Performing Calendar Calculations

Session 117

Chris Kane
Software Engineer

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

What We Will Cover

- Review calendars and calendrical calculations
- Introduce the APIs related to calendars, dates, and times
- Discuss troublesome calendrical issues

Labeling Time

- Time is a continuum
- People need to describe time
  - When events occurred
  - Amounts of time
- What to do?
  - Could count days
  - Today is “Day #410957”

Calendar Components

- Human inventions to describe event times
  - Count recurrences various natural cycles
  - Group and decompose them into smaller, human-tractable quantities
    - We will call these counters “units” and “components”
- Unit of “day” is too granular, so it is decomposed (hours, minutes, …)
- But “day” also too fine-grained, so we have years and months
  - “8 June, 2011” is easier to deal with than “Day #410957”

Different Calendars

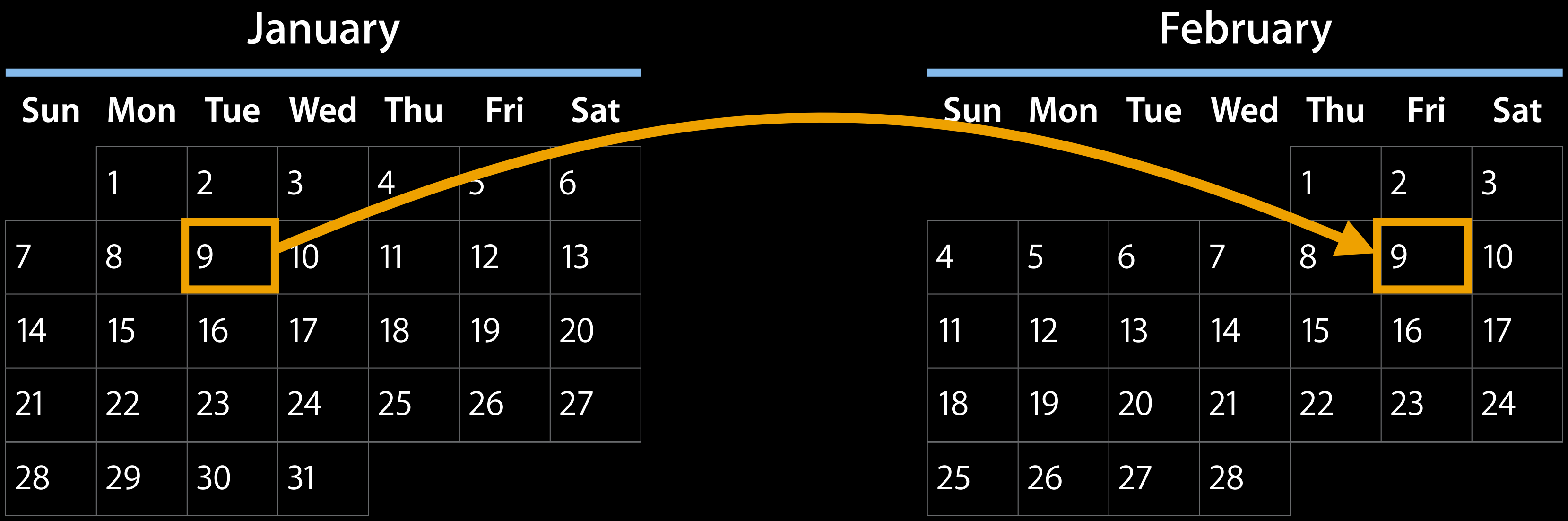
- Many different calendars evolved over the millennia
- Gregorian, Islamic, Japanese, Hebrew, Chinese, Indian, …
  - Gregorian calendar is the calendar used in Europe, North and South America, and many other parts of the world
- Each has unique ways of counting and grouping and describing eras, years, months, and days

Calendrical Calculations

- Arithmetic-like operations on calendar components
  - What day is 90 days from today?
  - How many weeks until my next birthday?
- Conversions between calendars
- Calendrical calculations are subtle

Example: Add One Month to a Date

9 January + 1 month = 9 February



January

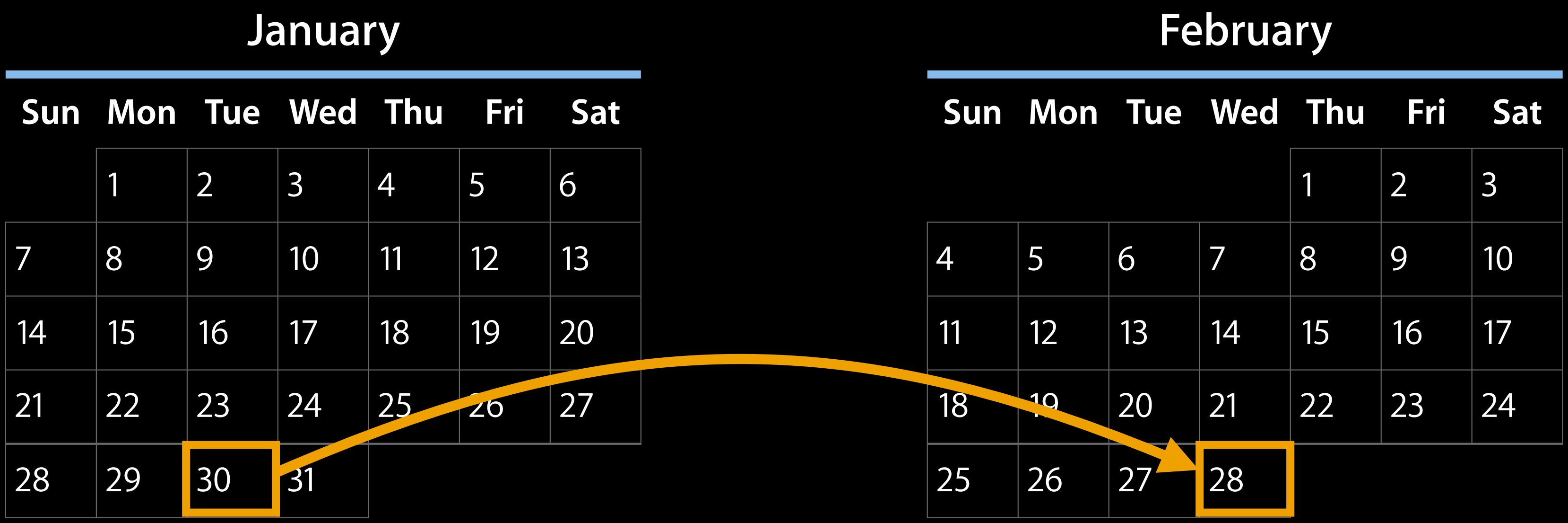
Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

February

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

When Best Result Does Not Exist

30 January + 1 month = 28 February



January

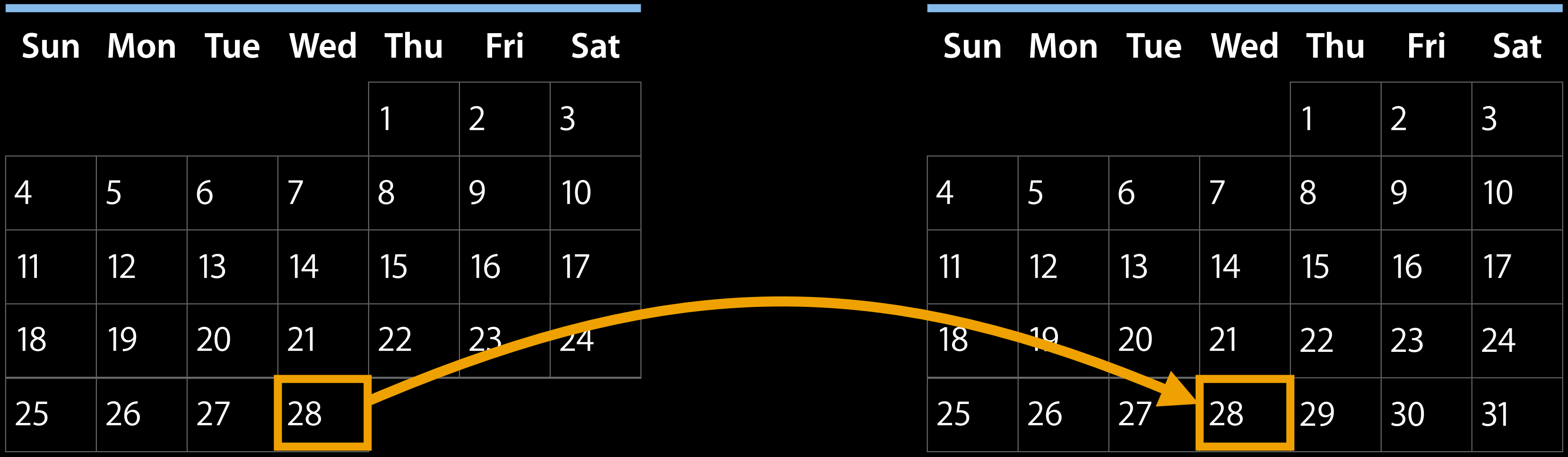
Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

February

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

Add Another Month

28 February + 1 month = 28 March



February

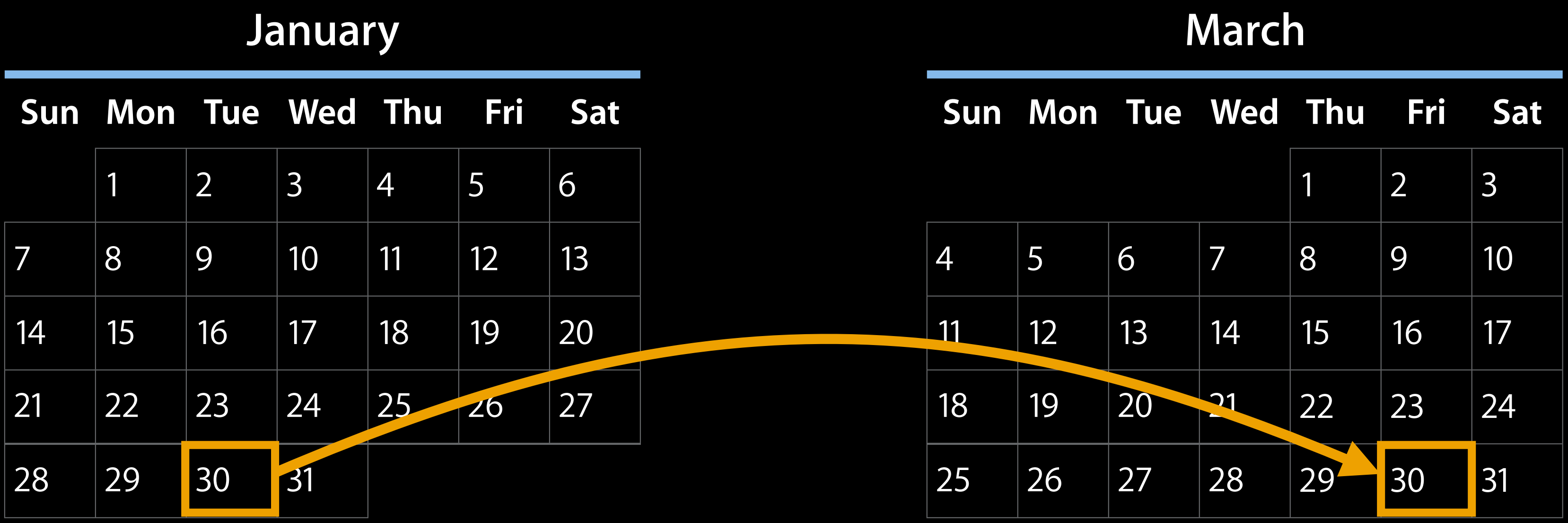
Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

March

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Or Instead, Add Two Months

30 January + 2 months = 30 March



January

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

March

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

So…

- 30 January + 1 month + 1 month = 28 March
- 30 January + 2 months = 30 March
- Calendrical calculations are subtle
- Generally, smaller components are preserved
- Do not do calendrical calculations yourself!
- But how you use the operations also matters…

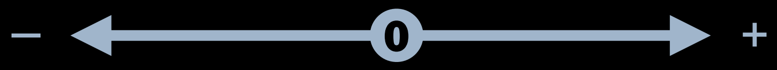
Calendar and Time APIs

Cocoa APIs

- NSTimeInterval
- NSDate
- NSCalendar
- NSDateComponents
- NSTimeZone
- NSDateFormatter

Calendar-Independent Time Scale

- Seconds since the reference date
  - NSTimeInterval
  - A floating-point number of seconds after the reference date, in the reference time zone (GMT/UTC)
- Value is currently about +329 million (seconds)



Reference Date

Gregorian	1 January 2001 00:00:00 GMT
Islamic	7 Shawwal 1421 00:00:00 GMT
Japanese	1 January 平成13 00:00:00 GMT
Hebrew	6 Tevet 5761 00:00:00 GMT
Republic of China	1 January 90 00:00:00 GMT
Indian	11 Pausa 1922 00:00:00 GMT

NSDate

- Object which contains a NSTimeInterval
  - The number of seconds after the reference date in GMT
- No associated calendar
- Time zone is GMT

NSCalendar

- Represents calendars
- Knows about the "arithmetic" properties of calendars
  - How many months are in a year
  - How many hours are in a day
  - How NSTimeIntervals map to and from calendar dates
- Know how to do calendrical calculations

NSDateComponents

- Object containing a set of calendar components
- Component values are signed integers
- Can be used to hold a set of absolute or relative components
  - Absolute: exact values
  - Relative: amounts
- Example: hour value “4” and minute “15” could mean:
  - Absolute: 4:15 (am)
  - Relative: 4 hours and 15 minutes

NSTimeZone

- Represents a time zone
  - A geopolitical region which defines a set of rules for how local time is calculated from the reference time zone (GMT/UTC)
- Hours and minutes offset from GMT
- Knows when “Daylight Saving Time” or “Summer Time” occur
  - A transition to a different offset from GMT
- Governments change the rules of their time zones from time to time

Calendrical Calculations

Common NSCalendar Operations

- components:fromDate:
- dateFromComponents:
- dateByAddingComponents:toDate:options:
- components:fromDate:toDate:options:

Causes of Trouble

- Calendar irregularities
- Ambiguities in a set of components
  - Nonexistent dates
  - Multiple matching dates

Irregularities

- Leap day in Gregorian calendar
  - 29 February
- Time zone transitions
  - Forward transitions cause an hour skip
  - Backward cause an hour to occur twice
- Dateline transitions
  - Samoa will skip 30 December, 2011

Hebrew Calendar

- Some months sometimes have 29 days, other years 30
- Year has either 12 or 13 months
- Months are numbered: 1, 2, 3, 4, 5, 6, (7), 8, 9, 10, 11, 12, 13
    - (7) is leap month

Japanese Imperial Eras

- Calendar same as Gregorian, except for year numbering
- Years are numbered from the start of the emperor’s reign
  - 31 December, Showa 63
  - 1 January, Showa 64
  - …
  - 7 January, Showa 64
  - 8 January, Heisei 1
- So, year number may change in the middle of a year

Example: Advancing by Days

- Given a starting date
- While some condition remains true
    - Perform some operation
    - Advance to the next day, at the same time

Advancing by Days

```
NSDate *date = [NSDate dateWithString:@”2011-01-01 00:00:00 +0000”];
while (... condition ...) {
    // perform operation
    ....
    date = ... advance date to next day midnight ...
}
```

86400

(number of seconds in a day)

Attempt #1: Add 86400 seconds

```
NSDate *date = [NSDate dateWithString:@”2011-01-01 00:00:00 +0000”];
while (... condition ...) {
    // perform operation
    ....
    date = [date dateByAddingTimeInterval: 86400];
}
```

Attempt #1: Add 86400 seconds

- 1 January 00:00:00 + 86400 seconds = 2 January 00:00:00
- 2 January 00:00:00 + 86400 seconds = 3 January 00:00:00
- …
- 13 March 00:00:00 + 86400 seconds = 14 March **01**:00:00 (2011, in U.S.)
- 14 March **01**:00:00 + 86400 seconds = 15 March **01**:00:00

Attempt #2: Add 1 Day

```
NSCalendar *calendar = ...
// create an NSDateComponents with “1 day”:
NSDateComponents *dc = [[NSDateComponents new] autorelease];
[dc setDay: 1];
NSDate *date = ...
while (... condition ...) {
   // perform operation
   date = [calendar dateByAddingComponents:dc toDate:date options:0];
}
```

Attempt #2: Add 1 Day

- 1 January 00:00:00 + 1 day = 2 January 00:00:00
- 2 January 00:00:00 + 1 day = 3 January 00:00:00
- …
- 13 March 00:00:00 + 1 day = 14 March **00**:00:00 (2011, in U.S.)
- 14 March **00**:00:00 + 1 day = 15 March **00**:00:00

Causes of Trouble

- Calendar irregularities
- Ambiguities in a set of components
    - Nonexistent dates
    - Multiple matching dates

Ambiguities

- A set of components can be ambiguous
- A date with those components may not exist
  - 37 June, in Gregorian calendar
  - 29 February, most years
- Multiple possible dates may exist
  - Tuesday at 16:00
  - Hour repeated during summer time back to standard time transition

Nonexistent Dates

- Nominal result date of arithmetic may not exist
- Time zone DST forward transitions
  - 01:59:59 + 1 second = 03:00:00 (in U.S.)
- <day before transition> 02:20 + 1 day = <day of transition> **???**:20

Return to Advancing-by-Days Example

- In Brazil, time zone transitions occur at “midnight”: 23:59:59 -> 01:00:00
- <day before transition> 00:00 + 1 day = <day of transition> **???**:00
- Loop continues…
- <day of transition> **???**:00 + 1 day = <day after transition> **???**:00
- … Just like when we added 1 month to 30 January, then again

Attempt #3

- Previous: add 1 day to the current working date to get next date
- Instead: add an increasing number of days to the original date

Attempt #3: Add Increasing Number of Days

```
NSCalendar *cal = ...;  NSDate *original = ...;
NSDateComponents *dc = [[NSDateComponents new] autorelease];
NSInteger numDays = 0;
NSDate *date = original;
while (... condition ...) {
   // perform operation
   numDays++;
   [dc setDay:numDays];
   date = [cal dateByAddingComponents:dc toDate:original options:0];
}
```

New Result in Brazil

- <original> 00:00 + N days = <day before transition> 00:00
- <original> 00:00 + (N+1) days = <day of transition> **???**:00
- <original> 00:00 + (N+2) days = <day after transition> **00**:00
- … Just like when we added 2 months to 30 January

Sidebar: Avoid Stressing Boundaries

- Midnight; end of the year; the year 1
- Problematic: Brazilian time zone transition at midnight
  - Other countries transition at 1 am or 2 am
  - Better: use noon instead of midnight as "don’t care" time
- Better: Samoan time zone change NOT skipping 31 December
- Problematic: using NSDate objects to represent “just a time”
  - Developer uses year 1, month 1, day 1, plus desired time
  - Better: use the date of time interval 0.0, + time

Week-based Calendars

Week-based Calendars

- Week: a cyclic period of 7 days (weekdays)
- Any calendar can be interpreted in a week-based fashion
- Can be convenient when doing calculations with weeks and weekdays
- How can you specify a given day?
  - {Year, Day # within year}: {2011, 159}
  - {Year, Month, Day # within month}: {2011, 6, 8}
  - {Week-based Year, Week # within year, Weekday}: {2011, 23, Wednesday}

Week-based Calendars Defined

- A week-based calendar has an integral number of weeks
- Two properties define a week-based calendar
  - The weekday which is the beginning of the week (and year)
  - Minimum number of days a straddling week needs in the new year to be considered the first week of that new year
- ISO 8601 defines a week-based calendar

First Week of Year

Suppose first day of the week is Monday

	2010-12		2011-01	
Saturday	18	25	1	8
Sunday	19	26	2	9
Monday	20	27	3	10
Tuesday	21	28	4	11
Wednesday	22	29	5	12
Thursday	23	30	6	13
Friday	24	31	7	14

First Week of Year

Suppose first day of the week is Monday

	2010-12		2011-01	
Saturday	18	25	**1**	8
Sunday	19	26	**2**	9
Monday	20	**27**	3	10
Tuesday	21	**28**	4	11
Wednesday	22	**29**	5	12
Thursday	23	**30**	6	13
Friday	24	**31**	7	14

First Week of Year

Suppose first day of the week is Monday

	2010-12		2011-01	
Saturday	18	25	1	8
Sunday	19	26	2	9
Monday	20	27	3	10
Tuesday	21	28	4	11
Wednesday	22	29	5	12
Thursday	23	30	6	13
Friday	24	31	7	14

First Week of Year

Suppose first day of the week is Monday

	2010-12		2011-01	
Saturday	18	25	1	8
Sunday	19	26	2	9
Monday	20	27	3	10
Tuesday	21	28	4	11
Wednesday	22	29	5	12
Thursday	23	30	6	13
Friday	24	31	7	14

Week-based Calendar Trouble

- Year number for days in a week-based calendar interpretation may not be same as for ordinary calendar
- Must not mix ordinary year number with week-based components
- Nor week-based year number with ordinary components
- Can cause ambiguity
    - There is no 2 January in “2011” (ISO 8601)
    - 2 January 2011 (ordinary year) is {2010, 52, 7}
    - “2 January 2011” (week-based) is {2012, 1, 1}

New NSCalendar API

New

- NSYearCalendarUnit constant specifies ordinary calendar year
- New component types in Mac OS X 10.7 and iOS 5.0
  - NSWeekOfYearCalendarUnit
  - NSWeekOfMonthCalendarUnit
  - NSYearForWeekOfYearCalendarUnit
  - Use of NSWeekCalendarUnit discouraged

Date Formatting and Parsing

NSDateFormatter

- An object to convert dates to strings and strings to dates, in a locale-sensitive way

Date Formatting Gotcha #1

- Suppose a developer writes this code in 2010:

```
NSDateFormatter *df = [NSDateFormatter new];
... other configuration ...
[df setDateFormat:@”YYYY-MM-dd”]; // want strings like “2011-01-01”
```

- This code appears to work
- But for 1 January 2011, yields “2010-01-01”
- “YYYY” is week-based calendar year
- “yyyy” is ordinary calendar year

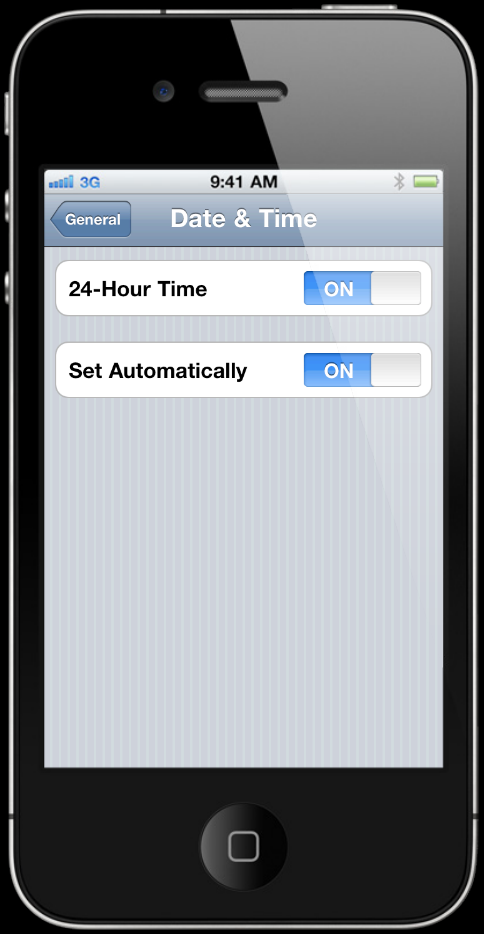
Date Formatting Gotcha #2

```
[df setDateFormat:@”yyyy–MM–dd HH:mm:ss”]; // hour: “00” – “23”
```

- For 1 January, 2011, 2pm, this yields one of:
  - “2011-01-01 14:00:00”
  - “2011-01-01 02:00:00 pm”
- For parsing, first string will succeed for some users, fails for others

Date Formatting Gotcha #2

- Date formatters start with the current user locale
- Some user preferences override even a specifically set format pattern
- 24-hour clock setting overrides set format pattern
- So “hour” result depends on user’s setting:
  - “2011-01-01 14:00:00”
  - “2011-01-01 02:00:00 pm”



24-Hour Time ON

Set Automatically ON

Date Formatting Gotcha #2

- For cases where the current user locale should not be used, the locale needs to be set on the date formatter
- For “internet” date strings, “en_US_POSIX” locale often works well

```
NSLocale *locale;
locale = [[NSLocale alloc] initWithLocaleIdentifier:@”en_US_POSIX”];
[df setLocale: locale];
[locale release];
```

More Information

Bill Dudney
Application Frameworks Evangelist
dudney@apple.com

Documentation
Mac OS X and iOS Foundation
http://developer.apple.com/cocoa

Apple Developer Forums
http://devforums.apple.com

Labs

Cocoa, Autosave, File Coordination and Resume Lab	Application Frameworks Lab A Thursday 2:00-4:00PM

Take-Away Points

- Use the system calculation algorithms
- Care must be taken when using them
- Avoid boundary conditions
- Try to imagine interesting boundary cases for testing

WWDC2011