

iOS Dev Accelerator

Week 6 Day 2

- UIDatePicker & UIPickerView
- Date Handling
- Advanced Fetching

UIDatePicker

UIDatePicker

- “The UIDatePicker class implements an object that uses multiple rotating wheels to allow users to select dates and times”
- Can also be used as a countdown timer.
- Has a date property that simply returns the current selected date in the date picker
- You can also hook up an IBAction that fires anytime the date has changed
- Has an intrinsic width of 320, you can squeeze it down to 240 and still see everything though
- Has options in the storyboard attribute inspector

Demo

UIPickerView

- UIPickerView is similar to a DatePicker, but allows you to fully customize what the picker displays instead of just Dates & Times.
- Basic Workflow for a UIPickerView:
 - Drag out your picker view from storyboard and make an outlet, or create the picker view in code.
 - Set your view controller to be the pickers datasource
 - Implement:
 - `numberOfComponentsInPickerView:`
 - `pickerView:numberOfRowsInComponent:`
 - `pickerView:titleForRow:ForComponent:`

NSDate & NSDateFormatter

Dates & Time

- There are 3 primary classes you will use when representing time in your apps:
 - NSDate - a representation of a single point in time
 - NSCalendar - a representation of a particular calendar type (Buddhist, Chinese, Hebrew, etc)
 - NSDateComponents - a representation of particular parts of a date, like an hour, minute, day, year, etc.

NSDate

- NSDate represents a single point in time.
- Date objects are immutable once created
- The standard unit of time for date objects is floating point value typed as NSTimeInterval expressed in seconds.
- NSDate computes times as seconds relative to an absolute reference time: the first instant of January 1, 2001, GMT.
- Dates before that time are stored as negative values. Dates after stored as positive.

Creating Date Objects

- To get a date that represents the exact current time:
 1. just init a NSDate instance OR
 2. use the class method date on NSDate
- To create a specific date:
 1. Use one of NSDate's initWithTimeInterval init methods
 2. Use an NSCalendar and date components

Demo

Basic Date Calculations

- There are many ways to compare dates:
 - isEqualToDate:
 - compare:
 - laterDate:
 - earlierDate:
- and you can also use the method `timeIntervalSinceDate:` on `NSDate` to get the time interval from 2 dates

Calendars

- You use calendar objects to convert between times and date components (years, days, minutes, etc)
- `NSCalendar` provides an implementation for many different types of calendars.
- Calendar types are specified by constants in `NSLocale`.
- the class method `currentCalendar` returns the user's preferred locale.

Date Components

- You can represent a component of a date using the NSDateComponent class.
- Use methods setDay:, setMonth:, setYear:, etc to set those individual components.

Date Calculations

- Use the `dateByAddingComponents:toDate:options:` on a calendar to add components (hours, minutes, days, years) to an existing date. They can be negative values to take time away as well.

New date convenience methods

- `[calendar component:NSCalendarUnitHour fromDate: [NSDate date]]` - returns a single component of your specification from a date
- `[calendar componentsInTimeZone: fromDate:]` returns date components shifted to the time zone of your specification.
- `[calendar components: fromDateComponents :toDateComponents :options]` returns the difference between two NSDateComponents instances.

New date comparison methods

- `isDateInToday`: returns true if date is today
- `isDateInTomorrow`:
- `isDateInYesterday`:
- `isDateInWeekend`:
- `isDate: inSameDayAsDate`:
- `isDate: equalToDate: toUnitGranularity`:

Demo

NSDateFormatter

- NSDateFormatter is a subclass of NSFormatter, which has a wide range of specialized subclasses:
 - NSNumberFormatter
 - NSDateFormatter
 - NSByteCountFormatter
 - NSDateComponentsFormatter
 - NSDateIntervalFormatter
 - NSEnergyFormatter
 - NSMassFormatter
 - NSLengthFormatter
 - MKDistanceFormatter

NSDateFormatter

- You use an NSDateFormatter to get text representations of both dates and times
- You can set the NSDateFormatter's style to set the desired style of date:

1. *NSDateFormatterNoStyle*: No style.
2. *NSDateFormatterShortStyle*: 12/18/13
3. *NSDateFormatterMediumStyle*: Dec 18, 2013
4. *NSDateFormatterLongStyle*: December 18, 2013
5. *NSDateFormatterFullStyle*: Wednesday, December 18, 2013

*sourced from <http://gtiapps.com/?p=1086>

Custom style

- If none of the pre-baked styles fit your needs, you can create a custom date format using the `setDateFormat:` method.
- This takes in a format string that follows a specific pattern:

- *yyyy*: Year using four digits, e.g. 2013
- *yy*: Year using two digits, e.g. '13
- *MM*: Month using two digits, e.g. 05
- *MMMM*: Month, full name, e.g. March
- *dd*: Day with two digits, e.g. 14
- *EEEE*: Day of the week, full name, e.g. Friday
- *EEE*: Day of the week, short, e.g. Mon

For example, the next format string:

`EEEE, dd MMM yyyy`

represents the following formatted date (when this quick tutorial was written):

Thursday, 07 March 2013

Demo

Advanced Fetching*

*Sourced from NSHipster

More Fetching

- Yesterday we learned what core data is, how to use it, how to insert objects, and how to do a very simple fetch.
- Our fetch just retrieved all the objects for a certain Entity.
- Thats great, but what if we wanted to get more specific with our query?
- Sometimes we don't necessarily always want ALL the objects from a certain entity, that could literally be millions of objects!
- We can configure our fetches to be much more specific. Lets take a look.

When to Fetch?

- Sometimes you don't need to do a fetch, you already have the data you need.
- Relationships are a much more efficient way to traverse your object graph vs fetch requests.
- You typically only need to fetch if
 - a) you haven't done a fetch yet
 - b) you need to search your object graph for specific items that match specific criteria

Creating Fetch Requests

- There's actually 4 different ways for fetch requests to be created:
 1. Using its plain initializer
 2. Using its initializer that takes in an entity name that you want to fetch against
 3. Accessing a fetch request that we pre-made in the MOM file (cool)
 4. Same as 3, but there is a method you can use to pass in extra variables for the fetch to be used in the predicate

Creating Fetch Requests in the MOM file

- Creating a pre-defined fetch request in the MOM is safe & easy!
- Its usually best to do this if you find yourself coding the same fetch over and over again.
- Pretty straight forward process

Demo

NSPredicate

- Query language describing how data should be fetched and filtered
- Describes logic conditions to use when searching collections
- Used in Core Data, but can also be used to filter regular foundation classes like arrays and sets.
- Familiar with SQL? Think of the WHERE clause

NSPredicate String syntax

- When creating a predicate, you use a predicate string to communicate to the database what exactly you are looking for.
- The predicate string parser is whitespace insensitive, case insensitive, and supports nested parenthetical expressions.
- The parser does not do any semantic type checking, so typos will cause run time errors.

```
NSString *lastNameSearchString = @"Wilson";
```

```
[NSPredicate predicateWithFormat:@"lastName like %@", lastNameSearchString];
```

This evaluates to fetching all person entities with the attribute lastName like "Wilson"

NSPredicate

- Use predicateWithFormat for building simple predicates
- You can use %@ to substitute for an object value (usually a string)
- You can use %K to substitute for an attribute or attributes (via keypath)

```
[NSPredicate predicateWithFormat:@"name = 'Clarus'"];  
[NSPredicate predicateWithFormat:@"species = %@", @"Dogcow"];  
[NSPredicate predicateWithFormat:@"%K like %@", attributeName, attributeValue];
```

(In the example above, if you had done “%@ like %@”, the query would be incorrect because %@ substitutes always add quotes to the string value. You never want quotes around the attribute you are querying against.)

- Predicates will traverse key paths in a query, which is super awesome

```
[NSPredicate predicateWithFormat:@"department.name like %@", department];  
[NSPredicate predicateWithFormat:@"ANY employees.salary > %f", salary];
```

%K vs %@

- Use %K if you are using a variable to substitute the name of the attribute you are fetching on
- Use %@ if you are using a variable to substitute a value you are using in your query

```
NSString *attributeName = @"lastName";  
NSString *attributeValue = @"Brad";
```

```
[NSPredicate predicateWithFormat:@"%K like %@", attributeName, attributeValue];  
produces @"lastName like "Brad"" ← Correct
```

```
[NSPredicate predicateWithFormat:@"%@" like %@", attributeName, attributeValue];  
produces @"lastName like "Brad"" ← Incorrect!!!
```


NSPredicate Comparisons

- `=`, `==`: the expression on the left is equal to the right-hand expression
- `>=`, `=>`: the left hand expression is greater than or equal to the right hand expression (and vice versa for `<=`, `=<`, and also `<`, `>`)
- `!=`, `<>`: the left hand expression is not equal to the right hand expression
- `BETWEEN`: the left hand expression is between, or equal to either of, the values specified in the right hand side. The right hand side is a two value array giving upper and lower bounds. Arrays denoted with `{}`

NSPredicate String Comparisons

- **BEGINSWITH:** the left-hand expression begins with the right-hand expression
- **CONTAINS:** the left-hand expression is contained in the right-hand expression
- **ENDSWITH:** the left-hand expression ends with the right-hand expression
- **LIKE:** the left-hand expression equals the right hand expression: ? and * are allowed as wildcard characters, where ? matches 1 character and * matches 0 or more characters
- **MATCHES:** the left-hand expression equals the right-hand expression using regex-style comparisons.

NSPredicate Relational Operators

- ANY, SOME: Specifies any of the elements in the following expression
- ALL: Specifies all of the elements in the following expression
- NONE: Specifies none of the elements in the following expression
- IN: Equivalent to an SQL IN operation, the left-hand side just appear in the collection specified by the right-hand side.

NSPredicate Basic Compound Predicates

- AND, &&: Logical AND
- OR, ||: Logical OR
- NOT, !: Logical NOT

```
[NSPredicate predicateWithFormat: @"room.hotel.name == %@ AND  
dateFrom <= %@ AND dateTo >= %@", @"Solid Stay",  
self.startPicker.date, self.endPicker.date];
```

NSPredicate and self

- You will see the word self used in NSPredicates
- self refers to each entity we are querying against

```
[NSPredicate predicateWithFormat:@"self == %@", someObject];
```

Demo

NSPredicate variable substitution

- Variable substitution is supported too, using \$ sign for variable names

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"id = $user"];  
NSDictionary *substitutions = @{@"user": currentUser}
```

```
NSPredicate *filter = [predicate  
predicateWithSubstitutionVariables:substitutions];
```

Searching Collections

- Collection classes can be searched and filtered using `NSPredicate`*
 - `NSFetchRequest` exposes a predicate property, for filtering entities
-
- `(NSArray *)filteredArrayUsingPredicate:(NSPredicate *)predicate;`
 - `(NSSet *)filteredSetUsingPredicate:(NSPredicate *)predicate;`

*Most collection classes; `NSDictionary` and `NSIndexSet` don't directly accept an `NSPredicate`

NSFetchRequest resultType

- NSFetchRequest has a property named resultType.
- By default it is set to NSManagedObjectResultType
- Here is all the types:
 - NSManagedObjectResultType: returns the managed objects
 - NSCountResultType: returns the count of all the objects that match the fetch
 - NSDictionaryResultType: This returns the results back in a dictionary
 - NSManagedObjectIDResultType: Returns the ObjectIDs instead of full-fledged managed objects

Fetching the count

- NSCountResultType lets you get back the count of objects that a fetch will retrieve.
- It returns a an array containing an NSNumber, which is the count.
- Its a little weird, but it works!

Demo