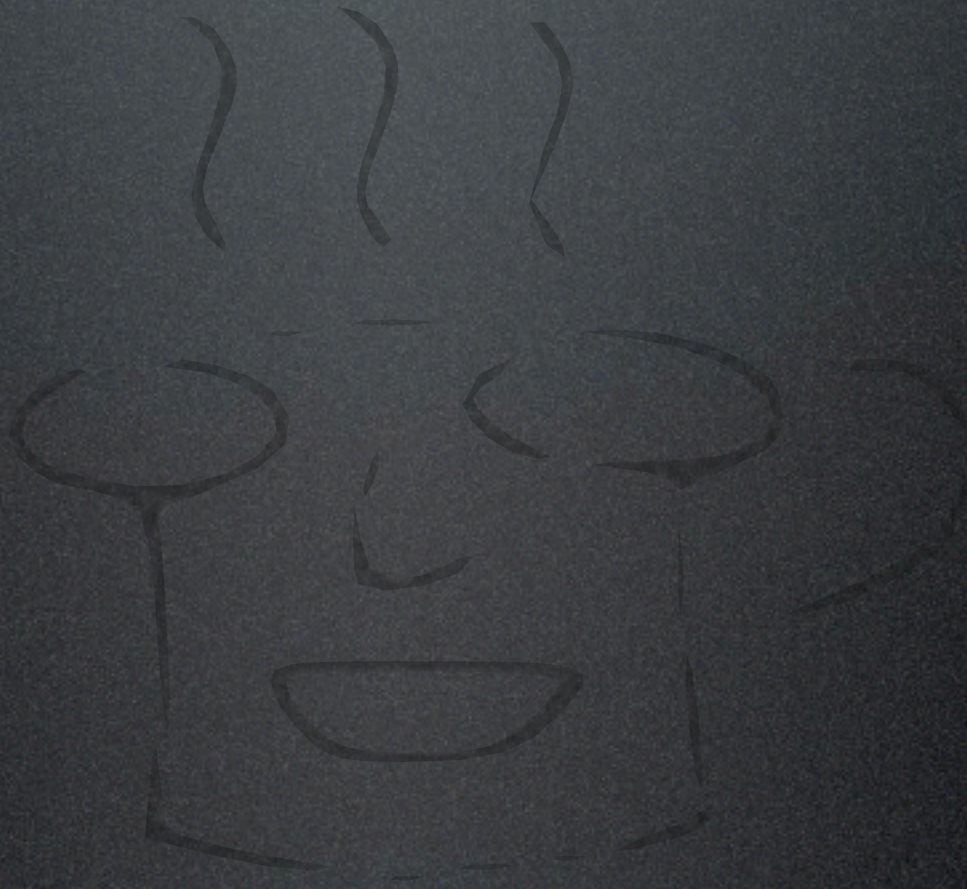


KVC/KVO and Bindings

"MVC with less code"



CocoaHeads México
27-Nov-2008

Key Value Coding

What is it?

- Access properties by name:
- Every class essentially becomes a dictionary
- **Not** a 'trick'. This is **fundamental** Cocoa programming.

What is it?

- Access properties by name:

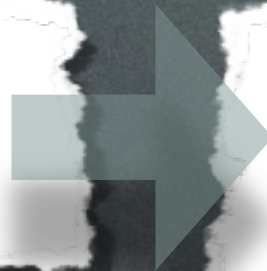
```
[person name]  
[person setName:@"Sergio"]
```

- Every class essentially becomes a dictionary
- **Not** a 'trick'. This is **fundamental** Cocoa programming.

What is it?

- Access properties by name:

```
[person name]  
[person setName:@"Sergio"]
```



```
[person valueForKey:@"name"]  
[person setValue:@"Sergio"  
forKey:@"name"]
```

- Every class essentially becomes a dictionary
- **Not** a 'trick'. This is **fundamental** Cocoa programming.

How it works

- **NSKeyValueCoding**
(informal) protocol
- Implemented by **NSObject**
- NSKeyValueCoding.h

Getting values

- valueForKey: ⓘ
- valueForKeyPath: ⓘ
- dictionaryWithValuesForKeys: ⓘ
- valueForKeyUndefinedKey: ⓘ
- mutableArrayValueForKey: ⓘ
- mutableArrayValueForKeyPath: ⓘ
- mutableSetValueForKey: ⓘ
- mutableSetValueForKeyPath: ⓘ

Setting Values

- setValue:forKeyPath: ⓘ
- setValuesForKeysWithDictionary: ⓘ
- setNilValueForKey: ⓘ
- setValue:forKey: ⓘ
- setValue:forUndefinedKey: ⓘ

Changing Default Behavior

- + accessInstanceVariablesDirectly ⓘ

Validation

- validateValue:forKey:error: ⓘ
- validateValue:forKeyPath:error: ⓘ

Using key value coding

Code!

Using key value coding

Accessor lookup

Accessor lookup

1. Search for a public accessor
(**'getLastName'** or **'lastName'**)

Accessor lookup

1. Search for a public accessor (**'getLastName'** or **'lastName'**)
2. If not found, search for a private accessor method based on key (**'_getLastName'** or **'_lastName'**).

Accessor lookup

1. Search for a public accessor (**'getLastName'** or **'lastName'**)
2. If not found, search for a private accessor method based on key (**'_getLastName'** or **'_lastName'**).
3. If not found' search for an instance variable (**'_lastName'** or **'lastName'**)

Accessor lookup

1. Search for a public accessor (**'getLastName'** or **'lastName'**)
2. If not found, search for a private accessor method based on key (**'_getLastName'** or **'_lastName'**).
3. If not found, search for an instance variable (**'_lastName'** or **'lastName'**)
4. If not found, invoke **valueForUndefinedKey:**

No magic!

- KVC uses the runtime information generated by the compiler
- Easy to fake...

"Implementing" KVC "from scratch"

Code!

"Implementing" KVC "from scratch"

Performance?

- Every Objective-C message goes through `objc_msgSend()` anyways
- KVC caches property lookups
- Not likely to be the bottleneck

KVC, value types and nil

- Property values are always of type **id** (reference type)
 - structs → **NSValue**
 - numbers → **NSNumber**
- You should implement **setNilValueForKey:**

NSDictionary KVC

- **NSDictionary** has a custom implementation of KVC that attempts to match keys against keys in the dictionary.

```
NSDictionary *dictionary;  
[dictionary setValue:@"Sergio" forKey:@"name"];
```


Practical uses of KVC

- Used by UI bindings
- Key-based archiving (serialization)

More on KVC

Model Validation

- Optional set of validation methods
- Some views call automatically the validation methods
 - **Important!**: You are not supposed to reject the value in **setValueForKey:**
 - **validateValueForKey:error:** automatically calls:
validate{property}:error:

Key Paths

- Like file paths (/usr/local/bin)
- Used to traverse an object graph:
"document.header.title"
- **valueForKeyPath:**,
setValue:forKeyPath:
- Warning!: you cannot pass a keyPath if the selector is just '**forKey:**' only to '**forKeyPath:**'

Using key paths

Code!

Using key paths

KVC and Array properties

- No native support. You cannot pass indexes in key paths:
 - ~~valueForKeyPath:@"order.items[1].price"~~
- Ask an array for a key, the array will ask all its elements and return an array of the resulting values.

Set and array operators

- Arrays and sets support 'aggregate' keys:
 - `@avg`, `@count`, `@max`, `@min`, `@sum`,
`@distinctUnionOfArrays`,
`@distinctUnionOfObjects`, `@distinctUnionOfSets`,
`@unionOfArrays`, `@unionOfObjects`, `@unionOfSets`
- You can use these aggregate properties to bind to in IB

```
NSNumber *total = [purchaseOrder valueForKeyPath: @"@sum.price"];
```


NSPredicates

- You can build **NSPredicates** to use more complex queries, like And, Or, RegExps, method calling, etc.
- Cocoa's 'LINQ'

```
NSArray *severalObjects = [NSArray arrayWithObjects:@"Stig",  
@"Shaffiq", @"Chris", nil];
```

```
NSPredicate *predicate = [NSPredicate  
predicateWithFormat:@"SELF IN %@", severalObjects];
```

```
BOOL result = [predicate evaluateWithObject:@"Shaffiq"];
```


The secret to KVC compliance...

The secret to KVC compliance...

Just use the correct naming
conventions for your accessors!
(or Objective-C 2.0 properties)

Key Value Observing

What is it?

- Change notifications to model classes
- Observer pattern
- Based on KVC

How it works

- Subscribe:
 - **addObserver:forKeyPath:options:context:**
 - options: **NSKeyValueObservingOptionOld**
NSKeyValueObservingOptionNew
- Unsubscribe:
 - **removeObserver:forKeyPath:**
- Receive notification:
 - **observeValue:forKeyPath:**
ofObject:change:context:

Important!

- Notification works by replacing your accessors.
- This means that direct ivar access will not produce notifications!

Observing with KVO notifications

Code!

Observing with KVO notifications

Derived properties

- **Example:** `fullName` property that changes when `lastName` and `firstName` change
- Implement:
`keyPathsForValuesAffectingValueForKey:`,
and return a set of all the properties that affect that `keyPath`.

Manual notifications

- **Example:** dynamic value that changes a lot. Calculated value you only want to display at certain times
- **willChangeValueForKey:**,
didChangeValueForKey:,
automaticallyNotifyObserversForKey:

Manual notifications and derived
properties

Code!

Manual notifications and derived
properties

Careful!

- **addObserver:...** does not throw an exception when you get the keyPath wrong.
- If you observe a path with multiple stages, you should observe at every level in the path:
 - **“document.header.title”**

Observing Array changes

- **Difficult.** You can't observe them at all
 - (in KVO, only the property is observed, not the value)
- **Option 1: `mutableArrayValueForKey:`**
 - Returns a proxy (expensive) that sends notifications when modified
- **Option 2:** Implement all the array-related accessors

KVC Accessor selector formats

`{property}`

`set{property}:`

`_set{property}:`

`{property}ForKeyPath:`

`_ {property}ForKeyPath:`

`get{property}`

`_get{property}`

`is{property}`

`_is{property}`

`validate{property}:error:`

`countOf{property}`

`objectIn{property}AtIndex:`

`{property}AtIndexes:`

`get{property}:range:`

`enumeratorOf{property}`

`memberOf{property}:`

`intersect{property}:`

`insertObject:in{property}AtIndex:`

`insert{property}:atIndexes:`

`removeObjectFrom{property}AtIndex:`

`remove{property}AtIndexes:`

`replaceObjectIn{property}AtIndex:withObject:`

`replace{property}AtIndexes:with{property}:`

`add{property}Object:`

`remove{property}:`

`remove{property}Object:`


`add{property}:`

`getPrimitive{property}`

`primitive{property}`

`setPrimitive{property}:`

Observe array element changes

- You can't. You have to observe each of the elements individually
- Solution: **NSArrayController** 
 - Bind the '**contents**' property to the array of interest and observe the path
arrangedObjects.{propertyName}
 - For insertions and deletions, observe the **arrangedObjects** property directly.

Uses for KVO?

- Coherence between multiple views and a model
- Simplifying flow control
- Simplifying undo implementation
- Any kind of observer, not only UI views.
- Binding controllers: **NSController** and **NSArrayController** are based on KVO

Bindings

What is it?

“Allows you to establish a mediated connection between a view and a piece of data, “binding” them such that a change in one is reflected in the other”

- Cocoa Bindings Programming Topics guide

- Not restricted to 'views' and 'models'.
General binding mechanism.

Manual binding

- Call:
bind:toObject:withKeyPath:options:
on the 'view' object
- The 'view' will use KVC/KVO
 - KVC to load the initial values
 - Receives notifications via KVO
 - Changes the 'model' with KVC





Manual bindings

Code!

Manual bindings

Binding Controllers



-  Basic binding works without controllers!
-  **Object controller:** dumb wrapper around KVO
-  **Array Controller:** **selection** proxy for master-detail UI, and observing of arrays.
-  **User Defaults Controller:** Simulates KVC/KVO compliance!

Bindings in IB

Code!

Bindings in IB

(if time permits)

One more thing...

Value Transformers

(if time permits)

Goal: Reduce 'glue' code

Goal: Reduce 'glue' code

- **Example:** binding an 'enabled' control to a 'false' property

Goal: Reduce 'glue' code

- **Example:** binding an 'enabled' control to a 'false' property
- **Example:** showing an image for a string property

Goal: Reduce 'glue' code

- **Example:** binding an 'enabled' control to a 'false' property
- **Example:** showing an image for a string property
- **Example:** handling null values on the model

Available value transformers:

- **NSNegateBooleanTransformer**
- **NSIsNilTransformer**
- **NSIsNotNilTransformer**
- **NSUnarchiveFromDataTransformer**
- **NSKeyedUnarchiveFromDataTransformer**
- Custom (write your own transformer!)

Value Transformers

Code!

Value Transformers

Resources

- [Introduction to Cocoa Bindings Programming Topics \(Apple\)](#)
- [Key Value Coding programming guide \(Apple\)](#)
- [Key Value Observing programming guide \(Apple\)](#)
- [Model Object Implementation Guide \(Apple\)](#)
- [Late Night Cocoa episode 35: KVC/KVO with Stefen Frey](#)
- [Introduction to Cocoa Bindings by Scott Stevenson](#)
- [Cocoa Bindings Wiki page at Cocoadev.com](#)

The End