iOS – Developing Applications

Divesh Goyal
Mark Dostal
• Getting Started
• Objective-C basics
• Common design patterns
• API Key Concepts
  - Viewing maps
  - Performing Analysis
  - Collecting Data
• Q&A
ArcGIS - A Complete Geographic Information System

Author

Serve

Use

. . . For Authoring, Serving & Using Geographic Knowledge
ArcGIS Web & Mobile APIs

ArcGIS Server

REST

Web APIs
- Flex
- JavaScript
- Silverlight

Mobile APIs
ArcGIS API for iOS

- Build native applications using Objective-C
  - iPhone 3GS, iPhone 4, iPod Touch, iPad
  - iOS 3.1.2 and up
Web or Native applications?

• ESRI supports both

• Advantages of native applications
  - Tighter integration with other native apps
  - Access to resources
    - Contacts, calendar events, photos
    - Marketing/Hosting/Reporting via AppStore

• Disadvantages
  - Dedicated effort to write and maintain
Before you begin..

• You need an Intel-based Mac running OSX 10.6 (Snow Leopard)

• Join Apple’s iOS Developer Program
  - Standard : AppStore distribution
  - Enterprise : In-House distribution

• Download Apple’s iOS SDK (4.2.x) & Xcode IDE (4.0.x)

• ArcGIS API for iOS v2.0
Agenda

• Getting Started
• **Objective-C basics**
• Common design patterns
• API Key Concepts
  - Viewing maps
  - Performing Analysis
  - Collecting Data
• Q&A
Objective-C

- Its C
  - Semicolons and curly braces
  - Pointers (uh-oh)

- But not your average C
  - colons and square brackets too
  - A different syntax

- Thankfully, Cocoa Touch frameworks provide elegant APIs
Objective-C basics

• Class = Interface + Implementation

- **MyController.h**
  ```objective-c
  @interface MyController: UIViewController {
  // private variables here
  }
  // method declarations here
  @end
  ```

- **MyController.m**
  ```objective-c
  @implementation MyController
  // method implementations here
  @end
  ```
Objective-C basics

Contd.

• Protocol
  - Declaring a Protocol

```objective-c
@protocol UIApplicationDelegate
@required
// method definitions here
@optional
// method definitions here
@end

@interface MyDelegate: NSObject <UIApplicationDelegate>
@end
```

- Adopting a protocol

```objective-c
@interface MyDelegate: NSObject <UIApplicationDelegate> {
}
@end
```
**Objective-C basics**

Contd.

- Invoking methods = passing messages to objects

```
[object alloc];
```

<table>
<thead>
<tr>
<th>C# / Java</th>
<th>Objective-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>object.<strong>alloc</strong>();</td>
<td>[object <strong>alloc</strong>];</td>
</tr>
<tr>
<td>object.<strong>alloc</strong>().<strong>init</strong>();</td>
<td>[[object <strong>alloc</strong>] <strong>init</strong>];</td>
</tr>
<tr>
<td>object.<strong>setExtent</strong>(initExtent);</td>
<td>[object <strong>setExtent</strong>: initExtent];</td>
</tr>
<tr>
<td>object.<strong>addLayer</strong>(layer, name);</td>
<td>[object <strong>addLayer</strong>: layer <strong>withName</strong>: name];</td>
</tr>
</tbody>
</table>
Objective-C Basics

Contd.

• Messages are read like English
  - presentViewController:
  - writeToFile:
  - layerFailedToLoad:

• Can get verbose
  gestureRecognizer:shouldRecognizeSimultaneouslyWith
  GestureRecognizer:
Objective-C basics

Contd.

• Garbage collection is for kids, real developers manage their own memory

• You own an object if you
  - alloc  MyObject* foo = [MyObject alloc];
  - retain  [foo retain];
  - Or, copy  [foo copy];

• If you own an object, you’re responsible for releasing it  [foo release];
Objective-C basics

Contd.

- Properties make memory management easy’ier
- Syntactic sugar – dot notation

```objective-c
@interface MyController: UIViewController {
    MyObject* _foo;
}
@property (nonatomic, retain) MyObject* foo
@end

@implementation MyController
@synthesize foo = _foo;
@end

myController.foo = bar; //bar automatically retained
myController.foo = nil; //bar automatically released
```

Monitor memory footprint with *Instruments*
Agenda

- Getting Started
- Objective-C basics
- **Common design patterns**
- API Key Concepts
  - Viewing maps
  - Performing Analysis
  - Collecting Data
- Q&A
Common design patterns

- Model – View – Controller
Common Design Patterns

Contd.

• Delegation

windowShouldClose?

YES

WindowDelegate
Agenda

• Getting Started
• Objective-C basics
• Common design patterns
• API Key Concepts
  - Viewing maps
  - Performing Analysis
  - Collecting Data
• Q&A
What you can do with the API

• Display maps
• Perform analysis
• Visualize results
• Collect data
Displaying a Map

- UI Component: AGSMapView
  - Responds to gestures
    - Pinch to zoom
    - Drag to pan
    - Tap & Hold to magnify
  - Displays GPS location
    - Auto pan
Adding data to your map

- **Mashup layers**
  - ArcGIS Server Tiled map
  - ArcGIS Server Dynamic map
  - ArcGIS Server Image
  - Bing
  - Open Street Map
  - Graphics
  - Sketch

- **Open web maps**
  - ArcGIS.com
  - ArcGIS Portal
DEMO
Display a map
Respond to Map events through Delegates

Map Delegates

• Layer Delegate
  - Map/Layer loaded, failed to load
  - `<AGSMapViewLayerDelegate>`

• Touch Delegate
  - Tap, Double Tap, Tap and Hold
  - `<AGSMapViewTouchDelegate>`

• Callout Delegate
  - Did Show Callout, Did Click Accessory Button
  - `<AGSMapViewCalloutDelegate>`
Responding to Map Touch events

1. Adopt the Delegate protocol

```swift
@interface MyController: UIViewController <AGSMapViewTouchDelegate> {
}
```

2. Implement the protocol methods

```swift
@implementation MyController

- (void) mapView:(AGSMapView*) mapView
didClickAtPoint:(CGPoint) screen
didClickAtPoint:(AGSPoint*) mappoint
graphics:(NSDictionary*) graphics {
    //handle touch event
}
```

3. Set Delegate

```swift
self.mapView.touchDelegate = self;
```
DEMO
Displaying GPS location
Performing Analysis

Using Tasks

• Query, Find, Identify Task
  - Search for features in the map

• Geoprocessing Task
  - Spatial analysis using GP tools and models

• Locator
  - Geocode and reverse geocode addresses
Performing Analysis

Contd.

- Geometry Engine
  - native, high performance engine for performing geometric operations on the device

- Routing Task
  - Point-to-point and multipoint driving directions
  - Barriers, Time Windows, Best Sequence

- Closest Facility Task
  - Find nearest facility

- Service Area Task
  - Compute drive times and service areas
Common Pattern for using Tasks

1. Adopt the Task Delegate protocol

```objc
@interface MyController: UIViewController <AGSLocatorDelegate>
@end
```

2. Implement the protocol methods

```objc
- (void)locator:(AGSLocator*)locator
    operation:(NSOperation*)op
didFindLocationsForAddress:(NSArray*)candidates {
    //todo
}

- (void)locator:(AGSLocator*)locator
    operation:(NSOperation*)op
didFailLocationsForAddress:(NSError*)error {
    //todo
}
```
Common Pattern for using Tasks

3. Instantiate the task

```objective-c
self.locator = [AGSLocator locatorWithURL:[NSURL URLWithString:kGeoLocatorURL]];
```

4. Set Delegate

```objective-c
self.locator.delegate = self;
```

5. Perform operation

```objective-c
NSOperation* op = [self.locator locationsForAddress:addresses returnFields:outFields];
```
Visualizing Results

- **Graphics**
  - Geometry
  - Attribute
  - Symbol

- **Symbols**
  - Picture, Marker, Line, Fill
  - Composite
  - Text
Visualizing Results

Contd.

• Renderers
  - Simple
  - Unique Value
  - Class Breaks
  - Temporal
Visualizing Results

Contd.

• Callout
  - Displayed automatically when user taps on a graphic

• Content
  - Title & Detail
  - Image
  - Accessory button
  - Custom UI View
Specifying Content for the Callout

1. Adopt the Delegate Protocol

```objective-c
@interface MyController: UIViewController
<AGSInfoTemplateDelegate>
{}
@end
```

2. Implement the protocol methods

```objective-c
@implementation MyController

- (NSString *) titleForGraphic:(AGSGraphic*)graphic
    screenPoint:(CGPoint)screen
    mapPoint:(AGSPoint*)map {
    //todo
}

- (NSString *) detailForGraphic:(AGSGraphic*)graphic
    screenPoint:(CGPoint)screen
    mapPoint:(AGSPoint*)map {
    //todo
}
@end
```

3. Set the delegate on the graphic

```objective-c
AGSGraphic *graphic = ...;
graphic.infoTemplateDelegate = self;
```
DEMO

Using Tasks and Visualizing Results
Collecting Data

Using Feature layers & Popups

- Feature Layers edit data through Feature Services

- Popups provide UI to
  - Display and edit attributes
  - Manage attachments
  - View charts, media

- Popups configured through web maps
  - Attributes to display & edit
  - User friendly aliases and hints
  - Formatting for numbers, dates
Collecting Data

Using Popups

- Edit feature
  - Attributes
  - Geometry
  - Attachments
Editing Attributes

Using Popups

- Input based on field data type
- Support for
  - Subtypes
  - Domains
- Validation
  - Length
  - Numeric range
Managing Attachments

Using Popups

- View & Download
- Add
- Delete
Editing Geometry

- Use GPS location

- Use Sketch Layer
  - Interactively create & reshape geometries
  - Point, line, polygon
  - Undo, redo changes
Demo
Editing with Popups
Application Based on Device Type

- iPhone / iPod Touch
- iPad
- Universal app
Summary

• Getting Started
• Objective-C basics
• Common design patterns
• API Key Concepts
  - Viewing maps
  - Performing Analysis
  - Collecting Data
• Q&A
More Resources

• iOS Resource Center
  - Conceptual help, API Reference
  - Blog, Forums
  - Download API v2.0

• Samples on ArcGIS.com
  - ArcGIS for iOS Developer Samples group

• Web Course: Getting Started with the ArcGIS API for iOS
  - training.esri.com
Related Sessions

• **Wednesday, July 13th**
  - 3:15pm – 4:30pm – ArcGIS for iOS (31b)

• **Thursday, July 14th**
  - 10:40am – 11:00am – Road Ahead – ArcGIS for iOS (6b)
  - 1:30pm – 2:45pm - Esri Mobile Solutions Overview (15a)
  - 10:40am – 11:00am – Road Ahead – ArcGIS for iOS (6b)

• **Friday, July 15th**
  - 9:00am – 10:15am – ArcGIS for iOS (10)
  - 9:00am – 10:15am – Esri Mobile Solutions Overview (4)
Thank You

Help make this session better…

… Turn in your surveys.