (e.g. maintain a history)

Database – Basics Operations

- **`CREATE`** statement – create a table, index, or trigger
- **`DROP`** statement – delete a table, index, or trigger
- **`SELECT`** statement – extract data from a database table
  - **`WHERE`** predicates – predicate used to restrict query results by filtering on a columns value
  - **`JOIN`** clause – used to query data from two or more tables, tables are "joined" into a new virtual table using relations you specify
  - **`ORDER BY`** clause – used to sort the result set
- **`INSERT`** statement – insert data into a table
- **`UPDATE`** statement – change values in an existing table row
- **`DELETE`** statement – remove data from a table

Database – Basics

- Operators
  - **`AND, OR, =`** – The usual binary operators…
  - **`LIKE`** – Pattern matching comparison, useful for searching (e.g. `LIKE 'A%'`)
- Column Data Types
  - **`NULL`** – The value is a `NULL` value.
  - **`INTEGER`** – A signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
  - **`REAL`** – Floating point value, stored as an 8-byte IEEE floating point number
  - **`TEXT`** – A text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16-LE)
  - **`BLOB`** – Blob of bytes

Example SQL

- Using the command line SQLite interface

Example SQL

```
# sqlite3 /tmp/teams.db
```

Example SQL

```
# sqlite3 /tmp/teams.db

sqlite> CREATE TABLE team (TEAM_ID INTEGER PRIMARY KEY, name TEXT,
wins INTEGER, losses INTEGER);
```

Example SQL

```
# sqlite3 /tmp/teams.db

sqlite> CREATE TABLE team (TEAM_ID INTEGER PRIMARY KEY, name TEXT,
wins INTEGER, losses INTEGER);
```

Example SQL

```
# sqlite3 /tmp/teams.db

sqlite> CREATE TABLE team (TEAM_ID INTEGER PRIMARY KEY, name TEXT,
wins INTEGER, losses INTEGER);

sqlite> INSERT INTO team VALUES(1, "Packers", 11, 5);
```

Example SQL

```
# sqlite3 /tmp/teams.db

sqlite> CREATE TABLE team (TEAM_ID INTEGER PRIMARY KEY, name TEXT,
wins INTEGER, losses INTEGER);

sqlite> INSERT INTO team VALUES(1, "Packers", 11, 5);
sqlite> INSERT INTO team VALUES(1, "Ravens", 9, 7);
SQL error: PRIMARY KEY must be unique
```

Example SQL

```
# sqlite3 /tmp/teams.db

sqlite> CREATE TABLE team (TEAM_ID INTEGER PRIMARY KEY, name TEXT,
wins INTEGER, losses INTEGER);

sqlite> INSERT INTO team VALUES(1, "Packers", 11, 5);
sqlite> INSERT INTO team VALUES(1, "Ravens", 9, 7);
SQL error: PRIMARY KEY must be unique

sqlite> INSERT INTO team VALUES(2, "Ravens", 9, 7);
sqlite> INSERT INTO team VALUES(3, "Redskins", 4, 12);
sqlite> INSERT INTO team VALUES(4, "Colts", 14, 2);
```

Example SQL

```
# sqlite3 /tmp/teams.db

sqlite> CREATE TABLE team (TEAM_ID INTEGER PRIMARY KEY, name TEXT,
wins INTEGER, losses INTEGER);

sqlite> INSERT INTO team VALUES(1, "Packers", 11, 5);
sqlite> INSERT INTO team VALUES(1, "Ravens", 9, 7);
SQL error: PRIMARY KEY must be unique

sqlite> INSERT INTO team VALUES(2, "Ravens", 9, 7);
sqlite> INSERT INTO team VALUES(3, "Redskins", 4, 12);
sqlite> INSERT INTO team VALUES(4, "Colts", 14, 2 );

sqlite> SELECT * from team;

1 | Packers | 11 | 5
2 | Ravens | 9 | 7
3 | Redskins | 4 | 12
4 | Colts | 14 | 2
```

Example SQL

```
sqlite> .mode line
sqlite> SELECT * FROM team;

TEAM_ID = 1
   name = Packers
   wins = 11
 losses = 5

TEAM_ID = 2
   name = Ravens
   wins = 9
 losses = 7

TEAM_ID = 3
   name = Redskins
   wins = 4
 losses = 12

TEAM_ID = 4
   name = Colts
   wins = 14
 losses = 2
```

Example SQL

```
sqlite> .mode col
sqlite> .header on
sqlite> SELECT * FROM team;

TEAM_ID     name        wins        losses
----------  ----------  ----------  ----------
1           Packers     11          5
2           Ravens      9           7
3           Redskins    4           12
4           Colts       14          2
```

Example SQL

- Order the results of a `SELECT` statement

```
sqlite> SELECT name FROM team ORDER BY name;

Colts
Packers
Ravens
Redskins
```

```
sqlite> SELECT name FROM team ORDER BY name DESC;

Redskins
Ravens
Packers
Colts
```

Example SQL

- Select rows that satisfy a predicate expression

```
sqlite> SELECT name FROM team WHERE wins > 10;
Packers
Colts

sqlite> SELECT name FROM team WHERE wins > 10 ORDER BY wins DESC;
Colts
Packers

sqlite> SELECT name FROM team WHERE wins > 10 AND losses <= 2;
Colts
```

- Make your `SELECT` statements run faster

```
sqlite> CREATE INDEX team_wins_idx ON team (wins);
```

Example SQL

- Select rows matching a pattern

```
sqlite> SELECT name FROM team WHERE name LIKE 'R%';

Ravens
Redskins
```

```
sqlite> SELECT name FROM team WHERE name LIKE '%e%';

Packers
Ravens
Redskins
```

Example SQL

- Clean up when you’re done…

```
sqlite> DROP TABLE team;

sqlite> SELECT * FROM team;
SQL error: no such table: team
```

Primary Key

- Instead of making your own `PRIMARY KEY`

```
CREATE TABLE team (TEAM_ID INTEGER PRIMARY KEY, name TEXT, ...);
```

- Let SQLite pick an **auto incremented index**

```
CREATE TABLE team (name TEXT, ...);
```

  - SQLite adds a 64-bit **`ROWID INTEGER PRIMARY KEY`** if needed
  - `ROWID` for each row is unique among all rows in the same table
  - In example, “TEAM_ID” became an alias for “`ROWID`”
  - If you don’t specify `ROWID` during INSERT, SQLite automatically picks

```
INSERT INTO team VALUES("Packers", 11, 5);
```

SQLite

C APIs

SQLite C API Basics

- Simple to use from a C or C++ program
  - 3 core functions
  - one opaque data structure
  - small list of constants used as return values

SQLite C API Basics

- 3 Core Functions
  - Open

```
sqlite *sqlite_open(const char *dbname, int mode, char **err);
```

  - Close

```
void sqlite_close(sqlite *db);
```

  - Execute statement or query

```
int sqlite_exec( ...params... );
```

```
int sqlite3_prepare_v2(... params ...);
int sqlite3_step(...params...);
...etc...
```

SQLite C API Basics

- Opaque data structure representing database connection

```
typedef struct sqlite3 sqlite3;
```

```
sqlite3 *db; // your database connection
```

- Small list of constants used as return values

```
#define SQLITE_OK           0
#define SQLITE_ERROR        1
#define SQLITE_ROW        100
#define SQLITE_DONE       101
```

Executing SQL Statements

- Two Approaches

Executing SQL Statements

- Two Approaches
- Option 1
  - Call `sqlite3_exec( )` with your SQL statement as a c-string
  - Implement "**row handler**" callback, will be called for each row in result set
  - Easy to use, but a little limited – good for "fire and forget"

Executing SQL Statements

- Two Approaches
- Option 1
  - Call `sqlite3_exec( )` with your SQL statement as a c-string
  - Implement "**row handler**" callback, will be called for each row in result set
  - Easy to use, but a little limited – good for "fire and forget"
- Option 2 – prepared statements (*recommended*)
  - Precompile sql into a reusable statement with `sqlite3_prepare_v2( )`
  - **Bind** parameters, and repeatedly call "**step**" while there are more results
  - Finalize (sort of like `-release`) the precompiled statement

Executing SQL – (1) Exec

- sqlite3_exec requires 5 parameters

```
int sqlite3_exec(sqlite3 *db, const char *sql,
                 int (*callback)(void*,int,char**,char**),
                 void *context, char **error);
```

Executing SQL – (1) Exec

- sqlite3_exec requires 5 parameters

```
int sqlite3_exec(sqlite3 *db, const char *sql,
                 int (*callback)(void*,int,char**,char**),
                 void *context, char **error);
```

  - **db** – the sqlite database handle
  - **sql** – one or more SQL statements and/or queries to be processed
  - **callback** – function pointer; called once for each row in the result set
  - **context** – pointer passed as the “context” parameter to the callback

```
// Your callback
int callback(void *context, int count,
             char **values, char **columns);
```

  - **error** – if an error is encountered, descriptive info put here (may be NULL)

Executing SQL – (2) Stepping

- “Prepare a statement”

```
int sqlite3_prepare_v2(sqlite3 *db, const char *sql,.., sqlite3_stmt **ppStmt, ..
```

Executing SQL – (2) Stepping

- “Prepare a statement”

```
int sqlite3_prepare_v2(sqlite3 *db, const char *sql,.., sqlite3_stmt **ppStmt, ..
```

- Step through the result rows yourself (no callback!)

```
int sqlite3_step(sqlite3_stmt*);
```

Executing SQL – (2) Stepping

- "Prepare a statement"

```
int sqlite3_prepare_v2(sqlite3 *db, const char *sql,.., sqlite3_stmt **ppStmt, ..
```

- Step through the result rows yourself (no callback!)

```
int sqlite3_step(sqlite3_stmt*);
```

- Prepare for reuse, or free resources if done

```
int sqlite3_reset(sqlite3_stmt *pStmt); // prep for reuse
```

```
int sqlite3_finalize(sqlite3_stmt *pStmt); // release
```

Executing SQL – (2) Stepping

- "Prepare a statement"

```
int sqlite3_prepare_v2(sqlite3 *db, const char *sql,.., sqlite3_stmt **ppStmt, ..
```

- Step through the result rows yourself (no callback!)

```
int sqlite3_step(sqlite3_stmt*);
```

- Prepare for reuse, or free resources if done

```
int sqlite3_reset(sqlite3_stmt *pStmt); // prep for reuse
```

```
int sqlite3_finalize(sqlite3_stmt *pStmt); // release
```

- Benefits
  - More control over what happens while gathering results
  - Compile and parse once, use multiple times if you "Bind" parameters

An Example

- Simple Example With **“Bindings”**…

```
NSArray *names =  // an array of NSStrings
NSArray *wins = .., *losses = // an array of NSNumbers

sqlite3_stmt *stmt = sqlite3_prepare(db, “INSERT INTO teams VALUES(?, ?, ?)”);
```

An Example

- Simple Example With **“Bindings”**…

```
NSArray *names =  // an array of NSStrings
NSArray *wins = .., *losses = // an array of NSNumbers

sqlite3_stmt *stmt = sqlite3_prepare(db, “INSERT INTO teams VALUES(?, ?, ?)”);
```

An Example

- Simple Example With Bindings…

```
NSArray *names =  // an array of NSStrings
NSArray *wins = .., *losses = // an array of NSNumbers

sqlite3_stmt *stmt = sqlite3_prepare(db, “INSERT INTO teams VALUES(?, ?, ?)”);
if (stmt)
{
   for (i=0; i<[names count]; i++)
   {
      NSString *name = [names objectAtIndex:i];
      sqlite3_bind_text(stmt, 1, [name UTF8String], [name length], SQLITE_STATIC);

      sqlite3_bind_int(stmt, 2, [[wins objectAtIndex:i] intValue]);
      sqlite3_bind_int(stmt, 3, [[losses objectAtIndex:i] intValue]);









}
```

An Example

- Simple Example With Bindings…

```
NSArray *names =  // an array of NSStrings
NSArray *wins = .., *losses = // an array of NSNumbers

sqlite3_stmt *stmt = sqlite3_prepare(db, “INSERT INTO teams VALUES(?, ?, )”);
if (stmt)
{
   for (i=0; i<[names count]; i++)
   {
      NSString *name = [names objectAtIndex:i];
      sqlite3_bind_text(stmt, 1, [name UTF8String], [name length], SQLITE_STATIC);

      sqlite3_bind_int(stmt, 2, [[wins objectAtIndex:i] intValue]);
      sqlite3_bind_int(stmt, 3, [[losses objectAtIndex:i] intValue]);

      if (sqlite3_step(stmt)!=SQLITE_DONE) {
         NSLog(@”ERROR: couldn’t insert %@ into db”, name);
      }

      sqlite3_reset(stmt); // prepare for reuse!
   }

}
```

An Example

- Simple Example With Bindings…

```
NSArray *names =  // an array of NSStrings
NSArray *wins = .., *losses = // an array of NSNumbers

sqlite3_stmt *stmt = sqlite3_prepare(db, “INSERT INTO teams VALUES(?, ?, )”);
if (stmt)
{
   for (i=0; i<[names count]; i++)
   {
      NSString *name = [names objectAtIndex:i];
      sqlite3_bind_text(stmt, 1, [name UTF8String], [name length], SQLITE_STATIC);

      sqlite3_bind_int(stmt, 2, [[wins objectAtIndex:i] intValue]);
      sqlite3_bind_int(stmt, 3, [[losses objectAtIndex:i] intValue]);

      if (sqlite3_step(stmt)!=SQLITE_DONE) {
         NSLog(@”ERROR: couldn’t insert %@ into db”, name);
      }

      sqlite3_reset(stmt); // prepare for reuse!
   }

}
```

An Example

- Simple Example With Bindings…

```
NSArray *names =  // an array of NSStrings
NSArray *wins = .., *losses = // an array of NSNumbers

sqlite3_stmt *stmt = sqlite3_prepare(db, “INSERT INTO teams VALUES(?, ?, )”);
if (stmt)
{
   for (i=0; i<[names count]; i++)
   {
      NSString *name = [names objectAtIndex:i];
      sqlite3_bind_text(stmt, 1, [name UTF8String], [name length], SQLITE_STATIC);

      sqlite3_bind_int(stmt, 2, [[wins objectAtIndex:i] intValue]);
      sqlite3_bind_int(stmt, 3, [[losses objectAtIndex:i] intValue]);

      if (sqlite3_step(stmt)!=SQLITE_DONE) {
         NSLog(@”ERROR: couldn’t insert %@ into db”, name);
      }

      sqlite3_reset(stmt); // prepare for reuse!
   }
   sqlite3_finalize(stmt); // were done with stmt!
}
```

Result Values From A Query

- Simple Example

```
stmt = sqlite3_prepare(db, “SELECT name, wins FROM ITEM WHERE wins > 10”);

if (stmt)
{
   while (sqlite_step(stmt) == SQLITE_ROW)
   {
      char *cstrName = sqlite3_column_text(stmt, 0);
      if (cstrName == NULL) continue; // unexpected, but lets be safe

      NSString *name = [NSString stringWithUTF8String:cstrName];
      int wins = sqlite3_column_int(stmt, 1);

      NSLog(@”team = %@, wins = %d”, name, wins);
   }

   sqlite3_reset(stmt); // not really necessary, we aren’t reusing...
   sqlite3_finalize(stmt);
}
```

Result Values From A Query

- Simple Example

```
stmt = sqlite3_prepare(db, “SELECT name, wins FROM ITEM WHERE wins > 10”);

if (stmt)
{
   while (sqlite_step(stmt) == SQLITE_ROW)
   {
      char *cstrName = sqlite3_column_text(stmt, 0);
      if (cstrName == NULL) continue; // unexpected, but lets be safe

      NSString *name = [NSString stringWithUTF8String:cstrName];
      int wins = sqlite3_column_int(stmt, 1);

      NSLog(@”team = %@, wins = %d”, name, wins);
   }

   sqlite3_reset(stmt); // not really necessary, we aren’t reusing...
   sqlite3_finalize(stmt);
}
```

Result Values From A Query

- Simple Example

```
stmt = sqlite3_prepare(db, “SELECT name, wins FROM ITEM WHERE wins > 10”);

if (stmt)
{
   while (sqlite_step(stmt) == SQLITE_ROW)
   {
      char *cstrName = sqlite3_column_text(stmt, 0);
      if (cstrName == NULL) continue; // unexpected, but lets be safe

      NSString *name = [NSString stringWithUTF8String: cstrName];
      int wins = sqlite3_column_int(stmt, 1);

      NSLog(@”team = %@, wins = %d”, name, wins);
   }

   sqlite3_reset(stmt); // not really necessary, we aren’t reusing...
   sqlite3_finalize(stmt);
}
```

Result Values From A Query

- Simple Example

```
stmt = sqlite3_prepare(db, “SELECT name, wins FROM ITEM WHERE wins > 10”);

if (stmt)
{
   while (sqlite_step(stmt) == SQLITE_ROW)
   {
      char *cstrName = sqlite3_column_text(stmt, 0);
      if (cstrName == NULL) continue; // unexpected, but lets be safe

      NSString *name = [NSString stringWithUTF8String: cstrName];
      int wins = sqlite3_column_int(stmt, 1);

      NSLog(@”team = %@, wins = %d”, name, wins);
   }

   sqlite3_reset(stmt); // not really necessary, we aren’t reusing...
   sqlite3_finalize(stmt);
}
```

Result Values From A Query

- Simple Example

```
stmt = sqlite3_prepare(db, “SELECT name, wins FROM ITEM WHERE wins > 10”);

if (stmt)
{
   while (sqlite_step(stmt) == SQLITE_ROW)
   {
      char *cstrName = sqlite3_column_text(stmt, 0);
      if (cstrName == NULL) continue; // unexpected, but lets be safe

      NSString *name = [NSString stringWithUTF8String:cstrName];
      int wins = sqlite3_column_int(stmt, 1);

      NSLog(@”team = %@, wins = %d”, name, wins);
   }

   sqlite3_reset(stmt); // not really necessary, we aren’t reusing...
   sqlite3_finalize(stmt);
}
```

Thread Safety

- From http://www.sqlite.org/faq.html – "Threads are evil. Avoid them"
  - What they mean is … be very careful …
  - `sqlite3 *db` can be used in multiple threads if you're careful
  - See SQLite.org for more information
- To be thread–safe, SQLite must be compiled with the SQLITE_THREADSAFE preprocessor macro set to 1
- A better option – create a new database connection for each thread (e.g. one `sqlite3_open( )` for each thread)

Resources – SQL

- Language Reference – “Sqlite As Understood By SQLite”

  http://www.sqlite.org/lang.html

- C-APIs – “Intro to the SQLite C Interface”

  http://www.sqlite.org/cintro.html

Resources – XML / JSON

- Event-Driven XML Programming Guide for Cocoa
  - Parser and Capabilities section
  - Handling XML Elements and Attributes (good NSXMLParser example)
- Libxml tutorial – http://xmlsoft.org/tutorial/xmltutorial.pdf
- NSXMLParser example – http://weblog.bignerdranch.com/?p=48
- XPath – http://www.w3schools.com/Xpath/xpath_syntax.asp
- “Introducing JSON” – http://www.json.org/