

# Getting Your Apps Ready for China and Other Hot Markets

Session 131

**Brent Ramerth**  
Software Engineer

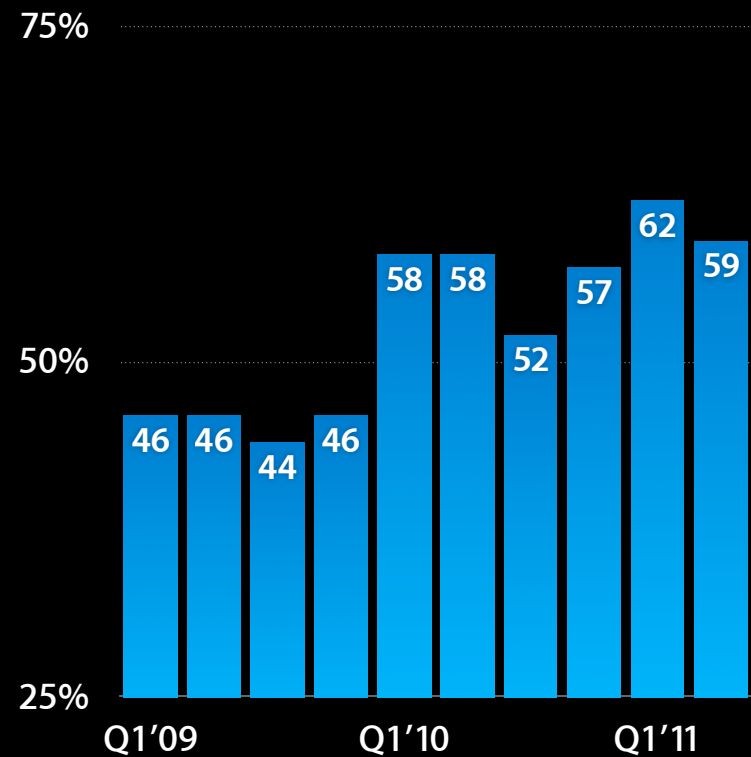
These are confidential sessions—please refrain from streaming, blogging, or taking pictures

# Introduction

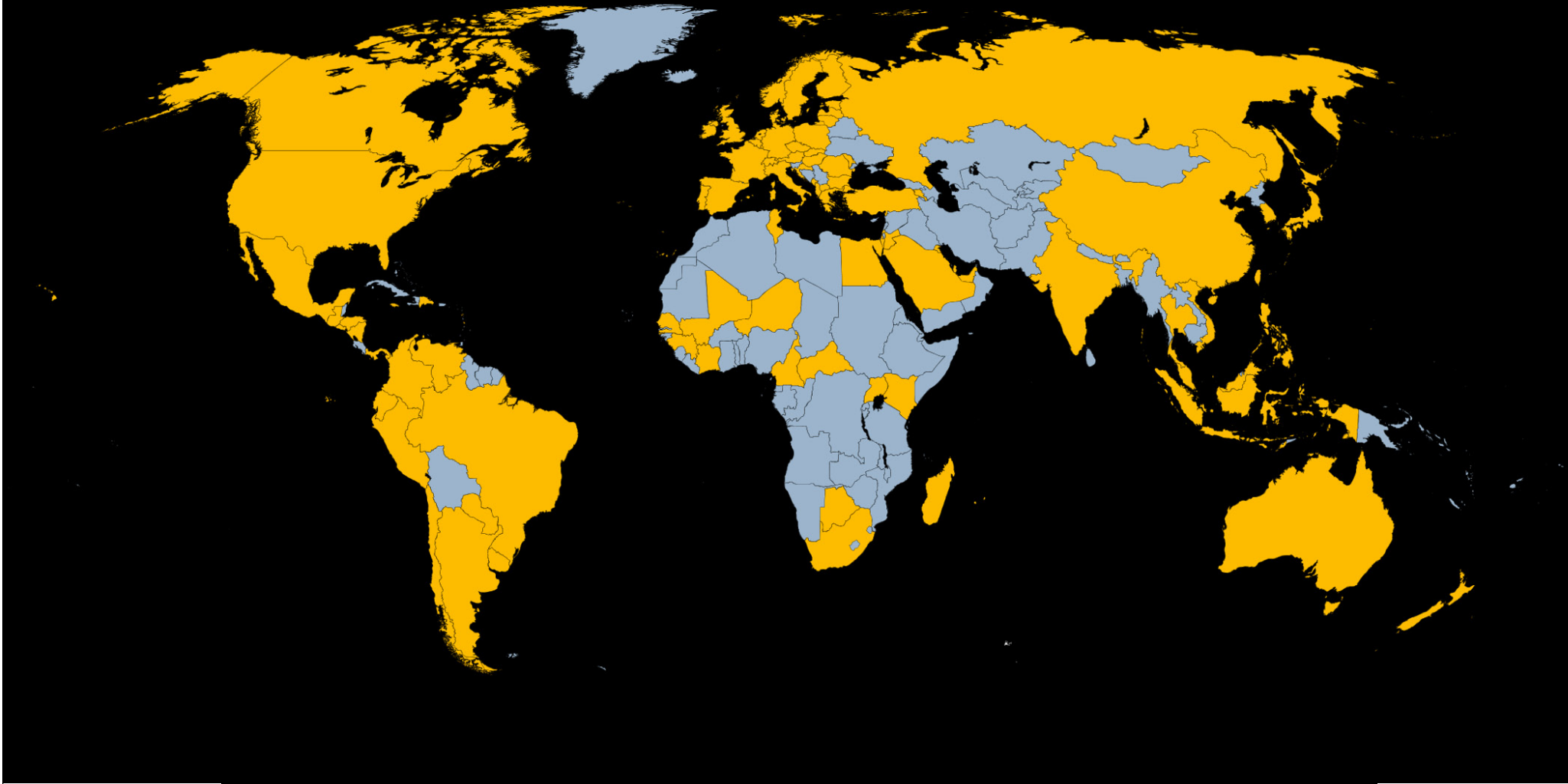
- Create a compelling experience for your international users
- Take advantage of the growing international market
- Learn the concepts involved in making an internationalized app

# International Markets

- Majority of Apple revenue comes from outside U.S.
- China is the fastest-growing market for Apple products
- Most customers outside of U.S. are not English native speakers



# Countries Shipping iPhone



# Localizable Languages

English

Arabic

Simplified Chinese

Traditional Chinese

Czech

Danish

Dutch

Finnish

French

German

Hungarian

Italian

Japanese

Korean

Norwegian

Polish

Portuguese (BR)

Portuguese (PT)

Russian

Spanish

Swedish

Turkish

Ukrainian

Croatian

Greek

Hebrew

Romanian

Slovenian

Thai

Indonesian

Malay

British English

Swedish

Catalan

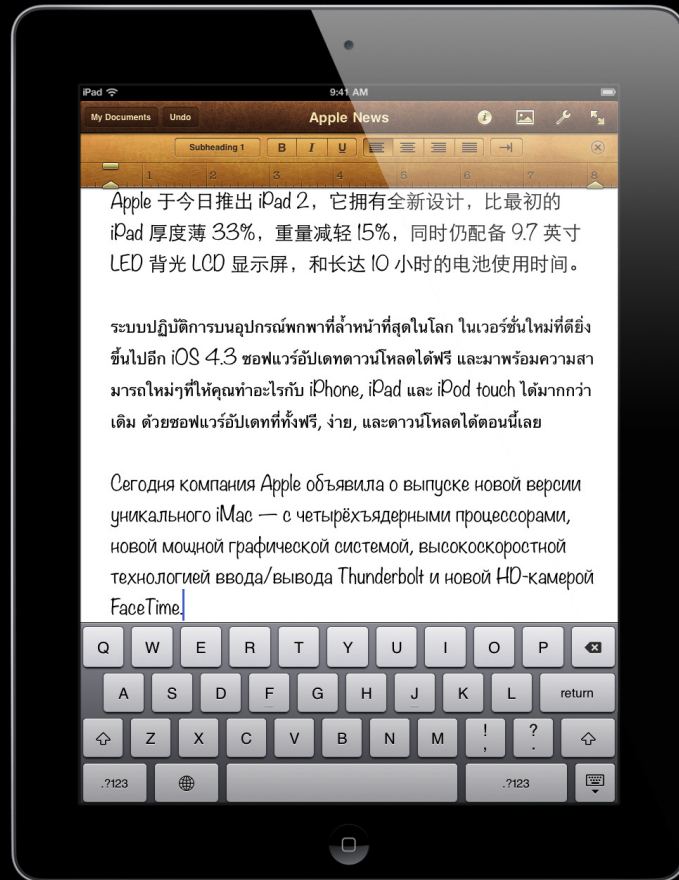
Vietnamese



# Internationalization vs. Localization

- Internationalization: designing for international compatibility
  - Text content (input, display, and processing)
  - Dates, times, numbers, calendars, time zones
- Localization: translation of your application's user interface

# Internationalizing Your Application



- Apple provides easy-to-use APIs
- Most of the work is done for you
- Goals:
  - One binary for all locales
  - Support content in any language

# Overview

- Internationalization architecture and guidelines
- International UI topics
  - User-interface guidelines
  - New APIs and features



# Internationalization Architecture

# Internationalization APIs

<b>NSLocale</b>	Current region, formats, and other properties
<b>NSNumberFormatter</b>	Formats and parses numbers
<b>NSDateFormatter</b>	Formats and parses dates and times
<b>NSCalendar</b>	Current calendar and associated operations
<b>NSTimeZone</b>	Current time zone and associated operations
<b>NSString</b>	Sorting, searching, and more

# Locales

- An identifier containing a language and usually a region
- Provides access to regional formatting standards
- Use the NSLocale API

# Locale vs. Localization

- “Locale” represents formatting standards
- “Localization” refers to the UI language
- User’s locale and localization usually match, but not necessary

# Region Format Differences



# Getting a Locale

- Automatically updating
  - +autoUpdatingCurrentLocale
- Static object
  - +currentLocale
  - +systemLocale
  - initWithLocaleIdentifier:

# Getting Locale Information

- Identify whether the locale uses the metric system

```
[locale objectForKey:NSLocaleUsesMetricSystem];
```

- Retrieve the currency code used in the locale

```
[locale objectForKey:NSLocaleCurrencyCode];
```

# Getting Locale Information

- Construct a quoted string with locale-sensitive quotes

```
bQuote = [locale objectForKey:NSLocaleQuotationBeginDelimiterKey];  
eQuote = [locale objectForKey:NSLocaleQuotationEndDelimiterKey];  
quotedString = [NSString stringWithFormat:  
    @"%@%@%@" , bQuote, myString, eQuote];
```

Chinese (China)	“iPhone”
Korean (Korea)	‘iPhone’
Japanese (Japan)	「iPhone」



# Common Errors

- Using the locale to identify the UI language

```
[NSString stringWithFormat:@"%d", [NSBundle mainBundle].preferredLanguages[0].languageCode];
```

- Using `+systemLocale` instead of `+currentLocale` for UI formatting

# Locale Demo

# Numbers

## Differences between locales

Digits	1,234.56	١,٢٣٤,٥٦
Decimal point and grouping separator	1,234.56	1 234,56
Currency	\$123.45	¥123.45
Percentage	45%	٤٥٪

# Formatting Numbers for Display

- Use NSNumberFormatter
- Convenience method

```
[NSNumberFormatter localizedStringFromNumber:numberStyle:]
```

- Create and maintain an NSNumberFormatter for:
  - Parsing numbers from strings
  - Creating a custom format

# Examples of Number Formatting

General	1,234.56
Currency	\$1,234.56
Percentage	123,456%
Scientific	1.23456E+03
Spell-out	One thousand two hundred thirty-four point five six

# Common Errors

- Formatting numbers with stringWithFormat:, printf, scanf, etc.

```
[NSNumberFormatterWithOptions] initWithDecimalStyle:NSNumberFormatterDecimalStyle];
```

English (U.S.)	241.23
Arabic (Egypt)	<del>٢٤١.٢٣</del> ✓

# Common Errors

- Erasing locale information by setting a constant pattern

```
[numberFormatter setNumberStyle:NSNumberFormatterCurrencyStyle];  
[numberFormatter stringFromNumber:myNumber];
```

English (US)	\$241.23
Chinese (China)	¥241.23  

# Dates and Times

## Differences between locales

Language of months, days, AM/PM, today	vendredi 10 juin 2011
Calendar in use	平成23年6月10日
Digits	शुक्रवार, १० जून २०११
12-hour vs. 24-hour time	2:15 PM vs. 14:15
Format, order of dates and times	10:00 ngày 10 tháng 6 năm 2011



# Formatting Dates and Times for Display

- Use NSDateFormatter
- Convenience method

```
[NSDateFormatter localizedStringFromDate:dateStyle:timeStyle:]
```

- Create and maintain an NSDateFormatter for:
  - Parsing dates and times
  - Using custom formats

# Standard Date and Time Formats

<b>Short</b>	6/10/11	11:03 AM
<b>Medium</b>	Jun 10, 2011	11:03:15 AM
<b>Long</b>	June 10, 2011	11:03:15 AM PDT
<b>Full</b>	Friday, June 10, 2011	11:03:15 AM Pacific Daylight Time

# Custom Date/Time Formats

- When the default formats don't meet your needs
- Use `+dateFormatFromTemplate:options:locale:`

```
NSString *formatString = [NSDateFormatter  
    dateFormatFromTemplate:@"dMMM" options:0  
    locale:[NSLocale currentLocale]];  
[dateFormatter setDateFormat:formatString];  
[dateFormatter stringFromDate:[NSDate date]];
```

French (France)	10 juin
Chinese (China)	6月10日

# Common Errors

- Using a static format for displaying dates in UI

```
[NSDateFormatter *formatter = [NSDateFormatter new];  
[formatter setDateStyle:NSDateFormatterNoStyle];  
[formatter setTimeStyle:NSDateFormatterNoStyle];  
NSString *myDate = [formatter stringFromDate:myDate];
```

English (U.S.)

06/10/11

Chinese (China)

06月10日



# Common Errors

- Storing/parsing a date without fixing the time zone

```
[dateFormatter setDateZone:[NSTimeZone timeZoneWithName:@"GMT:0"]];  
date = [dateFormatter dateFromString:@"2011-06-10 17:46:23"];  
[NSDateFormatter localizedStringFromDate:date  
    dateStyle:NSDateFormatterLongStyle  
    timeStyle:NSDateFormatterLongStyle];
```

In San Francisco (GMT-07:00)

June 10, 2011 5:46:23 PM



In Beijing (GMT+08:00)

2011年06月10日 17:46:23



# Common Errors


- Storing/parsing a date without fixing the locale

```
[dateFormatter setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
[dateFormatter setDateFormat:@"'yyyy'-'MM'-'dd' 'HH':'mm':'ss'"];
date = [dateFormatter dateFromString:@"2011-06-10 17:46:23"];
[NSDateFormatter localizedStringFromDate:date
    dateStyle:NSDateFormatterLongStyle
    timeStyle:NSDateFormatterLongStyle];
```

Retrieved in U.S. English locale

June 10, 2011 10:46:23 AM

Retrieved in Arabic locale

 ١٠ يونيو، ٢٠١١ ١٠:٤٦:٢٣ ص جرينتش-٠٠



# Calendars

## Differences between calendars

Year	2011, 1432, 2554, 5771
Era	AD, Heisei
Number of months per year	12, 13, variable
Lengths of months	From 5 to 31 days
First day of week	Saturday, Sunday, Monday
When years change	昭和64年1月7日 → 平成1年1月8日

# Calendar Operations

- Use NSCalendar for calendrical calculations
  - Number of days in month, weeks in year, etc.
  - Components of a date
  - Delta computations
- Mac OS X Lion supports several non-Gregorian calendars
- iOS 5.0 supports Gregorian-offset calendars (Thai Buddhist, Japanese)



# Strings and Unicode

- Sorting, searching, and case operations are locale dependent
- Tokenization behavior depends on text language
- Use NSString APIs for localized string operations

# NSStrings: Sorting and Comparison

- Different languages have different sort orders
- Diacritic (accent) significance and handling vary
  - German: u and ü are different letters
- Sorting may use language-specific knowledge
- Use localized sorts for user-visible strings

# Sorting Example

Chinese

compare:

Unicode Codepoint

公孙胜  
吴用  
宋江  
李逵  
林冲  
花荣  
鲁智深

# Sorting Example

## Chinese

~~compare:~~

Unicode Codepoint

公孙胜  
吴用  
宋江  
李逵  
林冲  
花荣  
鲁智深

localizedStandardCompare:

Pinyin Pronunciation

公孙胜	Gong Sunsheng
花荣	Hua Rong
李逵	Li Kui
林冲	Lin Chong
鲁智深	Lu Zhishen
宋江	Song Jiang
吴用	Wu Yong

Ordering is based on the pronunciation of the character in Mandarin

# Common Errors

- Using diacritic or case insensitivity for sorting
  - Diacritics are considered in sorting in some locales
  - Some languages sort uppercase first, others lowercase
  - Diacritic and case insensitivity are for searching, not sorting

# NSString: Searching for Substrings

- Use `[NSString rangeOfString:options:range:locale:]`
- Case- or diacritic-insensitive comparisons can vary by language
  - Turkish: Lowercase İ is i; uppercase İ is I
- Example: Localized case-insensitive search in Turkish for “iki”

```
myString = @"İki Dil Bir Bavul";  
searchRange = NSRange(0, [myString length]);  
[myString rangeOfString:@"iki" options:NSCaseInsensitiveSearch  
    range:searchRange locale:[NSLocale currentLocale]];
```

- See also `NSTextFinder` (new in Lion)

# NSString: Tokenizing Strings

- Use `enumerateSubstringsInRange:options:usingBlock:` to tokenize natural language
  - Lines
  - Paragraphs
  - Sentences
  - Words
  - Composed character sequences

# NSString: Tokenization

```
range = NSMakeRange(0, [myString length]);
[myString enumerateSubstringsInRange:range
 options:NSStringEnumerationByWords
 usingBlock: ^(NSString *substring, NSRange substringRange,
              NSRange enclosingRange, BOOL *stop) {
    NSLog(@"%@", substring);
}
];
```

Chinese (China)

我明天要去图书馆。

English (U.S.)

Tomorrow, I will go to the library.



# Common Errors

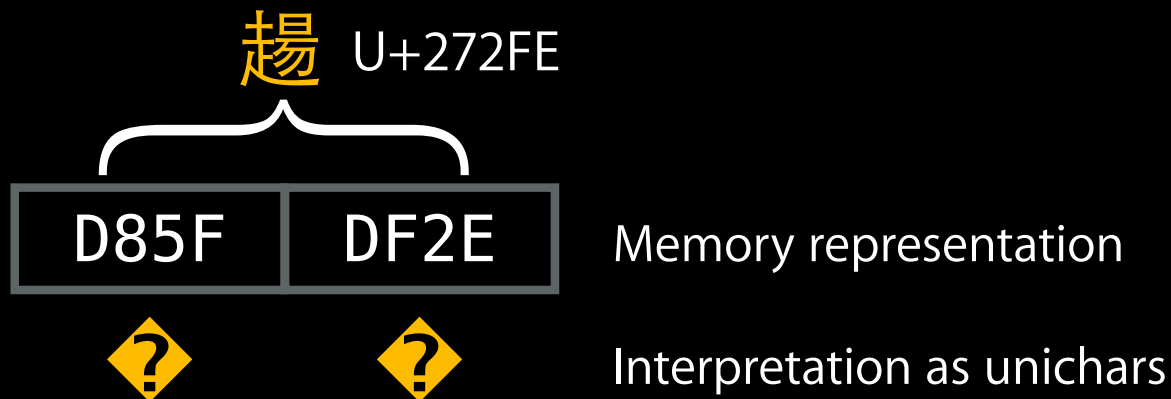
- Assuming words and lines are separated by white space
  - Japanese, Chinese use no white space at all  
諸国間の友好関係の発展を促進することが肝要であるので
  - Thai: white space separates phrases (roughly)  
อักษรไทย เป็นอักษรที่ใช้เขียนภาษาไทย และภาษากลุ่มน้อยอื่นๆ

# Common Errors

- Assuming one glyph corresponds to one unichar
  - É could be stored as one or two unichars
    - U+00C9 Latin Capital Letter E with Acute
    - U+0045 Latin Capital Letter E + U+0301 Combining Acute Accent
- Canonicalization does not solve the problem
  - `[string precomposedStringWithCanonicalMapping];`
  - Some combining character sequences have no precomposed variants
  - Characters outside of 16-bit range cannot be canonicalized

# Common Errors

- U+10000 and up are stored as two unichars (surrogate pairs)
  - Ideographic character extensions, historic scripts



- When iterating by characters, use `enumerateSubstringsInRange`

# International UI Topics

# Input Methods

- Input methods are used to type complex scripts
- May generate single or multiple candidates

pin yin shu ru fa

1 拼音输入法 2 拼音 3 品 4 贫 5 聘



拼音输入法

- Input has two stages
  - Marked: Input has been received but not converted
  - Committed: Input is converted and marked text is replaced

# Input-Method UIs

- Wait until marked text is committed before processing it
- Identifying marked text



textRange

isMarkedText, -markedRange

# Other Input-Method Notes

- iOS UI
  - Candidate bar used for Chinese and Japanese input methods
  - Your app should be resilient to changes in keyboard UI size
- OS X UI
  - Don't intercept key events, especially space and return

# Input-Method Demo



# Bidirectional Text

- Arabic and Hebrew are predominantly written from right to left
- Left-to-right text may be embedded as well

Visual Ordering شركة أبل (Apple Inc) هي شركة

Logical Ordering

# Handling Bidirectional Text

- Visually adjacent text may not be logically adjacent
- Don't make directionality assumptions
- Let text have its natural alignment and direction
  - OS X: `[NSParagraphStyle setAlignment:]`
  - iOS: `[UITextInput setBaseWritingDirection:]`



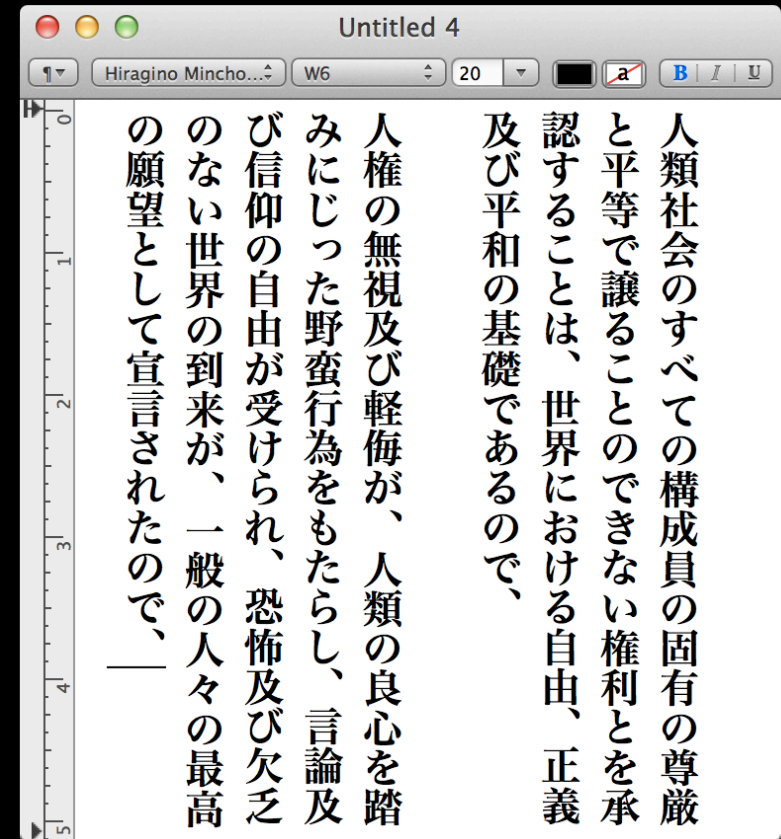
# Vertical Text

New

- Japanese traditionally written vertically
- Cocoa now supports vertical text editing and display

```
[textView setLayoutOrientation:  
    NSTextLayoutOrientationVertical];
```

- Vertical glyphs substituted by default
- Use a horizontal i-beam cursor



# Emoji

New

- New in OS X 10.7, now available in all locales in iOS 5.0



- Emoji previously only available in Japan prior to standardization
- Both iOS and OS X have color-font support for Emoji
- Emoji are Supplementary Multilingual Plane characters
- See Unicode 6.0 specification for more details

# Summary

- Regional formats and standards vary widely from locale to locale
- Test your app in scenarios that exercise international functionality
- Use Foundation APIs, standard UI controls, and built-in formats

# Related Sessions

Advanced Text Processing	Mission Thursday 3:15PM
What's New in Cocoa	Presidio Tuesday 10:15AM
Performing Calendar Calculations	Russian Hill Wednesday 4:30PM
Rich Text Editing in Safari on iOS	Pacific Heights Thursday 9:00AM

# Labs

Internationalization Lab

App Frameworks Lab A  
Friday 11:30AM

# More Information

**Apple Developer Forums**

<http://devforums.apple.com>



