iPhone Programming
CMSC 498I – Fall 2010

Text
Adam Porter
Today’s Topics

• Getting Text Input
• Configuring UI and Behavior
• Editable Text Containers – UITextField, UITextView
• Understanding unicode and encodings
• NSString’s companion classes
Keyboard

Display and Configuration
Virtual Keyboard

- Appears when needed
Virtual Keyboard

- Appears when needed
Virtual Keyboard

- Appears when needed
- Portrait and Landscape
Virtual Keyboard

- Appears when needed
- Portrait and Landscape
Virtual Keyboard

• Appears when needed
• Portrait and Landscape
• Adapted to task
Virtual Keyboard

• Appears when needed
• Portrait and Landscape
• Adapted to task
Virtual Keyboard

- Appears when needed
- Portrait and Landscape
- Adapted to task
Virtual Keyboard

- Appears when needed
- Portrait and Landscape
- Adapted to task

Phone Numbers
Virtual Keyboard

- Appears when needed
- Portrait and Landscape
- Adapted to task

Numbers
Virtual Keyboard

- Appears when needed
- Portrait and Landscape
- Adapted to task

Multi-Line Input
Virtual Keyboard

- Appears when needed
- Portrait and Landscape
- Adapted to task
- Supports many languages

Languages

40
English

There is a tide in the affairs of men,
Which, taken at the flood, leads on to fortune;
Omitted, all the voyage of their life
Is bound in shallows and in miseries.
Auguste Escoffier est un cuisinier français. Le « roi des cuisiniers, le cuisinier des rois », il modernisa et codifia la haute cuisine raffinée créée par marie-antoine carême et développa le concept de « brigade de cuisine » en rationalisant la répartition des tâches.
Шахматы — настольная логическая игра, сочетающая в себе элементы искусства, науки и спорта. Одна из древнейших игр на земле, сохранившихся до нашего времени; долгое время считалась игрой королей из аристократов.
Japanese Kana
Chinese Pinyin
Chinese Handwriting

- Simplified and Traditional
Working With The Keyboard

• Present the keyboard
• Configure keyboard
• Receive and handle input
• Dismiss the keyboard
Working With The Keyboard (cont.)

• Can’t interact with keyboard directly
  ▪ Interact with editable “Text Containers”
  ▪ Containers interface with keyboard

• Example editable text containers
  ▪ UITextField
  ▪ UITextView
Becoming First Responder

• View must be able to become first responder
  
  -(BOOL)canBecomeFirstResponder;

• View must be told to become first responder
  
  [aView becomeFirstResponder]

• Active first responder sent -resignFirstResponder

• Automatic for editable text containers (UITextField, etc...)
  
  -canBecomeFirstResponder returns YES

  -becomeFirstResponder sent when tapped
Presenting The Keyboard

- UIKit presents keyboard when editable text container becomes first responder
- Keyboard slides up from the bottom
- May need to resize / move text view to maintain visibility

- Watch for keyboard visibility notifications

```c
NSString *const UIKeyboardWillShowNotification;
NSString *const UIKeyboardDidShowNotification;
NSString *const UIKeyboardWillHideNotification;
NSString *const UIKeyboardDidHideNotification;
```

- Keys for getting values from keyboard notifications

```c
NSString *const UIKeyboardBoundsUserInfoKey;
NSString *const UIKeyboardAnimationDurationUserInfoKey;
NSString *const UIKeyboardAnimationCurveUserInfoKey;
```
Configuring The Keyboard

- Keyboard takes on traits of its target, the first responder

-UITextInputTraits
  - Protocol that all editable text containers support
  - Defines configurable options
UITextInputTraits

- Keyboard Type
- Appearance
- Secure Entry
- Capitalization
- Auto Correction
- Return Key Behavior
Configuration – Keyboard Type

- **UIKeyboardTypeDefault**
  - Displays users choice of keyboard (whatever language they want)

- **UIKeyboardTypeASCIICapable**
  - Not a generic Unicode input method, ASCII only

- **UIKeyboardTypeURL**
  - Features “.”, “/”, and “.com” prominently

- **UIKeyboardTypeNumberPad**
  - Displays numbers 0 - 9. Designed for PIN entry

- Others – **UIKeyboardTypeNamePhonePad, etc…**
Configuration – Appearance

- UIKeyboardAppearanceDefault
Configuration – Appearance

- `UIKeyboardAppearanceDefault`
- `UIKeyboardAppearanceAlert` – darker keyboard UI suitable for displaying with an alert panel
Configuration – Behavior

- **UITextAutocapitalizationType**
  - None, Words, Sentences, All Characters

- **UITextAutocorrectionType**
  - Default – Use the user’s choice from the Settings App
  - No, Yes – Force a particular setting
Configuration – Return Key

- **UIReturnKeyType**
  - Default – set the return key text to “return”
  - Go, Search, Send
  - Google, Yahoo

- **Return Key Enabling**
  - BOOL enablesReturnKeyAutomatically
  - Set this to **YES** if you want the return key disabled when there is no text in the editable text container
Dismissing The Keyboard

- To dismiss the keyboard send -resignFirstResponder
- Keyboard will then automatically slide out
Text Containers
Working With Text Containers

- **UITextField** – single line editor
- **UITextView** – multi-line editor
- API to grab text from the editable text containers
  
  ```
  // UITextField.h
  @property(nonatomic, copy) NSString *text;
  
  // UITextView.h
  @property(nonatomic, copy) NSString *text;
  ```

- Editing Sequence and Change “Events”
  - Did begin editing
  - Did end editing
  - Did change text
Text “Events”

- Mechanism you choose depends on your situation…
- “Did” events available to all mechanisms
- “Should” available only to the delegate
- Target / Action only for UITextField

<table>
<thead>
<tr>
<th>Event</th>
<th>Notification</th>
<th>Delegation</th>
<th>Target / Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>did begin editing</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>should begin editing</td>
<td></td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>did end editing</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>should end editing</td>
<td></td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>did change</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>should change</td>
<td></td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>– other options –</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
UITextField “Events”

• Target / Action

```swift
// UIControl.h
UIControlEventsEditingDidBegin
UIControlEventsEditingChanged
UIControlEventsEditingDidEnd
UIControlEventsEditingDidEndOnExit
```

• Example

```swift
// Set up action to be called when editing ends (could do this in IB)
UITextField *myTextField = ...;

[myTextField addTarget:self
      action:@selector(textDidEndEditing)
    forControlEvents:UIControlEventEditingDidEnd];
```
UITextField “Events”

• Notifications

NSString *const UITextFieldTextDidBeginEditingNotification;
NSString *const UITextFieldTextDidChangeNotification;
NSString *const UITextFieldTextDidEndEditingNotification;

• Delegation

// Begin
- (BOOL)textFieldShouldBeginEditing:(UITextField *)textField;
- (void)textFieldDidBeginEditing:(UITextField *)textField;

// End
- (BOOL)textFieldShouldEndEditing:(UITextField *)textField;
- (void)textFieldDidEndEditing:(UITextField *)textField;

// Change
- (BOOL)textField:(UITextField *)textField
shouldChangeCharactersInRange:(NSRange)range
replacementString:(NSString *)string;
UITextField – End Editing Sequence

- Keyboard does not automatically dismiss
- End editing by sending -resignFirstResponder
- End by moving focus (first responder) to another text field
- End when Return Key tapped
  - Delegate’s -textFieldShouldReturn:
  - Send -resignFirstResponder or pick another text field to -becomeFirstResponder

- (BOOL)textFieldShouldReturn:(UITextField *)textField {
  // When the return button is pressed, dismiss the keyboard!
  [textField resignFirstResponder]; // end editing, send end edit actions, etc...
  return YES; // dismiss keyboard
}
**UITextView “Events”**

- Not a **UIControl**, so no target / action

- Notification and delegation API similar to **UITextField**’s

```objective-c
NSString *const UITextViewTextDidBeginEditingNotification;
NSString *const UITextViewTextDidChangeNotification;
NSString *const UITextViewTextDidEndEditingNotification;
```

```objective-c
// Begin
- (BOOL)textViewShouldBeginEditing:(UITextView *)textView;
- (void)textViewDidBeginEditing:(UITextView *)textView;

// End
- (BOOL)textViewShouldEndEditing:(UITextView *)textView;
- (void)textViewDidEndEditing:(UITextView *)textView;

// Change
- (BOOL)textView:(UITextView *)textView
  shouldChangeTextInRange:(NSRange)range
  replacementText:(NSString *)string;
```
UITextView “Events”

• Other delegate methods

```cpp
// Called when selected text range changes, including cursor position
- (void)textViewDidChangeSelection:(UITextView *)textView;
```

• Notice there is no equivalent to “-textFieldShouldReturn:”
  
  ▪ Return key needs to go to the UITextView since it is multi-line
Summary

Putting It All Together
Putting It All Together

• **UITextView** or **UITextField** becomes first responder

• Keyboard slides up
  - Appearance notifications posted
  - Keyboard adopts the “input traits”

• While typing
  - Should change, did change notifications / delegate messages

• Dismiss the keyboard
  - User taps in another view that can become first responder
  - Manually resign, or transfer first responder in code
NString

Unicode and Companion Classes
NSString And Unicode

- **NSString**’s are not C char arrays
  - They are arrays of Unicode characters

- What is Unicode?
  - A universal character encoding standard
  - Used for representing characters of major world languages
Terms

• **Unicode** – standard which defines a “world” character set (UCS) mapping and encoding schemes

• **Code Point** – numeric value assigned to a unicode character

• **Glyph** – graphical representation of a character
  - Many-to-many relationship between characters and glyphs

• **UTF-8** – UCS encoding maximally compatible with ASCII

• **Precomposed** character is a Unicode entity that can be decomposed into an equivalent string of other characters

• **Canonical Equivalence** – Character orders may differ, but a “canonical” representation of them may be equivalent
Composed Characters

- Precomposed

- Decomposed

- Canonical
Code Point

• **Definition** - Numeric value used to represent a character

• Unicode defines a table representing major languages
  - Most commonly used characters fall in a 16-bit range
  - Unicode defines 21 bits of its 32 bit space; >1 million characters

• Each abstract character, given a name and numeric value
  - U+0391 - GREEK CAPITAL LETTER ALPHA
  - Using in an NSString `@"\u0391"`

• **Find Tables at** [http://www.fileformat.info/info/unicode](http://www.fileformat.info/info/unicode)
Encodings

• Specify how codes map to sequences of bytes

• ASCII encoding – 7 bit encoding of English/control characters

• Unicode specifies several encoding forms
  ▪ UTF-8, UTF-16, UTF-32
  ▪ Each maps a unicode character to one or more “units”
    ▪ For example, UTF-8 “unit” is one byte, character is 1-4 units
    ▪ UTF-8/16 use variable number of units, UTF-32 uses one unit

• UTF-8 is typically most compact, and is ASCII-compatible
Character / Letter

- The word “character” is used in multiple ways
  - char – ASCII base type in C programming
  - (unichar)characterAtIndex: – UTF-16 unit returned by NSString
  - Neither of these is what the user thinks of as a “letter”

- “letter” may require multiple UTF-8 or UTF-16 characters
  - Code point could require more than 16 bits, or…
  - Decomposition: Ä can legally be represented as Ä, or A + ̈
Some Take-Aways

- **NSString API presented in terms of UTF-16 units**
  - `-length` returns the number of UTF-16 units making up the string
  - `-characterAtIndex:` returns UTF-16 represented units (unichar)
    - Internal storage is not necessarily UTF-16
    - The actual Unicode character might span multiple indexes
Some Take-Aways

• Direct ‘character’ APIs should be avoided
  ▪ Use NSString companion classes
  ▪ “character” != byte
  ▪ Do not break up characters that are part of a decomposed seq.

• Example: Given the NSString storing: A + Ė
  ▪ User considers this a single “Letter” (displays as Ä )

• -characterAtIndex:0, returns “A”
• -length returns 2
• -rangeOfComposedCharacterSequenceAtIndex:0 returns {0,2}
NSScanner – Scans and interpret values from an NSString

NSCharacterSet – Represents a set of unicode characters

- An example – process file data by line

```swift
// lineSeparatorSet represents ‘\n’, ‘\r’, ...
NSCharacterSet *lineSeparatorSet = [NSCharacterSet newlineCharacterSet]

// Load a string from a file
NSString *string = [NSString stringWithContentsOfFile:...];

// Separate into individual lines
NSArray *lines = [string componentsSeparatedByCharactersInSet:lineSeparatorSet];
```