MSSE 652: iOS Enterprise Software Development

Topic 2: Web Services

Agenda

- Resources
- Introduction
- Web Services
- iOS Web Services
- Grand Central Dispatch
- Object blocks
- NSURLConnection (and friends)

Resources

- Online resources
  - http://www.w3.org/DesignIssues/WebServices.html
  - http://www.infoq.com/articles/rest-introduction
  - http://fusesource.com/docs/esb/4.2/rest/RESTIntro.html
  - http://www.restapitutorial.com/index.html
  - http://rest.elkstein.org/2008/02/what-is-rest.html
  - http://mobile.tutsplus.com/tutorials/iphone/ios-quick-tip-interacting-with-web-services/?search_index=10

Resources (cont'd)

- iOS resources: NSURLConnection
  - https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/URLLoadingSystem/Tasks/UsingNSURLConnection.html
  - http://codewithchris.com/tutorial-how-to-use-ios-nsurlconnection-by-example/
  - http://stackoverflow.com/questions/8515667/how-to-send-asynchronous-url-request
  - http://yuvarajmanickam.wordpress.com/2012/10/17/nsurlconnection-basics-for-ios-beginners/

Introduction

- Often there's a need for an application to exchange information with another application
  - e.g., a server on a well known IP address
- The two fundamental techniques for app to app communication are …
  - sockets
    - where each application instantiates a socket and then performs IO over the socket
  - web services
    - a higher-level abstraction that rides on top of sockets

Our focus

Web Services

- Web services facilitate process-to-process communication that allows applications to …
  - share resources (information)
  - invoke each other's functionality
- Two forms of web services exist:
  - SOAP (simple object access protocol)
    - based on a heavy-weight implementation (WSDL)
  - REST (representational state transfer)
    - based on a light-weight implementation (WADL)

Our Focus

RESTful Web Services

- REST: Representational State Transfer
  - an “architectural style” that’s based on web-standards and specifically HTTP, JSON, and XML

- Basic concept:
  - a REST server provides access to resources
    - think of a resource as information (e.g., university courses)
  - client apps use HTTP to access the resources
    - using standard HTTP methods GET, PUT, POST, DELETE

From an architecture perspective we have the following …

REST Architecture

Clients creating, requesting, updating, deleting information

- Clients:

Network / Internet

HTTP Protocol

- Servers

Servers hosting informational resources

REST Characteristics

- REST Web Services are characterized by …
  - a uniform interface
    - based on the HTTP standard
  - client-server
    - a request-response paradigm (handshake)
  - stateless
    - each client request “stands alone” (independent of other requests)
  - cacheable
    - clients may cache the response returned by the server
  - layered system
    - clients and servers do not need to be directly connected

REST in a Nutshell …

- Clients submit requests to servers specifying …
  - the HTTP operation to be performed (i.e., the action)
    - e.g., POST, GET, PUT, DELETE
  - the URI (uniform resource identifier) for the service
    - e.g., somedomainname/somepath/someresourcename
  - the supported MIME-type to be returned to the client
    - e.g., JSON, XML, Text
- Servers perform the requested operation and return a response (using the requested MIME-type)

HTTP

- HTTP: the Hyper Text Transfer Protocol
  - the primary Internet protocol for web applications
  - based on a request/response interaction
    - clients send a request … servers return a response
- Structure of HTTP request / response messages
  - the first line is the “request/response line”
  - followed by various headers (having keyword values pairs; e.g., the MIME-type)
  - followed by a blank line
  - followed by the body of the request/response

For instance …

HTTP Request / Response

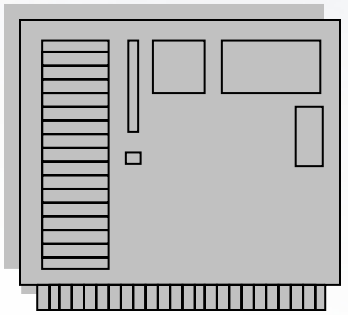
Clients

HTTP request

request line
headers

body

HTTP Web Server



HTTP response

response line
headers

body

URIs

- Uniform Resource Identifiers
  - are used to identify the resource of the request
- General form of URI:
  - domainname/pathtoresource/resourcename
- Example:
  - acme.com/products/books/gonewiththewind

domain name | path to resource | resource

HTTP Methods

- REST supports the following HTTP methods
  - POST
    - creates a new resource on the server
  - GET
    - gets a resource from the server
  - PUT
    - updates a resource on the sever
  - DELETE
    - deletes a resource on the server

Sounds like CRUD stuff …

A bunch of CRUD ☺

- HTTP methods support CRUD operations

HTTP Method	CRUD Operation
POST	*Create* a new resource
GET	*Retrieve* a resource
PUT	*Update* a resource
DELETE	*Delete* a resource

Data Formats

- Two data-exchange formats are typically used:
  - XML (eXtensible Markup Language)
    - a verbose markup language, using tags: <thisisatag> to “markup” the information
    - originated in the mid 90s

pronounced “Jason”

  - JSON (JavaScript Object Notation)
    - a relatively lightweight notation
      - more concise than XML, and faster/easier to parse
    - originated in ~2002

Note: both are highly used today, but the trend is toward JSON

XML – brief overview

- eXtensible Markup Language
  - text, human readable
  - MIME type in HTTP header: *application/xml* and *text/xml*
  - note: with XML the information is “marked up”
    - i.e., the data is surrounded by tags denoted by < … >
      - where each tag starts with a “<“ and ends with a “>”
        - » and has a meaningful name; e.g., <program>
  - extensible
    - the XML markup tags are defined by developers as needed
      - e.g., to “markup” university program information, you may have …
        - » <program> … </program>

this tag marks up “program” information

start tag

end tag

XML formatted “program” data

- An XML list of “university program” information

```
<programs>
    <program>
        <id>1</id>
        <name>CIS</name>
    </program>
    <program>
        <id>2</id>
        <name>CN</name>
    </program>
    …
<programs>
```

Start tag for a list (array) of programs

start tag for a program

contains the program’s id: 1

contains the program’s name: CIS

end tag for a program

the start of another program

End tag for a list (array) of programs

JSON – brief overview

- JavaScript Object Notation
  - text, human readable
  - MIME type in HTTP header: application/json
  - used with Google Search, Yahoo!, Flickr, Facebook

- Example: a university program,
  - where each program has an id and a name

```
{"id":1,"name":"CIS"}
```

The object starts with {

The object ends with }

JSON formatted “program” list

the [symbol denotes the start of an array


```
[{"id":1,"name":"CIS"},
{"id":2,"name":"CN"},
{"id":3,"name":"CS"},
{"id":4,"name":"MSCC"},
{"id":5,"name":"MCT"},
{"id":6,"name":"MSCD"},
{"id":7,"name":"MSCI"},
{"id":8,"name":"MSCT"},
{"id":9,"name":"MSIA"},
{"id":10,"name":"MSSE"},
{"id":11,"name":"MT"}]
```


11 objects, each …

- denoted by { }
- having attributes:
  - id
  - name

the] symbol denotes the end of an array

Please note the following …

- A deployed web service may support …
  - only XML returned data
  - only JSON returned data
  - or both XML and JSON returned data
    - but not in the same request (see below)
      - the service returns either XML or JSON in each request
- When a service supports both formats, …
  - a client can identify a particular format by specifying the "MIME type" in the HTTP request header
    - more on this later

iOS and REST

- There are a number of techniques for integrating REST web services into your iOS application
  - using capabilities in iOS
    - a relatively simple approach; see the following charts

Our focus

  - using a 3rd Party product
    - provided by external, open source libraries
      - that need to be downloaded and installed in Xcode
        - » e.g., AFNetworking, RestKit
      - discussed in later topics

iOS and REST (cont'd)

- The following techniques are provided with iOS
  - using NSString's method stringWithContentsOfURL
    - constructs an NSString with the content returned from a web site
  - using a combination of NSMutableURLRequest, NSURLConnection, NSURLResponse
- Both techniques are illustrated in the following charts
  - along with Grand Central Dispatch (GCD)

iOS REST using NSString

- iOS provides the following technique for retrieving information from a REST web service

  [NSString stringWithContentsOfURL …]

  – The static method stringWithContentsOfURL is used to create an NSString containing the contents of the requested Web resource
    - thus, if the URL is a Web service, the created string will contain the **response** as returned from the web service

Really? Is it that easy? …

For instance …

- Suppose there's a web resource located at …
  - http://example.com
- Then, here's how you retrieve the web resource using NSError, NSURL, and NSString

```
NSError *error = nil;

NSURL *url = [NSURL URLWithString:@"http://example.com"];

NSString *response = [NSString stringWithContentsOfURL:url
                                              encoding:NSASCIIStringEncoding
                                                 error:&error];
```

The error string, if one occurs

The URL of the resource

The response

The URL

The error string

But wait, there's more …

Parsing the response

- To make sense of the response data, it needs to be parsed
- As mentioned previously, web services typically return data in one of two formats:
  - JSON
    - which can be parsed using the class NSJSONSerialization
  - XML
    - which can be parsed using the class NSXMLParse

Our focus

First, let's look at NSJSONSerialization …

Parsing a JSON response msg

- If the web service is returning JSON data, it can be parsed with iOS class NSJSONSerialization
  - the results are placed in an NSArray (or NSDictionary)
- For instance …

```
NSError *error = nil;
NSData *jsonData = [response
      dataUsingEncoding:NSASCIIStringEncoding];
NSArray *array = [NSJSONSerialization
      JSONObjectWithData:response  options:kNilOptions  error:&error];
```

The error string, if one occurs

The parsed data

The data returned from the web service

Iterating thru the parsed data

- Here's how you iterate thru parsed JSON data …

```
…
NSArray *array = [NSJSONSerialization
      JSONObjectWithData:response  options:kNilOptions  error:&error];
for (int i=0; i<array.count; i++) {
        NSLog(@"string: %@", array[i]);
        NSDictionary *pgm = array[i];
        for (id key in pgm) {
            id value = [pgm objectForKey:key];
            NSLog(@"key: %@, value: %@", key, value);
            …
        }
}
```

Get an object and iterate thru its keys

e.g., here you could place the data in a domain object

A working example …

- The following charts illustrate a working example of the previously discussed code snippets
- But please note:
  - the web service is running on the same machine that's running the Xcode simulator
    - hence, the domain name is "localhost"
      - which is an alias for IP address 127.0.0.1
  - also, note the use of port 8080
    - normally web services are on port 80
      - which is the default port in an HTTP request

Combining the code

Accessing a web service on the local machine using port 8080

```
NSError *error = nil;
NSURL *url = [NSURL URLWithString:@"http://localhost:8080/SCIS/webresources/domain.programs"];
NSString *response = [NSString stringWithContentsOfURL:url
                                              encoding:NSASCIIStringEncoding
                                                 error:&error];

if(!error) {
    NSLog(@"\nJSON: %@", response);
    NSData *jsonData = [response dataUsingEncoding:NSASCIIStringEncoding];
    NSArray *array = [NSJSONSerialization JSONObjectWithData:jsonData
                                                     options:kNilOptions
                                                       error:&error];

    NSLog(@"The contents of the array");
    for (int i=0; i<array.count; i++) {
        NSLog(@"program: %@", array[i]);
        NSString *item = array[i];
        NSLog(@"string: %@", item);

        NSDictionary *pgm = array[i];
        for(id key in pgm) {
            id value = [pgm objectForKey:key];
            NSLog(@"key: %@, value: %@", key, value);
        }
    }
} else {
    NSLog(@"Error: %@", error);
}
```

Invoking the web service

Invoking the parser

Iterating thru the data

Asynchronous Operations

- Please note the following:
  - as a developer, you need to be aware of tasks that can "potentially" take a long time to complete
    - e.g., invoking a remote web service
  - Why? because …
    - if the task is executed in the "main event loop", it can "freeze" (i.e., block) the UI
      - and if the UI is blocked, users will not be happy ☹
  - Hmmmmmmm, what can we do? Enter GCD …

Multi Tasking with GCD

- To address this problem, we make use of GCD: Grand Central Dispatch …
  - an iOS thread manager which manages “dispatch queues” for running tasks in the background
- So in a nutshell, here’s the idea …
  - you place your task (i.e., a web service call) in a GCD managed queue to await execution in the background
    - the main event loop can then continue to respond to user events (so your users are happy)

Benefits of GCD dispatch queues

- From developer.apple.com, the benefits include …
  - a straightforward and simple programming interface
  - thread pool management
  - an efficient use of memory since the thread stacks do not live in application memory
  - the asynchronous dispatching of tasks to a dispatch queue which cannot deadlock the queue
  - gracefully scalling
  - a more efficient alternative to locks and other synchronization primitives

One more thing: block objects …

Block objects

- Block objects are a “C” language feature similar to “function pointers”
- In essence, a block is collection of statements that can be …
  - named
  - passed parameters (arguments)
  - return a value
  - referenced for execution

  Similar to C functions
  - passed into other functions (for execution)

Unnamed blocks (what we will use)

- Unnamed blocks can also be defined
  - unnamed blocks are used to define a block “inline”
    - that is, they are defined at the point of use (e.g., GCD)
- General syntax for an unnamed block

note: the caret symbol denotes the start of a block

```
^{
    //statements that belong to the block go here
};
```

So how are unnamed blocks used with GCD? …

Launching your background task

- To place a task in the GCD event queue, you use:
  - the “C” method dispatch_async
  - a block object

place your task in a queue

an inline object block

```
dispatch_async(dispatch_get_global_queue
        (DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{

        //TODO: your web service call goes here

});
```

Define the steps of your task in an object block

Combining the async code

Launching an asynchronous task

start of inline object block

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    NSError *error = nil;
    NSURL *url = [NSURL URLWithString:@"http://localhost:8080/SCIS/webresources/domain.programs"];
    NSString *response = [NSString stringWithContentsOfURL:url
                                                  encoding:NSASCIIStringEncoding
                                                     error:&error];
    if(!error) {
        NSLog(@"\nJSON: %@", response);
        NSData *jsonData = [response dataUsingEncoding:NSASCIIStringEncoding];
        NSArray *array = [NSJSONSerialization JSONObjectWithData:jsonData
                                                         options:kNilOptions
                                                           error:&error];
        NSLog(@"The contents of the array");
        for (int i=0; i<array.count; i++) {
            NSLog(@"program: %@", array[i]);
            NSString *item = array[i];
            NSLog(@"string: %@", item);

            NSDictionary *pgm = array[i];
            for(id key in pgm) {
                id value = [pgm objectForKey:key];
                NSLog(@"key: %@, value: %@", key, value);
            }
        }
    } else {
        NSLog(@"Error: %@", error);
    }
});
```

the } denotes the end of object block

But wait, there's more …

Updating the UI

- What happens when the background task completes?
  - we typically want to update the UI
    - e.g., with information returned from the web service
- Question: can the background task update the UI?
  - **No, never, absolutely NOT**! ☹
    - Why is that? …

Updating the UI (cont'd)

- The only thread that should update the UI is the iOS "main event loop"
  - if any other thread updates the UI, the app can/will crash

- So what do we do?
  - we have our background task (on the previous page) insert a "UI update task" in the main event queue
    - where it's safe to update the UI

Basic code pattern to update UI

- To update the UI from a background task, you insert another task in the “main queue” …

Launch worker task

worker task in unnamed block

```
dispatch_async(dispatch_get_global_queue
    (DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
        //TODO: background processing goes here

        dispatch_async(dispatch_get_main_queue(), ^{
            //TODO: update UI here
        });
});
```

nested UI update task in unnamed block

All together now …

Launch worker task

start of inline object block

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    NSError *error = nil;
    NSURL *url = [NSURL URLWithString:@"http://localhost:8080/SCIS/webresources/domain.programs"];
    NSString *response = [NSString stringWithContentsOfURL:url
                                                  encoding:NSASCIIStringEncoding
                                                     error:&error];

    if(!error) {
        NSLog(@"\nJSON: %@", response);
        NSData *jsonData = [response dataUsingEncoding:NSASCIIStringEncoding];
        NSArray *array = [NSJSONSerialization JSONObjectWithData:jsonData
                                                         options:kNilOptions
                                                           error:&error];
        NSLog(@"The contents of the array");
        for (int i=0; i<array.count; i++) {
            NSLog(@"program: %@", array[i]);
            NSString *item = array[i];
            NSLog(@"string: %@", item);

            NSDictionary *pgm = array[i];
            for(id key in pgm) {
                id value = [pgm objectForKey:key];
                NSLog(@"key: %@, value: %@", key, value);
            }
        }

        dispatch_async(dispatch_get_main_queue(), ^{
            // code for updating the UI goes here
        });

    } else {
        NSLog(@"Error: %@", error);
    }
});
```

Launch UI update task

For instance, …

- Suppose the web service is used to retrieve data for a table view
  - how would you trigger the table view to reload itself?
    - by invoking the method “reloadData”
- Example:
  - assuming there’s an IBOutlet named programsTable
    - then the code to trigger the reload of the table is …

      [_programsTable reloadData];

  - see following chart

Invoking reloadData

- In the task to update the UI, …

```
dispatch_async(dispatch_get_main_queue(), ^{
    NSLog(@"In thread to update the UI");

    [_programsTable reloadData];

    NSLog(@"programsTable updated");
});
```

Reload the data in the table view

In summary …

- The previous charts have illustrated the use of …
  - NSString to retrieve the contents of a remote web service
    - using the static method URLWithString
  - Grand Central Dispatch to …
    - run the web service invocation as a background task
      - using an inline object block
    - update the UI in a second background task
      - using a nested inline object block executed in the “main event loop”
- But iOS provides another technique too …

Another iOS web service technique

- In addition to the NSString approach for accessing web services, iOS provides another approach …
  - NSMutableURLRequest
    - contains the request information (e.g., the url)
  - NSURLConnection
    - performs the web service call using the information in the above request
  - NSURLResponse
    - contains the response returned from the web service

But NSString is so simple; why use the above classes?

Why use NSMutableURLRequest? …

- Simply because it offers you more control
  - For instance, you can specify …
    - the HTTP method
      - POST, GET, PUT, DELETE
    - HTTP headers
      - including MIME-type
        - » which dictates whether JSON or XML is returned in the response
    - and whether the request is synchronous or asynchronous
      - note: if it's asynchronous, you don't need to use GCD ☺
  - Also, it can be used for both REST & SOAP web services (but for now, our focus is on REST)

NSMutableURLRequest

- To use NSMutableURLRequest, you …
  - first create an NSURL with the targeted url information
  - and then use the above NSURL to initialize the request

- For example:

Create the url

```
NSURL *url = [NSURL URLWithString:
                @" http://localhost:8080/SCIS/webresources/domain.programs"];
NSMutableURLRequest theRequest =
                [[NSMutableURLRequest alloc] initWithURL:url];
```

Create the request using the url

NSURLConnection

- Once you have an NSMutableURLRequest …
  - you use NSURLConnection to invoke the service
- But first, please note:
  - NSURLConnection supports both …
    - synchronous requests
      - that should be spawned with GCD (as illustrated with NSString)
    - asynchronous requests
      - that don't require the use of GCD

Synchronous requests

- To invoke a synchronous request, use …
  - NSURLConnection sendSynchronousRequest
- Example:

```
NSURL *url =[NSURL URLWithString:@"http://localhost:8080/SCIS/webresources/domain.programs"];
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];
[request setHTTPMethod:@"GET"];
NSURLResponse * response = nil;
NSError * error = nil;
NSData * data = [NSURLConnection sendSynchronousRequest:request
                                      returningResponse:&response
                                                  error:&error];

if(error == nil)
{
    // parse the data here
}
else
{
    NSLog(@"error: %@", error);
}
```

Note: all this code should be placed in dispatch_async as illustrated with NSString

Asynchronous requests

- To invoke an asynchronous request, you need to do the following:
  - update the enclosing class interface to conform to NSURLConnectionDelegate
  - update the enclosing class impl with …
    - with a variable to hold the response data
    - the delegate method implementations
    - NSURLConnection to launch an asynchronous request

Update the class interface

- Update the enclosing class interface to …
  - conform to NSURLConnectionDelegate
- For example:

```
@interface SCISProgramsViewController : UITableViewController
    <UITableViewDelegate, UITableViewDataSource,
     NSURLConnectionDelegate>
```

Interface updated to conform to NSURLConnectionDelegate

Update the class impl

- First, we add a variable declaration to hold the response data:

  NSMutableData *_responseData;

- For instance …

```
@implementation SCISProgramsViewController

NSMutableData *_responseData;
```

NSURLConnection delegate methods

- When the response comes back from the server, the following delegate methods are invoked …
  - didReceiveResponse
    - may happen multiple times per request
      - e.g., this can happen if the response is in multipart MIME encoding
  - didReceiveData
    - responsible for storing the newly received data
  - didFailWithError
    - called if an error occurs; if so, no other methods will be called
  - connectionDidFinishLoading
    - indicates successful receipt of the request

Implement the delegate methods

```
NSMutableData *_responseData;

- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse *)response {
    _responseData = [[NSMutableData alloc] init];
}

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data {
    [_responseData appendData:data];
}

- (NSCachedURLResponse *)connection:(NSURLConnection *)connection
                  willCacheResponse:(NSCachedURLResponse*)cachedResponse {
    return nil;  // return nil to indicate a cached response is not necessary
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
    // parse the data (JSON, XML) in _responseData
    // TODO
}

- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error {
    NSLog(@"error: %@", error);
}
```

Invoke the asynchronous request

- Invoking an asynchronous request

```
NSURL *url =[NSURL URLWithString:
    @"http://localhost:8080/SCIS/webresources/domain.programs"];

NSMutableURLRequest *request =
    [NSMutableURLRequest requestWithURL:url];

[request setHTTPMethod:@"GET"];

[[NSURLConnection alloc] initWithRequest:request delegate:self];
```

Finally, notice no GCD

- When you use NSURLConnection to invoke an asynchronous web request, …
  - you don’t need to directly interface with GCD
  - rather, GCD is used “under the hood”

- In Summary, NSURLConnection …
  - provides more programmatic control of URLs
  - removes the need to directly invoke GCD