Connecting to the Net

iPhone and iPod touch Development
Fall 2009 — Lecture 21

Questions?

Announcements

- Last class we discussed audio in some detail, here's another reference you can use for that part of the material...
  - Manning just released a free sample chapter on audio from their upcoming iPhone in Action Second Edition book
    - You can found out more about the book, including the free chapter at...
      - http://manning.com/trebitowski/
- Assignment #8 (the last of the week-long assignments) out in the next day or so

Today's Topics

- Foundation Framework
- Embedding a Web View
- Composing Email
- Consuming XML Web Services
- Web Server
- Other Networking APIs

Notes

- I'm showing the relevant portions of the view controller interfaces and implementations in these notes
- Remember to release relevant memory in the -dealloc methods — they are not shown here
- You will also need to wire up outlets and actions in IB
- Where delegates are used, they too require wiring in IB

Foundation Framework

Foundation Framework

- Many classes in the Foundation framework support initialization of some form via a URL
- NSString...

```
+ (id)stringWithContentsOfURL:(NSURL *)url
                     encoding:(NSStringEncoding)enc
                        error:(NSError **)error;
```

- NSArray...

```
+ (id)arrayWithContentsOfURL:(NSURL *)url;
```

- NSDictionary...

```
+ (id)dictionaryWithContentsOfURL:(NSURL *)url;
```

The App

- Our app is going provide a text field for the user to enter a zipcode that they wish to lookup
- The user can then press Search to kickstart the lookup process
- We'll craft the URL and ask the PHP script for the corresponding city and state
- The PHP script returns a plist (a serialized dictionary)
- After getting a response, the data is extracted and updated in the UI

Server-Side Script

```
<?php
  $zip = $_REQUEST['zip'];
  $zips = file('zip.csv');
  $city = '';
  $state = '';
  foreach($zips as $data) {
    $segments = split(":", $data);
    if($segments[0] == $zip) {
      $city = $segments[1];
      $state = $segments[2];
      break;
    }
  }
  print '<?xml version="1.0" encoding="UTF-8"?>' . "\n";
?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>city</key>
    <string><?= $city ?></string>
    <key>state</key>
    <string><?= $state ?></string>
  </dict>
</plist>
```

zip.csv

```
21520:Accident:MD
21521:Barton:MD
21522:Bittinger:MD
21523:Bloomington:MD
21524:Corriganville:MD
21528:Eckhart Mines:MD
21529:Ellerslie:MD
21530:Flintstone:MD
21531:Friendsville:MD
21532:Frostburg:MD
21536:Grantsville:MD
21538:Kitzmiller:MD
21539:Lonaconing:MD
21540:Luke:MD
21541:Mc Henry:MD
21542:Midland:MD
21543:Midlothian:MD
21545:Mount Savage:MD
21550:Oakland:MD
...
```

ZipcodeViewController.xib

- In this NIB, we have a text field for the user to enter a zipcode
- A search button to initiate the lookup of city and state
- Labels in which to display the city and state associated with the given zipcode

ZipcodeViewController.h

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

@interface ZipcodeViewController : UIViewController {

  UITextField *zipcode;
  UILabel *city;
  UILabel *state;

}

@property(nonatomic, retain) IBOutlet UITextField *zipcode;
@property(nonatomic, retain) IBOutlet UILabel *city;
@property(nonatomic, retain) IBOutlet UILabel *state;

- (IBAction)lookupZip;

@end
```

ZipcodeViewController.m

```
#import "ZipcodeViewController.h"

#define ZIPSCRIPT @"http://userpages.umbc.edu/~dhood2/courses/cmsc491i/fall2009/
lectures/lecture21/zip.php"

@implementation ZipcodeViewController

@synthesize zipcode;
@synthesize city;
@synthesize state;

- (IBAction)lookupZip {
  [self.zipcode resignFirstResponder];
  NSString *urlStr = [NSString stringWithFormat:@"%@?zip=%@", ZIPSCRIPT,
                                                 self.zipcode.text];
  NSURL *url = [NSURL URLWithString:urlStr];
  NSDictionary *results = [NSDictionary dictionaryWithContentsOfURL:url];
  self.city.text = results ? [results objectForKey:@"city"] : @"Unknown";
  self.state.text = results ? [results objectForKey:@"state"] : @"Unknown";
}

// ...

@end
```

The Resulting App

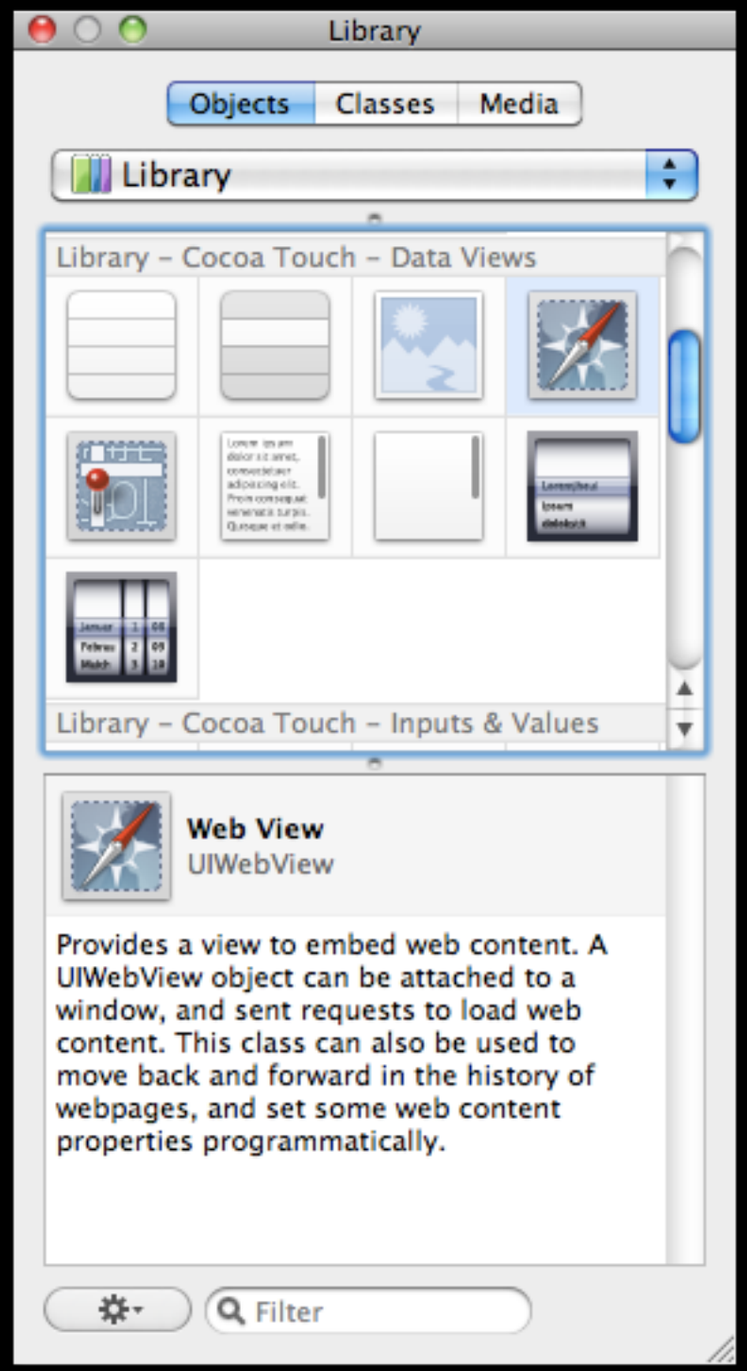
Embedding a Web View

Embedding a UIWebView

- You can display web content by embedding a UIWebView into your application
- Content can be...
  - A remote or local URL
  - An string that contains HTML
  - Raw data & corresponding MIME type

UIWebView

- The UIWebView utilizes WebKit to perform the rendering
- Simple API for loading & navigating
- UIWebViewDelegate supports callback control
- Limited JavaScript execution support
  - 5 seconds of execution
  - 10 MB of memory



UIWebView

- Load a URL from an NSURLRequest...

```
- (void)loadRequest:(NSURLRequest *)request;
```

- Load HTML from a string...

```
- (void)loadHTMLString:(NSString *)string
               baseURL:(NSURL *)baseURL;
```

- Load from NSData...

```
- (void)loadData:(NSData *)data
        MIMEType:(NSString *)MIMEType
textEncodingName:(NSString *)textEncodingName
         baseURL:(NSURL *)baseURL;
```

UIWebView

- Can we can navigate history backwards/forwards...

```
@property(nonatomic,readonly,getter=canGoBack) BOOL canGoBack;
@property(nonatomic,readonly,getter=canGoForward) BOOL canGoForward;
```

- Go backwards/forwards...

```
- (void)goBack;
- (void)goForward;
```

- Is the view currently loading something...

```
@property(nonatomic,readonly,getter=isLoading) BOOL loading;
```

- Reloading methods...

```
- (void)reload;
- (void)stopLoading;
```

UIWebViewDelegate

- Is the view about to starting loading...

```
- (BOOL)webView:(UIWebView *)webView
        shouldStartLoadWithRequest:(NSURLRequest *)request
        navigationType:(UIWebViewNavigationType)navigationType;
```

- Has the view started loaded...

```
- (void)webViewDidStartLoad:(UIWebView *)webView;
```

- Is the view done loading...

```
- (void)webViewDidFinishLoad:(UIWebView *)webView;
```

- Was there an error...

```
- (void)webView:(UIWebView *)webView
        didFailLoadWithError:(NSError *)error;
```

WebBrowserViewController.xib

- In the NIB, we've added...
  - A toolbar with a text field and a "go" button for the user to enter in URLs
  - A big UIWebView to draw the requested resource
  - We'll set the view controller to be the delegate for both the text field and web views

WebBrowserViewController.h

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

@interface WebBrowserViewController : UIViewController <UITextFieldDelegate,
                                                        UIWebViewDelegate> {

    UITextField *urlField;
    UIWebView *webView;

}

@property(nonatomic, retain) IBOutlet UITextField *urlField;
@property(nonatomic, retain) IBOutlet UIWebView *webView;

- (IBAction)loadURL;

@end
```

WebBrowserViewController.m

```
#import "WebBrowserViewController.h"

@implementation WebBrowserViewController

@synthesize urlField;
@synthesize webView;

- (IBAction)loadURL {
  NSURL *url = [NSURL URLWithString:self.urlField.text];
  NSURLRequest *req = [NSURLRequest requestWithURL:url];
  [self.urlField resignFirstResponder];
  [self.webView loadRequest:req];
}

- (BOOL)textFieldShouldReturn:(UITextField *)textField {
  [self loadURL];
  return YES;
}

- (void)webViewDidFinishLoad:(UIWebView *)theWebView {
  self.urlField.text = [NSString stringWithFormat:@"%@", theWebView.request.URL];
}

// ...

@end
```

The Resulting App

Adding an “about:” Screen

- We can also display local content by constructing a file URL to a local resource...

```
- (IBAction)loadURL {
  NSURL *url;
  if ([self.urlField.text isEqualToString:@"about:"]) {
    NSString *path = [[NSBundle mainBundle] pathForResource:@"about"
                                                     ofType:@"html"];
    url = [[[NSURL alloc] initFileURLWithPath:path] autorelease];
  } else {
    url = [NSURL URLWithString:self.urlField.text];
  }
  NSURLRequest *req = [NSURLRequest requestWithURL:url];
  [self.urlField resignFirstResponder];
  [self.webView loadRequest:req];
}
```

about.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
    <title>About</title>
    <style type="text/css">
      body {
        background: maroon;
        color: white;
        font-style: italic;
        font-size: 300%;
      }
    </style>
  </head>
  <body>
    <p>
      Hey, this isn't Mozilla!
      What did you expect?
    </p>
  </body>
</html>
```

The Resulting App

Other WebView Uses

- There are other uses for embedding a WebKit view into your app, including...
  - Stylizing fonts beyond UIKit's built-in capabilities
  - Providing hyperlinks to external sites
  - Providing a "mailto" hyperlink to open the mail app
    - Optionally populate fields via URL encoding

about.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
    <title>About</title>
    <style type="text/css">
      h1 {
        letter-spacing: 3px;
        text-decoration: underline;
        font-family: "Marker Felt";
      }
      .shadows {
        text-shadow: 0 0 24px #c00, 0 0 4px #c00, 1px 1px 2px #333;
        color: #fff;
      }
      p {
        font-size: 250%;
        margin-top: 10px;
        margin-bottom: 5px;
      }
      del {
        color: red;
      }

      /* ... */
```

about.html

```
    /* ... */

    ins {
      color: green;
    }
  </style>
</head>
<body>
  <h1>WebBrowser App</h1>
    <p class="shadows">Fun with Shadows</p>
    <p>
      foo <del>bah</del> <ins>bar</ins> baz
    </p>
    <p>
      <a href="http://cs491f09.wordpress.com/">Course Blog</a>
    </p>
    <p>
      <a href="mailto:danielhood@umbc.edu?subject=App%20Support">Email Support</a>
    </p>
  </body>
</html>
```

WebBrowserViewController.m

- By default, web links open in the current web view
- If you'd like them to open in Mobile Safari, you need to implement the following method on the view controller...

```
- (BOOL)webView:(UIWebView *)webView
      shouldStartLoadWithRequest:(NSURLRequest *)request
      navigationType:(UIWebViewNavigationType)navigationType {

  if (navigationType == UIWebViewNavigationTypeLinkClicked) {
    [[UIApplication sharedApplication] openURL:request.URL];
    return NO;
  }

  return YES;

}
```

Simulator Support

- Since there's not Mail app on the simulator, clicking on the mailto: URLs does nothing

The Resulting App

Composing Email

Composing Email

- We just saw a way (using a mailto: link) to open the Mail app, however this is not without it's limitations...
  - No way to detect if the user cancelled or sent message
  - Limited message options (no HTML or attachments)
  - Once the user's done, they're left in the Mail app
- The MFMailComposeViewController class was added in iPhone 3.0 which provides a standard mail composition interface inside your app
  - Detection of cancellation or sent
  - Richer message options
  - User stays in your app

MFMailComposeViewController

- First you should always check to see if the device supports sending email...

```
+ (BOOL)canSendMail;
```

- You can set the various recipients for each of the mail heading fields (arrays of strings)...

```
- (void)setToRecipients:(NSArray *)toRecipients;
- (void)setCcRecipients:(NSArray *)ccRecipients;
- (void)setBccRecipients:(NSArray *)bccRecipients;
```

- Set the subject of an email...

```
- (void)setSubject:(NSString *)subject;
```

MFMailComposeViewController

- Set the contents of the message...
  - Body is just the text contents if not an HTML message
  - Otherwise, it's an HTML marked up message

```
- (void)setMessageBody:(NSString *)body isHTML:(BOOL)isHTML;
```

- Add attachments, first need to be slurped into an NSData object...

```
- (void)addAttachmentData:(NSData *)attachment
                 mimeType:(NSString *)mimeType
                 fileName:(NSString *)filename;
```

MFMailComposeViewControllerDelegate

- The MFMailComposeViewController has a delegate with only the following method...

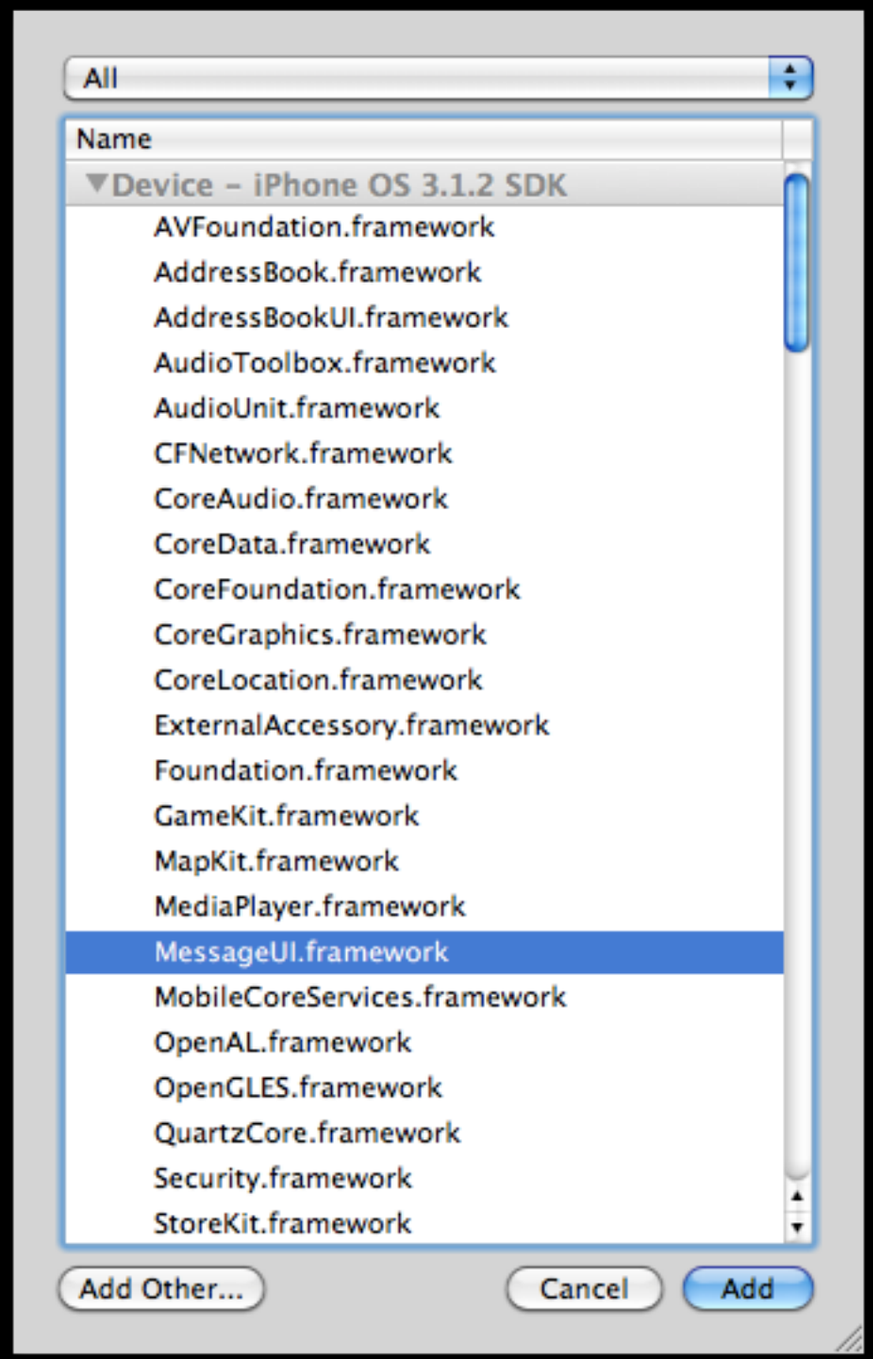
```
- (void)mailComposeController:(MFMailComposeViewController *)controller
          didFinishWithResult:(MFMailComposeResult)result
                        error:(NSError *)error;
```

- You can use the method to tell if there was an error composing the message, or what the status was...

```
enum MFMailComposeResult {
    MFMailComposeResultCancelled,
    MFMailComposeResultSaved,
    MFMailComposeResultSent,
    MFMailComposeResultFailed
};
typedef enum MFMailComposeResult MFMailComposeResult;
```

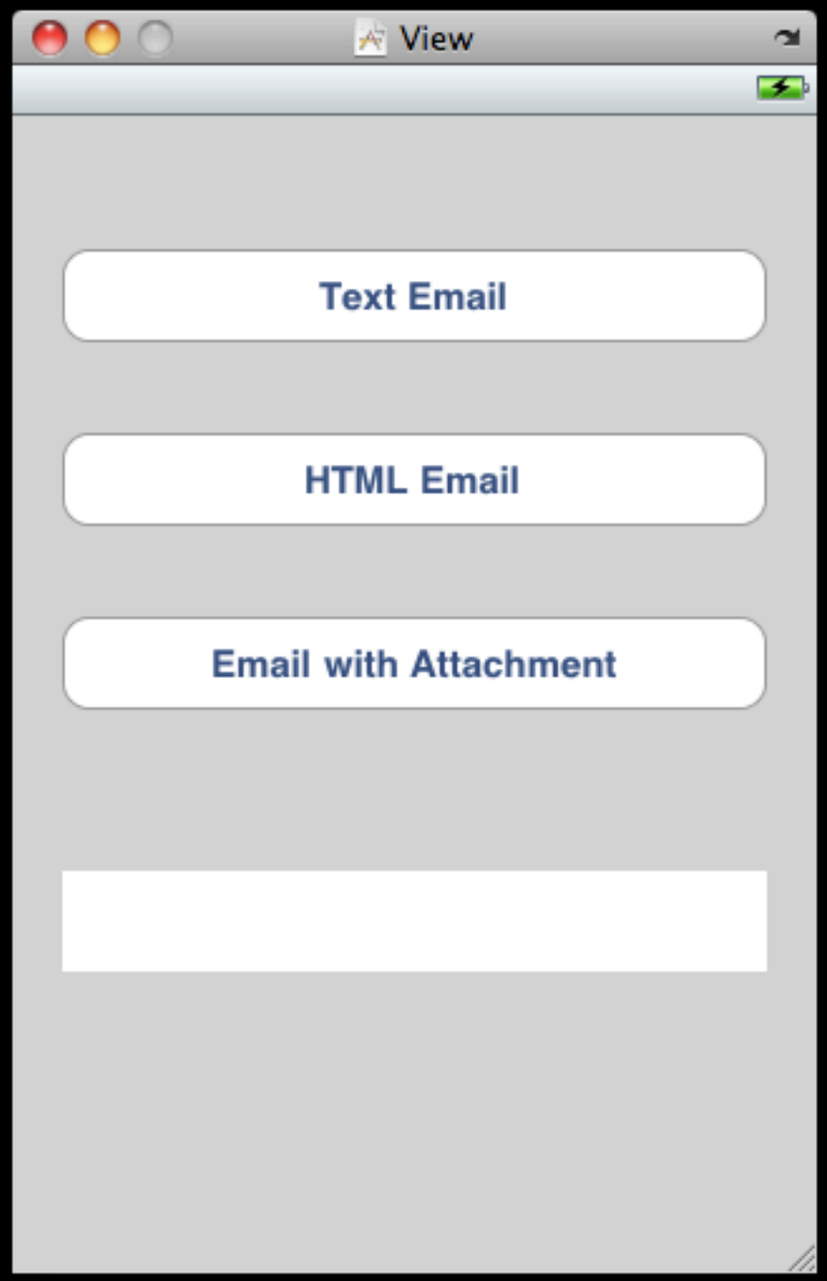
MessageUI Framework

- MFMailComposeViewController is contained in the MessageUI framework, which you'll need to add to your project



EmailerViewController.xib

- Our NIB for this app will consist of 3 buttons, each sending a different type of email
- We'll also have a status area where we can display any errors or status messages



EmailerViewController.h

```
#import <MessageUI/MessageUI.h>
#import <UIKit/UIKit.h>

@interface EmailerViewController : UIViewController
                                   <MFMailComposeViewControllerDelegate> {

  UILabel *status;

}

@property(nonatomic, retain) IBOutlet UILabel *status;

- (IBAction)sendPlainEmail;
- (IBAction)sendHTMLEmail;
- (IBAction)setEmailWithAttachment;

@end
```

EmailerViewController.m

```
#import "EmailerViewController.h"

@implementation EmailerViewController

@synthesize status;

- (void)mailComposeController:(MFMailComposeViewController *)controller
          didFinishWithResult:(MFMailComposeResult)result
                        error:(NSError *)error {
  if (error) {
    self.status.text = [error localizedDescription];
  } else {
    switch (result) {
      case MFMailComposeResultCancelled:
        self.status.text = @"Cancelled"; break;
      case MFMailComposeResultSaved:
        self.status.text = @"Saved Message"; break;
      case MFMailComposeResultSent:
        self.status.text = @"Sent Message"; break;
      case MFMailComposeResultFailed:
        self.status.text = @"Failure"; break;
       }
   }
   [controller dismissModalViewControllerAnimated:YES];
}
// ...
```

EmailerViewController.m

```
// ...

- (void)cantSendMail {
    self.status.text = @"Error: device can't send mail!";
}

- (IBAction)sendPlainEmail {
    if ([MFMailComposeViewController canSendMail]) {
    MFMailComposeViewController *mailVC = [[[MFMailComposeViewController alloc] init]
                                           autorelease];
        [mailVC setToRecipients:[NSArray arrayWithObjects:@"danielhood@umbc.edu", nil]];
        [mailVC setSubject:@"Email from your app"];
        [mailVC setMessageBody:@"Hello World!" isHTML:NO];
        mailVC.mailComposeDelegate = self;
        [self presentModalViewController:mailVC animated:YES];
    } else {
        [self cantSendMail];
    }
}

// ...
```

EmailerViewController.m

```
// ...

- (IBAction)sendHTMLEmail {
  if ([MFMailComposeViewController canSendMail]) {
    NSMutableString *html = [NSMutableString string];
    [html appendString:@"<html><body>"];
    [html appendString:@"<p style=\"font-size: 200%; color: purple\">PONIES!!!</p>"];
    [html appendString:@"<img src=\"http://userpages.umbc.edu/~dhood2/courses/
cmsc491i/fall2009/lectures/lecture21/pony.jpg\" />"];
    [html appendString:@"</body></html>"];
    MFMailComposeViewController *mailVC = [[[MFMailComposeViewController alloc]
                                            init] autorelease];
    [mailVC setSubject:@"ZOMG Ponies!!!"];
    [mailVC setMessageBody:html isHTML:YES];
    mailVC.mailComposeDelegate = self;
    [self presentModalViewController:mailVC animated:YES];
  } else {
    [self cantSendMail];
  }
}

// ...
```

EmailerViewController.m

```
// ...

- (IBAction)setEmailWithAttachment {
  if ([MFMailComposeViewController canSendMail]) {
  MFMailComposeViewController *mailVC = [[[MFMailComposeViewController alloc] init]
                                         autorelease];
    [mailVC setSubject:@"Your Data"];
    [mailVC setMessageBody:@"Data is attached." isHTML:NO];
    NSString *path = [[NSBundle mainBundle] pathForResource:@"foobar" ofType:@"txt"];
    NSData *data = [NSData dataWithContentsOfFile:path];
    [mailVC addAttachmentData:data mimeType:@"text/plain" fileName:@"foobar.txt"];
    mailVC.mailComposeDelegate = self;
    [self presentModalViewController:mailVC animated:YES];
  } else {
    [self cantSendMail];
  }
}

- (void)viewDidLoad {
  [super viewDidLoad];
  self.view.backgroundColor = [UIColor groupTableViewBackgroundColor];
}

// ...

@end
```

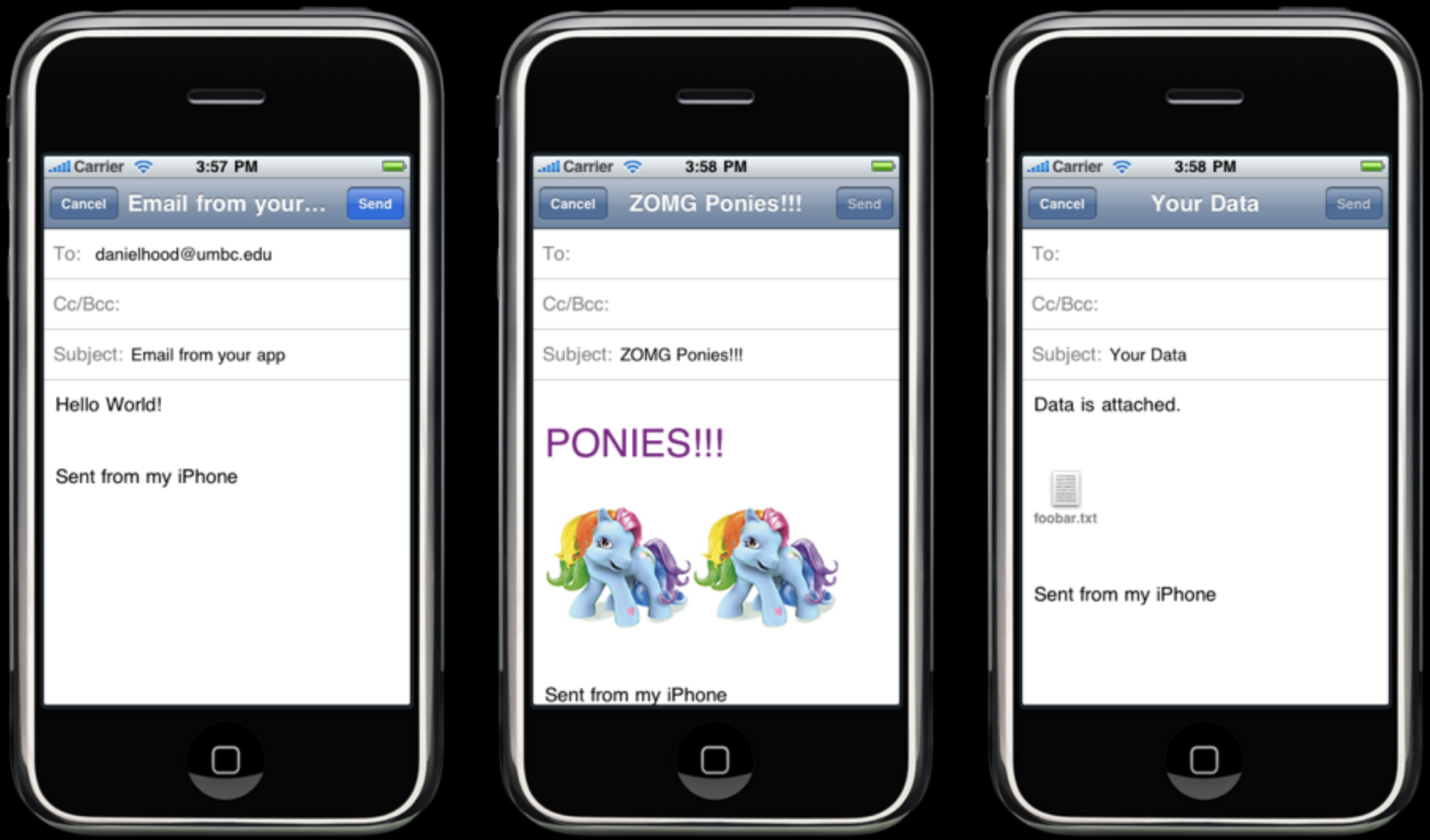
Blue Stripes

Simulator Support

- Since MessageUI isn't part of the Mail app, it will come up in your app
- Can't seem to find Apple documentation as to whether or not it can send emails
  - Conflicting information online
- Didn't seem to send anything when I tried it

The Resulting App

The Resulting App



Consuming XML Web Services

Consuming XML Web Services

- Web services come in a number of flavors...
  - SOAP XML
  - REST XML
  - JSON
  - etc...

A Dictionary Web Service

- For this example we'll use a simple XML-based dictionary web service from Aonaware...
  - http://services.aonaware.com/DictService/DictService.asmx
- This service has multiple endpoint variations...
  - SOAP 1.1
  - SOAP 1.2
  - REST (GET)
  - etc...
- We'll use the simple REST based GET interface for this app

Web Service

- To consume this web service, we need to send the following HTTP request...

```
GET /DictService/DictService.asmx/DefineInDict?dictId=string&word=string HTTP/1.1
Host: services.aonaware.com
```

- And we'll get a response back like...

Web Service

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<WordDefinition xmlns="http://services.aonaware.com/webservices/">
  <Word>string</Word>
  <Definitions>
    <Definition>
      <Word>string</Word>
      <Dictionary>
        <Id>string</Id>
        <Name>string</Name>
      </Dictionary>
      <WordDefinition>string</WordDefinition>
    </Definition>
    <Definition>
      <Word>string</Word>
      <Dictionary>
        <Id>string</Id>
        <Name>string</Name>
      </Dictionary>
      <WordDefinition>string</WordDefinition>
    </Definition>
  </Definitions>
</WordDefinition>
```

The App

- Our app is going provide a text field for the user to enter a word that they wish to lookup
- The user can then press Done or click Define to kickstart the lookup process
- We'll craft the URL and ask the web service for a definition for the user's word
- The app will buffer the data coming back until it has all of the data from the server
- It will then hand if off to the XML parser to extract the definitions
- When done, it will display all of the definitions in a text view

Parsing XML

- There are several XML parsing libraries for ObjC
- Probably the easiest one to use is NSXMLParser
  - NSXMLParser is an event driven parser
  - It notifies its delegate about the items (elements, attributes, CDATA blocks, comments, etc...) that it encounters as it processes an XML document
- It does not itself do anything with those parsed items except report them
- We need to provide the logic in these callbacks to target and handle the data we're interested in

DefineViewController.xib

- The view is pretty simple...
  - A text field at the top
  - Button to perform the lookup
  - A large text area to display the resulting definitions

DefineViewController.h

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

@interface DefineViewController : UIViewController <UITextFieldDelegate> {

    UITextField *word;
    UITextView *definition;
    NSMutableData *definitionData;
    NSMutableString *buffer;
    BOOL inDefinitionTag;
    BOOL inWordDefinitionTag;

}

@property(nonatomic, retain) IBOutlet UITextField *word;
@property(nonatomic, retain) IBOutlet UITextView *definition;
@property(nonatomic, retain) NSMutableData *definitionData;
@property(nonatomic, retain) NSMutableString *buffer;
@property(nonatomic, assign) BOOL inDefinitionTag;
@property(nonatomic, assign) BOOL inWordDefinitionTag;

- (IBAction)define;

@end
```

DefineViewController.m

```
#import "DefineViewController.h"

#define HOST @"http://services.aonaware.com"
#define PATH @"DictService/DictService.asmx/DefineInDict"
#define DICT @"gcide"

@implementation DefineViewController

@synthesize word, definition, definitionData;
@synthesize buffer, inDefinitionTag, inWordDefinitionTag;

- (IBAction)define {
  [self.word resignFirstResponder];
  self.definitionData = [NSMutableData data];
  self.definition.text = @"";
  NSString *urlStr = [NSString stringWithFormat:@"%@/%@?dictId=%@&word=%@", HOST,
                                                PATH, DICT, self.word.text];
  NSURL *url = [NSURL URLWithString:urlStr];
  NSURLRequest *req = [NSURLRequest requestWithURL:url];
  [NSURLConnection connectionWithRequest:req delegate:self];
}

// ...
```

DefineViewController.m

```
// ...

- (NSString *)cleanupString:(NSString *)str {
  NSMutableString *result = [NSMutableString string];
  NSScanner *scanner = [NSScanner scannerWithString:str];

  while (![scanner isAtEnd]) {
    if ([scanner scanCharactersFromSet:[NSCharacterSet
                  whitespaceAndNewlineCharacterSet] intoString:NULL]) {
      [result appendString:@" "];
    }
    NSString* stringPart = nil;
    if ([scanner scanUpToCharactersFromSet:[NSCharacterSet
                  whitespaceAndNewlineCharacterSet] intoString:&stringPart]) {
      [result appendString:stringPart];
      [result appendString:@" "];
    }
  }
  return result;
}

// ...
```

DefineViewController.m

```
// ...

- (BOOL)textFieldShouldReturn:(UITextField *)textField {
  [self define];
  return YES;
}

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

- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
  NSXMLParser *parser = [[[NSXMLParser alloc] initWithData:self.definitionData]
                         autorelease];
  parser.delegate = self;
  [parser parse];
}

// BEGIN PARSING METHODS
- (void)parserDidStartDocument:(NSXMLParser *)parser {
   self.buffer = [NSMutableString string];
}

// ...
```

DefineViewController.m

```
// ...

- (void)parser:(NSXMLParser *)parser
       didStartElement:(NSString *)elementName
       namespaceURI:(NSString *)namespaceURI
       qualifiedName:(NSString *)qName
       attributes:(NSDictionary *)attributeDict {
  if ([elementName isEqualToString:@"Definition"]) {
    self.inDefinitionTag = YES;
  }
  if (self.inDefinitionTag && [elementName isEqualToString:@"WordDefinition"]) {
    self.inWordDefinitionTag = YES;
  }
}

- (void)parser:(NSXMLParser *)parser
       foundCharacters:(NSString *)string {
  if (self.inDefinitionTag && self.inWordDefinitionTag) {
    NSString *stripped = [string stringByReplacingOccurrencesOfString:@"\n"
                                 withString:@""];
    [self.buffer appendString:[self cleanupString:stripped]];
  }
}

// ...
```

DefineViewController.m

```
// ...

- (void)parser:(NSXMLParser *)parser
       didEndElement:(NSString *)elementName
       namespaceURI:(NSString *)namespaceURI
       qualifiedName:(NSString *)qName {
  if (self.inDefinitionTag && [elementName isEqualToString:@"WordDefinition"]) {
    [self.buffer appendString:@"\n\n"];
    self.inWordDefinitionTag = NO;
  }
  if ([elementName isEqualToString:@"Definition"]) {
    self.inDefinitionTag = NO;
  }
}

- (void)parserDidEndDocument:(NSXMLParser *)parser {
  if ([self.buffer length] == 0) {
    self.definition.text = @"No Matches";
  } else {
    self.definition.text = self.buffer;
  }
}

// ...

@end
```

Web Server

Web Server

- Not only can the iPhone consume network resources, it can also provide them
- There are actually a number of web server implementations written for the iPhone
- Why you might want to have the iPhone be web server...
  - Easy way to get data off the phone — just pull up a web browser and type in the iPhone's URL
  - Convenient way to share files — upload or download
  - Why not?!? ...it's a web server on your phone!

Objective-C Web Servers

- There are a number of web servers out there that will run on the iPhone — some of them are even written in ObjC (vs. plain old C)
  - Apple's CocoaHTTPServer
    - http://developer.apple.com/mac/library/samplecode/CocoaHTTPServer/
  - cocoahttpserver on Google Code
    - http://code.google.com/p/cocoahttpserver/

cocoahttpserver

- For this example, we'll look at the cocoahttpserver on Google Code
- This library is very well documented and even contains a fairly robust iPhone Xcode project that's more complex than what we're going to do here
- http://cocoahttpserver.googlecode.com/files/iPhoneHTTPServer3.zip

Library Files

- For this example, I've copied over the following files from their example into a new view-based project...
  - localhostAddresses.[mh]
  - DDNumber.[mh]
  - DDRange.[mh]
  - DDData.[mh]
  - AsyncSocket.[mh]
  - HTTPServer.[mh]
  - HTTPResponse.[mh]
  - HTTPConnection.[mh]
  - HTTPAuthenticationRequest.[mh]

WebServerViewController.xib

- Our NIB will contain a simple switch to toggle the server on and off
- It will also contain a text label for the app to publish its IP address to so users know what URL to open on their browsers

cocoahttpserver Usage

- Basically we're going to have code in 2 places...
  - View Controller
    - Server instantiation & configuration
    - Server startup & shutdown
  - Custom Request Handler
    - Our implementation on how to handle a request

Our App

- For this demo we're going to package up some files with the bundle and copy them into the app's Documents directory
- In a "real-world" app, you'd probably be sharing either data your app generated or uploaded into the Documents
- In our app, we'll support 2 types of requests...
  - A request for a directory listing — i.e. "GET /"
  - Or, a request for a file — i.e. "GET /test.html"
- Our app doesn't do much error checking and could be much more robust, but none-the-less it illustrates the web server usage

WebServerViewController.h

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

@class HTTPServer;
@interface WebServerViewController : UIViewController {

  HTTPServer *httpServer;
  NSDictionary *addresses;
  UILabel *status;

}

@property(nonatomic, retain) HTTPServer *httpServer;
@property(nonatomic, retain) NSDictionary *addresses;
@property(nonatomic, retain) IBOutlet UILabel *status;

- (IBAction)toggleServer:(id)sender;

@end
```

WebServerViewController.m

```
#import "HTTPServer.h"
#import "localhostAddresses.h"
#import "MyHTTPConnection.h"
#import "WebServerViewController.h"

@implementation WebServerViewController

@synthesize status, httpServer, addresses;

- (IBAction)toggleServer:(id)sender {
  UISwitch *server = sender;
  if (server.isOn) {
    NSError *error;
    if(![self.httpServer start:&error]) {
      self.status.text = [NSString stringWithFormat:@"Error: %@", error];
    } else {
      [localhostAddresses performSelectorInBackground:@selector(list) withObject:nil];
    }
  } else {
    [self.httpServer stop];
       self.status.text = @"Not Running";
   }
}

// ...
```

WebServerViewController.m

```
// ...

// Here we are copying files from the bundle into the Documents directory.
// In the real-world, your app might generate files or allow upload of files
// in lieu of copying from the bundle.
- (void)loadDummyFiles {
  NSFileManager *mgr = [NSFileManager defaultManager];
  NSArray *files = [NSArray arrayWithObjects:@"foobar.txt", @"pony.jpg",
                                             @"test.html", nil];
  for (NSString *f in files) {
    NSArray *fileParts = [f componentsSeparatedByString:@"."];
    NSString *base = [fileParts objectAtIndex:0];
    NSString *ext = [fileParts objectAtIndex:1];
    NSString *from = [[NSBundle mainBundle] pathForResource:base ofType:ext];
    NSArray *segments = [NSArray arrayWithObjects:NSHomeDirectory(), @"Documents",
                                                  f, nil];
    NSString *to = [NSString pathWithComponents:segments];
     [mgr copyItemAtPath:from toPath:to error:NULL];
  }
}

// ...
```

WebServerViewController.m

```
// ...

- (void)update:(NSNotification *) notification {
  if(notification) {
    self.addresses = [notification object];
  }
  if(!addresses) {
    return;
  }
  UInt16 port = [httpServer port];
  NSString *localIP = [addresses objectForKey:@"en0"];
  if (!localIP) {
    localIP = [addresses objectForKey:@"en1"];
  }
  if (!localIP) {
    self.status.text = @"No Connection!";
  } else {
    NSString *running = [NSString stringWithFormat:@"Running @ http://%@:%d\n",
                                                   localIP, port];
    NSLog(@"%@", running);
    self.status.text = running;
  }
}

// ...
```

WebServerViewController.m

```
// ...

- (void)viewDidLoad {

  [super viewDidLoad];
  [self loadDummyFiles];
  NSArray *segments = [NSArray arrayWithObjects:NSHomeDirectory(), @"Documents", nil];
  NSString *root = [NSString pathWithComponents:segments];

  // setup web server
  self.httpServer = [HTTPServer new];
  [self.httpServer setPort:8080];
  [self.httpServer setType:@"_http._tcp."];
  [self.httpServer setConnectionClass:[MyHTTPConnection class]];
  [self.httpServer setDocumentRoot:[NSURL fileURLWithPath:root]];

  // setup address resolution notifications
  [[NSNotificationCenter defaultCenter] addObserver:self
                                           selector:@selector(update:)
                                               name:@"LocalhostAdressesResolved"
                                             object:nil];
}

// ...

@end
```

MyHTTPConnection.h

```
#import <Foundation/Foundation.h>
#import "HTTPConnection.h"

@interface MyHTTPConnection : HTTPConnection {

}


@end
```

MyHTTPConnection.m

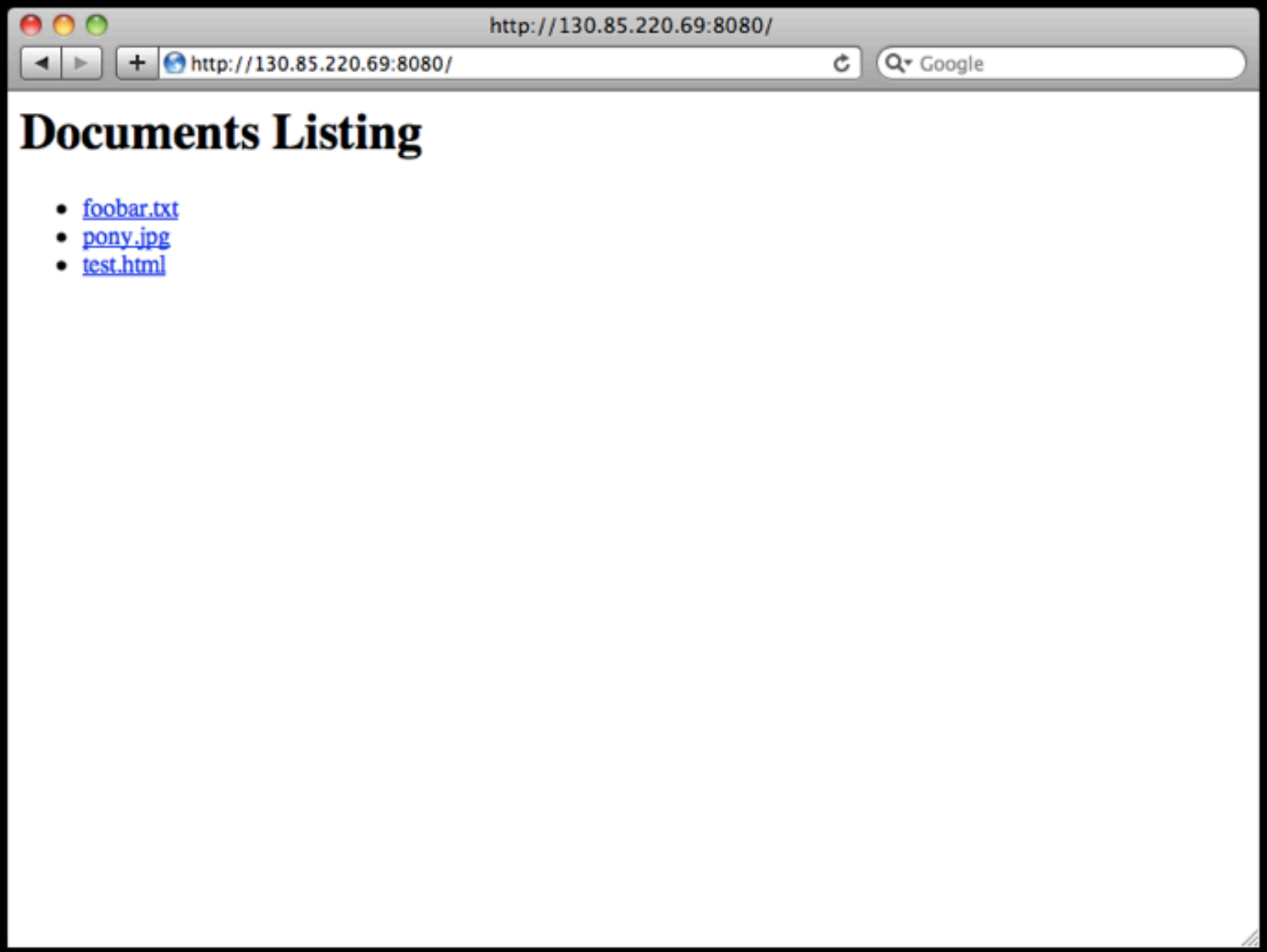
```
#import "HTTPResponse.h"
#import "HTTPServer.h"
#import "MyHTTPConnection.h"

@implementation MyHTTPConnection

- (NSObject<HTTPResponse> *)httpResponseForMethod:(NSString *)method
                                              URI:(NSString *)path {
  NSLog(@"%@ %@", method, path);

  // if directory listing
  if ([path isEqualToString:@"/"]) {
    NSArray *files = [[NSFileManager defaultManager] directoryContentsAtPath:
                      [[server documentRoot] path]];
    NSMutableString *outdata = [NSMutableString string];
    [outdata appendString:@"<html><body><h1>Documents Listing</h1><ul>"];
    for (NSString *f in files) {
      [outdata appendFormat:@"<li><a href=\"%@\">%@</a></li>", f, f];
    }
    [outdata appendString:@"</ul></body></html>"];
    NSData *browseData = [outdata dataUsingEncoding:NSUTF8StringEncoding];
    return [[[HTTPDataResponse alloc] initWithData:browseData] autorelease];

   // ...
```

MyHTTPConnection.m

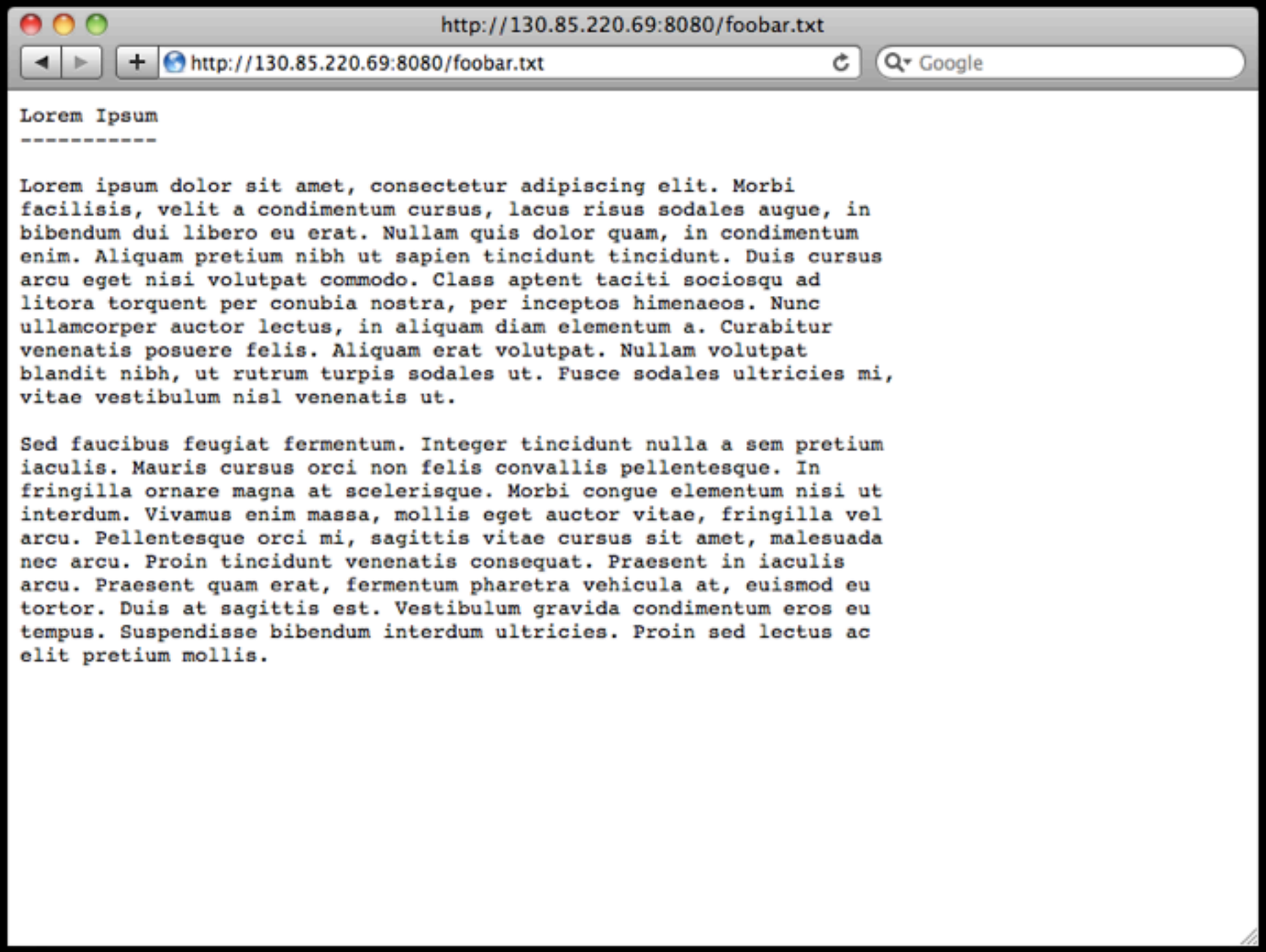
```
// ...

// else get a given file
} else {
  NSArray *segments = [NSArray arrayWithObjects:NSHomeDirectory(), @"Documents",
                                                [path substringFromIndex:1], nil];
  NSString *filePath = [NSString pathWithComponents:segments];
  if ([[NSFileManager defaultManager] fileExistsAtPath:filePath]) {
    return [[HTTPFileResponse alloc] initWithFilePath:filePath] autorelease];
  } else {
    NSData *browseData = [@"404 - Not Found" dataUsingEncoding:NSUTF8StringEncoding];
    return [[HTTPDataResponse alloc] initWithData:browseData] autorelease];
  }
 }
}

@end
```

The Resulting App

The Resulting App



The Resulting App



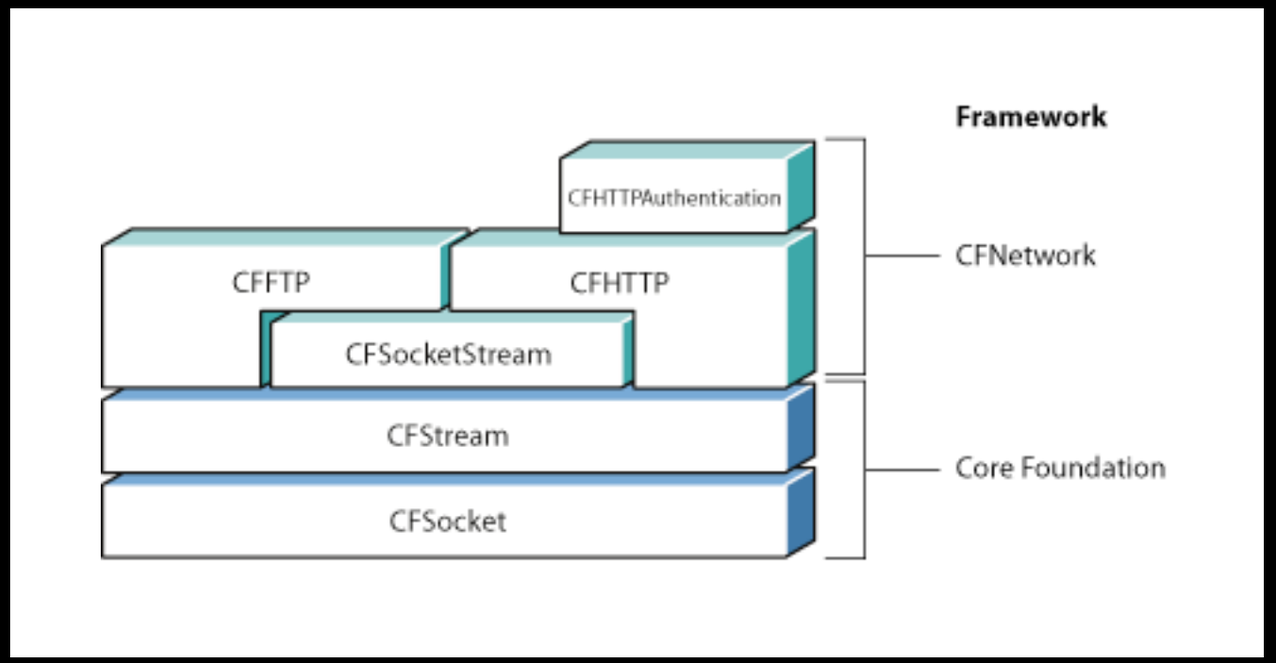
Other Networking APIs

Other Networking APIs

- If you're doing something other than HTTP, you'll probably end up having to do something else
- Your primary options are probably...
  - The CFNetwork framework
  - BSD sockets
  - Some 3rd party library

CFNetwork Framework

- CFNetwork is a low-level, high-performance framework that gives you the ability to have detailed control over the protocol stack
- It is an extension to BSD sockets that provides objects to simplify tasks such as communicating with FTP and HTTP servers or resolving DNS hosts



BSD Sockets

- C-level API for network programming...
  - socket() — creates a new socket
  - bind() — reserve an IP and port number
  - listen() — wait for connections
  - connect() — attempt to establish an IP connection
  - accept() — create a new connection from client
  - send(), recv(), write(), read() — Input/Output
  - close() — terminate and release resources

BSD Sockets

- Continued...
  - gethostbyname(), gethostbyaddr() — lookup functions.
  - select() — picks from existing sockets
  - poll() — check state of socket
  - getsockopt(), setsockopt() — retrieve and set options
- For full documentation check out the iPhone OS man pages at...
  - http://developer.apple.com/iPhone/library/documentation/System/Conceptual/ManPages_iPhoneOS/

Additional Resources

- CFNetwork Programming Guide
  - http://developer.apple.com/iphone/library/DOCUMENTATION/Networking/Conceptual/CFNetwork/
- cocoahttpserver on Google Code
  - http://code.google.com/p/cocoahttpserver/
- iPhone OS Man pages (for BSD sockets)
  - http://developer.apple.com/iPhone/library/documentation/System/Conceptual/ManPages_iPhoneOS/

For Next Class

- Game Kit Programming Guide
  - http://developer.apple.com/IPhone/library/documentation/NetworkingInternet/Conceptual/GameKit_Guide/