

Effective Core Data with Swift

try! Swift NYC 2018

Tom Harrington, tph@atomicbird.com

@atomicbird on a social network near you

Core Data vs. Swift: Fight!

What we'll cover

- Core Data vs. Swift
- What Xcode can and can't do for you
- Core Data WTFs when using Swift
- Better Core Data with Swift

Not an intro to Core Data

Core Data vs. Swift

Timeline (abridged):

- 2005: Core Data for Mac OS X 10.4
- 2009: Core Data for iOS 3.0
- 2014: Swift 1.0
- 2018: Swift 4.2

**Core Data
still shows
Objective-C roots.**

How does this show up?

Data model

- Can only store model **objects**
- Model objects must inherit from **NSManagedObject**

How does this show up?

Properties

How does this show up?

Properties

- Core Data properties must be representable in Objective-C.

How does this show up?

Properties

- Core Data properties must be representable in Objective-C.
- No pure Swift types (no **struct**, **enum**)

How does this show up?

Properties

- Core Data properties must be representable in Objective-C.
- No pure Swift types (no **struct**, **enum**)
- Objects must inherit from **NSObject**.

How does this show up?

Properties

- Core Data properties must be representable in Objective-C.
- No pure Swift types (no **struct**, **enum**)
- Objects must inherit from **NSObject**.
- Optionals can be tricky

How does this show up?

Properties

- Core Data properties must be representable in Objective-C.
- No pure Swift types (no **struct**, **enum**)
- Objects must inherit from **NSObject**.
- Optionals can be tricky
- Support for **NSCoding** but not **Codable**

How does this show up?

Relationships

- To-many relationships are untyped **NSSets**
- Need weird add/remove methods to modify to-many relationships

Xcode, Swift, and Core Data

ENTITIES

E

 Event

E

 Ticket

E

 Venue

FETCH REQUESTS

CONFIGURATIONS

C

 Default

▼ Attributes

Attribute	^	Type	
<div>S</div> name		String	↕
<div>D</div> timestamp		Date	↕

+ —

▼ Relationships

Relationship	^	Destination	Inverse
<div>M</div> tickets		Ticket	↕ event ↕
<div>O</div> venue		Venue	↕ events ↕

+ —

Xcode code generation

Entity

Name

☐ Abstract Entity

Parent Entity

Class

Name

Module

Codegen ☒ Class Definition ☐ Category/Extension

Constraints

No Content

+ -

Generated code

```
extension Event {  
  
    @nonobjc public class func fetchRequest() -> NSFetchRequest<Event> {  
        return NSFetchRequest<Event>(entityName: "Event")  
    }  
  
    @NSManaged public var name: String?  
    @NSManaged public var timestamp: Date?  
    @NSManaged public var tickets: NSSet?  
    @NSManaged public var venue: Venue?  
  
}
```

With Xcode's generated code...

- Add more extensions to add custom code
- You need to `NSSet <--> Set<Ticket>` a lot
- Scalar types can't be optional
- Can't easily inspect code

Core Data WTFs: Optionals #1

Swift has optionals!

Core Data has optionals!

But remember the timeline...

Attribute

Name

Properties ☐ Transient ☐ Optional

Attribute Type

Validation ☐ Min Length

☐ Max Length

Default Value

Reg. Ex.

@NSManaged public var name: String?

Core Data WTFs: Optionals #1

(Non-optionals, really)

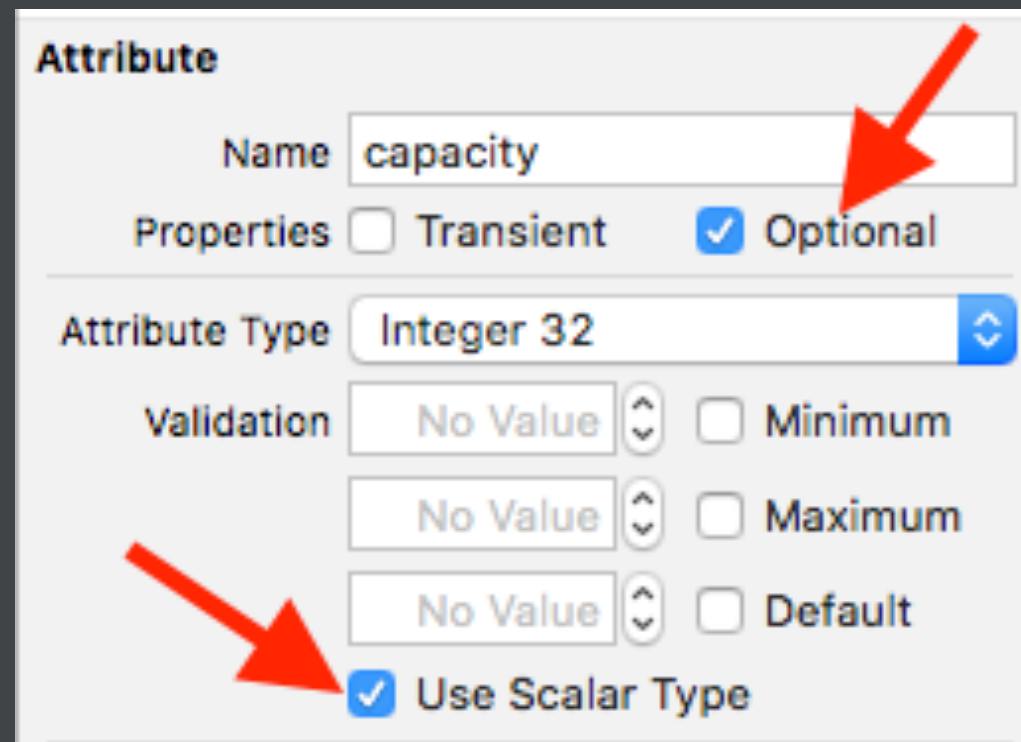
- Core Data: Must not be nil **when you save changes**
- Swift: Must not be nil **at any time**

So...

- Xcode generates Swift properties as optionals

Core Data WTFs: Optionals #2

But wait, there's more!



The screenshot shows the 'Attribute' inspector in Xcode. The 'Name' field is 'capacity'. Under 'Properties', the 'Optional' checkbox is checked, highlighted by a red arrow. The 'Attribute Type' is 'Integer 32'. Under 'Validation', there are three rows, each with a 'No Value' field and a checkbox for 'Minimum', 'Maximum', and 'Default'. At the bottom, the 'Use Scalar Type' checkbox is checked, also highlighted by a red arrow.

```
@NSManaged public var capacity: Int32
```

Core Data WTFs: Optionals #2

- *Core Data properties must be representable in Objective-C.*
- Objective-C doesn't have optional scalars.
- **These properties have default values** even if you don't configure a default.
 - `0` for numeric types
 - `false` for boolean

The screenshot shows the 'Attribute' inspector in Xcode. The 'Name' field is set to 'capacity'. Under 'Properties', the 'Optional' checkbox is checked. The 'Attribute Type' is set to 'Integer 32'. Under 'Validation', there are three rows, each with a 'No Value' field and a checkbox for 'Minimum', 'Maximum', and 'Default'. The 'Use Scalar Type' checkbox at the bottom is checked. Two red arrows point to the 'Optional' checkbox and the 'Use Scalar Type' checkbox.

Attribute	
Name	capacity
Properties	<input type="checkbox"/> Transient <input checked="" type="checkbox"/> Optional
Attribute Type	Integer 32
Validation	<input type="checkbox"/> Minimum
	<input type="checkbox"/> Maximum
	<input type="checkbox"/> Default
	<input checked="" type="checkbox"/> Use Scalar Type

Can we do better?

(spoiler warning)

Yes we can.

But first a little background...

NSManagedObject.self

- You start by creating **Entities**
- **NSManagedObject** instances represent an entity
- Properties are declared by the entity, not **NSManagedObject**

NSManagedObject.self

ENTITIES	
<div>E</div> Event	
<div>E</div> Ticket	
<div>E</div> Venue	
FETCH REQUESTS	
CONFIGURATIONS	
<div>C</div> Default	

Attributes	
Attribute	Type
<div>S</div> name	String
<div>D</div> timestamp	Date
+ -	

Relationships	
Relationship	Destination
<div>M</div> tickets	Ticket
<div>O</div> venue	Venue
+ -	

NSManagedObject.self

Subclassing is *recommended* but not required.

```
let myEvent: NSManagedObject =  
    NSEntityDescription.insertNewObject(forEntityName: "Event",  
                                        into: context)
```

```
myEvent.setValue("try! Swift", forKey: "name")
```

```
let name = myEvent.value(forKey: "name") as? String
```

NSManagedObject.self

Subclassing is *recommended* but not required.

```
let myEvent: NSManagedObject =  
    NSEntityDescription.insertNewObject(forEntityName: "Event",  
        into: context)
```

```
myEvent.setValue("try! Swift", forKey: "name")
```

```
let name = myEvent.value(forKey: "name") as? String
```

NSManagedObject.self

Subclassing is *recommended* but not required.

```
let myEvent: NSManagedObject =  
    NSEntityDescription.insertNewObject(forEntityName: "Event",  
        into: context)
```

```
myEvent.setValue("try! Swift", forKey: "name")
```

```
let name = myEvent.value(forKey: "name") as? String
```

NSManagedObject.self

Subclassing is *recommended* but not required.

```
let myEvent: NSManagedObject =  
    NSEntityDescription.insertNewObject(forEntityName: "Event",  
        into: context)  
  
myEvent.setValue("try! Swift", forKey: "name")  
  
let name = myEvent.value(forKey: "name") as? String
```

NSManagedObject.self

Subclassing with Xcode

```
extension Event {  
    @NSManaged public var capacity: Int32  
    @NSManaged public var name: String?  
    @NSManaged public var timestamp: Date?  
    @NSManaged public var tickets: NSSet?  
    @NSManaged public var venue: Venue?  
}
```


NSManagedObject.self

Subclassing with Xcode

```
extension Event {  
    @NSManaged public var capacity: Int32  
    @NSManaged public var name: String?  
    @NSManaged public var timestamp: Date?  
    @NSManaged public var tickets: NSSet?  
    @NSManaged public var venue: Venue?  
}
```

NSManagedObject.self

Subclassing with Xcode

```
extension Event {  
    @NSManaged public var capacity: Int32  
    @NSManaged public var name: String?  
    @NSManaged public var timestamp: Date?  
    @NSManaged public var tickets: NSSet?  
    @NSManaged public var venue: Venue?  
}
```

```
@NSManaged public var name: String?
```

@NSManaged **public var** name: **String?**

- @NSManaged is not a stored property

@NSManaged **public var** name: **String**?

- @NSManaged is not a stored property
- Computed property? But no **set**, **get**?

@NSManaged **public var** name: **String**?

- @NSManaged is not a stored property
- Computed property? But no **set**, **get**?
- **NSManagedObject** dynamically creates the accessors

@NSManaged **public var** name: **String**?

- @NSManaged is not a stored property
- Computed property? But no **set**, **get**?
- **NSManagedObject** dynamically creates the accessors
- @NSManaged declaration must match the entity



@NSManaged is not required

```
@NSManaged public var name: String?
```

```
public var name: String? {  
    set {  
        willChangeValue(forKey: "name")  
        defer { didChangeValue(forKey: "name") }  
        setPrimitiveValue(newValue, forKey: "name")  
    }  
    get {  
        willAccessValue(forKey: "name")  
        defer { didAccessValue(forKey: "name") }  
        return primitiveValue(forKey: "name") as? String  
    }  
}
```




@NSManaged is not required

```
@NSManaged public var name: String?
```

```
public var name: String? {  
    set {  
        willChangeValue(forKey: "name")  
        defer { didChangeValue(forKey: "name") }  
        setPrimitiveValue(newValue, forKey: "name")  
    }  
    get {  
        willAccessValue(forKey: "name")  
        defer { didAccessValue(forKey: "name") }  
        return primitiveValue(forKey: "name") as? String  
    }  
}
```

NSManagedObject.self

- *Core Data properties must be representable in Objective-C.*

Except...

- This applies to the *entity*, not necessarily to the class.
- Custom accessors can convert to/from Swift types.

```
enum EventType: Int {  
    case conference  
    case party  
    case meeting  
    case concert  
}
```

```
@NSManaged public var eventType: EventType?
```

```
@NSManaged public var eventType: Int16
```

```
enum EventType: Int {  
    case conference  
    case party  
    case meeting  
    case concert  
}
```

```
@NSManaged public var eventType: EventType?
```

```
@NSManaged public var eventType: Int16
```

```
enum EventType: Int {  
    case conference  
    case party  
    case meeting  
    case concert  
}
```

```
@NSManaged public var eventType: EventType?
```

```
@NSManaged public var eventType: Int16
```

However...

```
public var eventType: EventType? {
    set {
        willChangeValue(forKey: "eventType")
        defer { didChangeValue(forKey: "eventType") }
        setPrimitiveValue(newValue?.rawValue, forKey: "eventType")
    }
    get {
        willAccessValue(forKey: "eventType")
        defer { didAccessValue(forKey: "eventType") }
        guard let rawValue = primitiveValue(forKey: "eventType") as? Int else {
            return nil
        }
        return EventType(rawValue: rawValue)
    }
}
```

However...

```
public var eventType: EventType? {
    set {
        willChangeValue(forKey: "eventType")
        defer { didChangeValue(forKey: "eventType") }
        setPrimitiveValue(newValue?.rawValue, forKey: "eventType")
    }
    get {
        willAccessValue(forKey: "eventType")
        defer { didAccessValue(forKey: "eventType") }
        guard let rawValue = primitiveValue(forKey: "eventType") as? Int else {
            return nil
        }
        return EventType(rawValue: rawValue)
    }
}
```

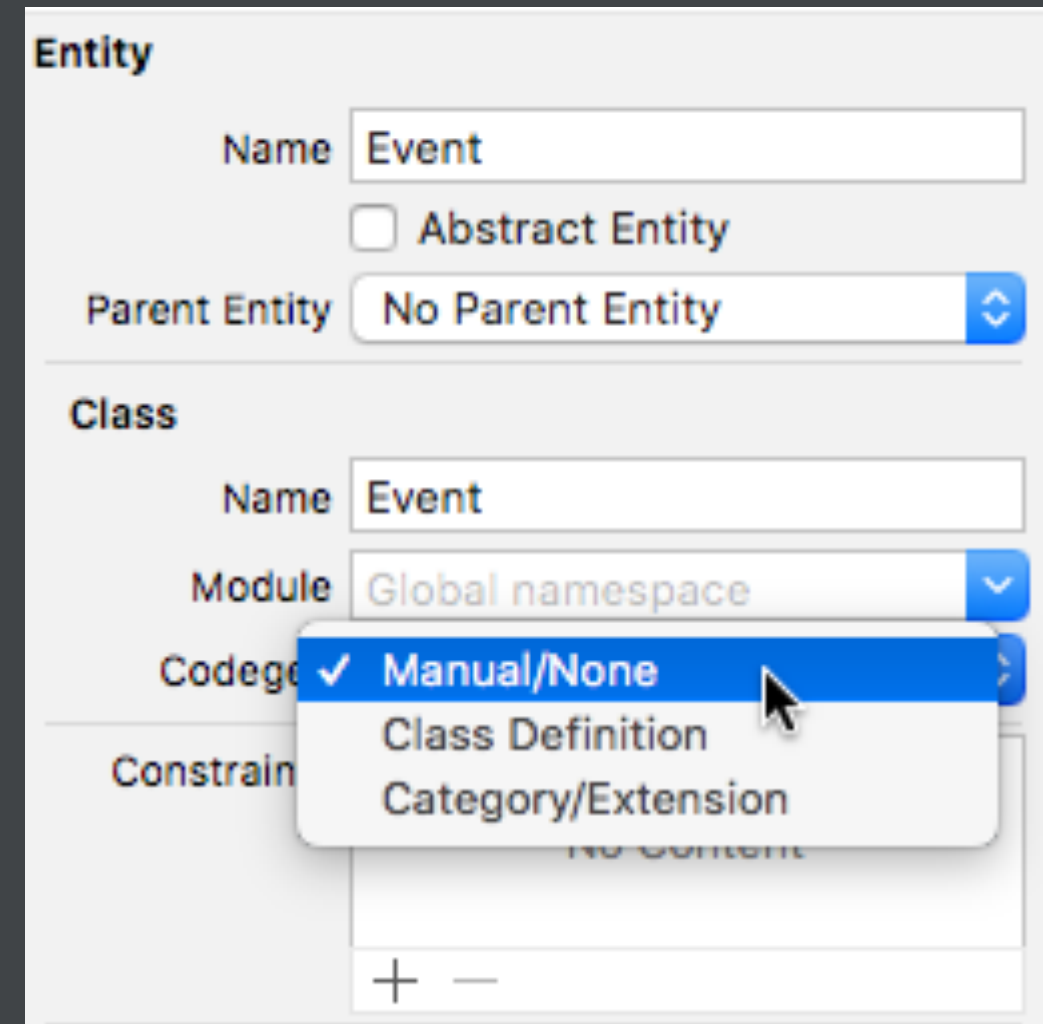
However...

```
public var eventType: EventType? {
    set {
        willChangeValue(forKey: "eventType")
        defer { didChangeValue(forKey: "eventType") }
        setPrimitiveValue(newValue?.rawValue, forKey: "eventType")
    }
    get {
        willAccessValue(forKey: "eventType")
        defer { didAccessValue(forKey: "eventType") }
        guard let rawValue = primitiveValue(forKey: "eventType") as? Int else {
            return nil
        }
        return EventType(rawValue: rawValue)
    }
}
```


Xcode will fight you on this

- Generated code lives in **DerivedData**
- Will be re-generated and overwritten
- Can't add it to your code repo
- So turn off Xcode code generation?

But then what?



**Generate code
without Xcode**



mogenerator generates Objective-C code for your [Core Data](#) custom classes

Unlike Xcode, mogenerator manages *two* classes per entity: one for **machines**, one for **humans**

The machine class can always be overwritten to match the data model, with humans' work effortlessly preserved



[Download mogenerator 1.31](#)

or install via [homebrew](#):

```
$ brew install mogenerator
```

upgrading using [homebrew](#):

```
$ brew update && brew upgrade mogenerator
```

Managed Object Generator

The screenshot shows the GitHub repository page for `rentzsch/mogenerator`. The repository has 112 watchers, 2,984 stars, and 403 forks. It contains 647 commits, 4 branches, 14 releases, and 67 contributors. The license is MIT. The repository is currently on the `master` branch. A recent commit by `atomicbird` is highlighted, titled "Add entity level 'additionalImports', a comma-separated list of modul...". Below the commit list, there is a table of files and folders, including `.github`, `MiscMerge`, `categories`, `contributed templates`, `ddcli`, `installer`, `mogenerator.xcodeproj`, `momcom`, `ponso`, `templates`, `test`, `.gitignore`, `.travis.yml`, `CONTRIBUTING.md`, `LICENSE`, and `README.md`.

Core Data code generation <http://rentzsch.github.com/mogenerator>

647 commits 4 branches 14 releases 67 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

atomicbird Add entity level "additionalImports", a comma-separated list of modul... Latest commit aaea23b 5 days ago

.github	Added pull request & issue templates.	2 years ago
MiscMerge	Don't check for failure by looking at the NSError	2 years ago
categories	Replaced RegexKitLite with NSRegularExpression.	3 years ago
contributed templates	Reverted PonsoTest to version on rentzsch/master	7 years ago
ddcli	instancetype support.	3 years ago
installer	Bump version to 1.29, copyright to 2015.	3 years ago
mogenerator.xcodeproj	Remove user-specific command line args	9 days ago
momcom	Support "Custom Class" for Transformable attributes:	9 days ago
ponso	instancetype support.	3 years ago
templates	Add entity level "additionalImports", a comma-separated list of modul...	5 days ago
test	Specify path to 10.13 SDK	11 months ago
.gitignore	check-in `mogenerator` shared scheme	3 years ago
.travis.yml	Update travis config to specify Xcode 9	11 months ago
CONTRIBUTING.md	[TYPO] Create-an-issue link	2 years ago
LICENSE	Add MIT LICENSE file to make it clear templates are under the same li...	5 years ago
README.md	Updated version history in README	11 days ago

Timeline (abridged) (revised):

- 2005: Core Data for Mac OS X 10.4
- **2006: mogenerator**
- 2009: Core Data for iOS 3.0
- 2014: Swift 1.0
- 2018: Swift 4.2

mogenerator

- Command line tool
- Reads data model
- Generates `NSManagedObject` subclasses using templates
- Better Swift

*Note: The following slides refer to **mogenerator swift42** branch. Release coming very soon.*

Attribute

Name

Properties ☐ Transient ☒ Optional

Attribute Type

Validation ☐ Minimum

☐ Maximum

☒ Default

☒ Use Scalar Type

Advanced ☐ Index in Spotlight

☐ Preserve After Deletion

Deprecated

Spotlight ☐ Store in External Record File

User Info

Key	Value
attributeValueScalarType	EventType

mogenerator

SWIFT TYPED RELATIONSHIPS 😊

```
@NSManaged public var tickets: NSSet?
```

```
@NSManaged public var tickets: Set<Ticket>?
```

mogenerator

SWIFT TYPED RELATIONSHIPS 😊

```
@NSManaged public var tickets: NSSet?
```

```
@NSManaged public var tickets: Set<Ticket>?
```


Optional scalars: They just work

```
public var capacity: Int32? {  
    get {  
        let key = Event.Attributes.capacity  
        willAccessValue(forKey: key)  
        defer { didAccessValue(forKey: key) }  
  
        return primitiveValue(forKey: key) as? Int32  
    }  
    set {  
        let key = Event.Attributes.capacity  
        willChangeValue(forKey: key)  
        defer { didChangeValue(forKey: key) }  
  
        guard let value = newValue else {  
            setPrimitiveValue(nil, forKey: key)  
            return  
        }  
        setPrimitiveValue(value, forKey: key)  
    }  
}
```

Optional scalars: They just work

```
public var capacity: Int32? {  
    get {  
        let key = Event.Attributes.capacity  
        willAccessValue(forKey: key)  
        defer { didAccessValue(forKey: key) }  
  
        return primitiveValue(forKey: key) as? Int32  
    }  
    set {  
        let key = Event.Attributes.capacity  
        willChangeValue(forKey: key)  
        defer { didChangeValue(forKey: key) }  
  
        guard let value = newValue else {  
            setPrimitiveValue(nil, forKey: key)  
            return  
        }  
        setPrimitiveValue(value, forKey: key)  
    }  
}
```

Optional scalars: They just work

```
public var capacity: Int32? {  
    get {  
        let key = Event.Attributes.capacity  
        willAccessValue(forKey: key)  
        defer { didAccessValue(forKey: key) }  
  
        return primitiveValue(forKey: key) as? Int32  
    }  
    set {  
        let key = Event.Attributes.capacity  
        willChangeValue(forKey: key)  
        defer { didChangeValue(forKey: key) }  
  
        guard let value = newValue else {  
            setPrimitiveValue(nil, forKey: key)  
            return  
        }  
        setPrimitiveValue(value, forKey: key)  
    }  
}
```

mogenerator

Avoid stringly typing:

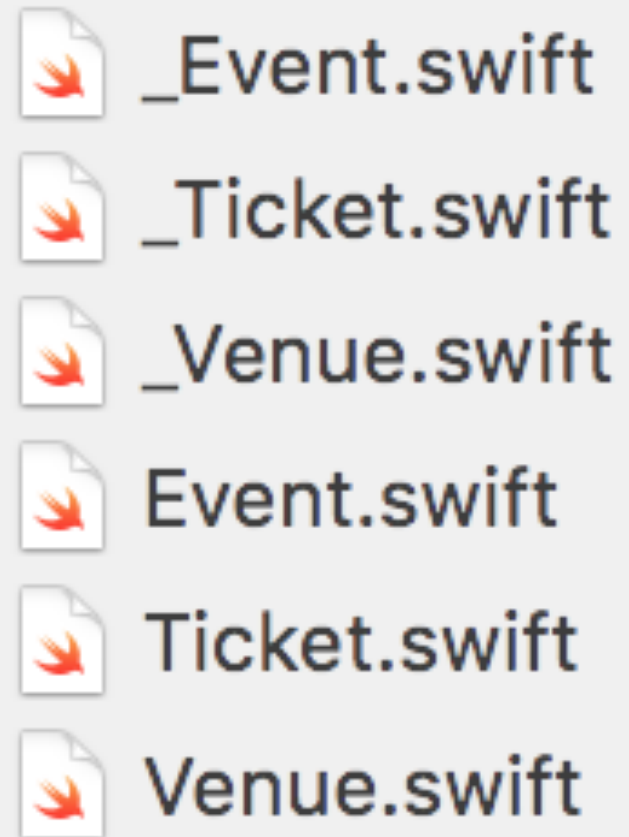
```
public struct Attributes {  
    static let capacity = "capacity"  
    static let eventType = "eventType"  
    static let name = "name"  
    static let timestamp = "timestamp"  
}
```

```
public struct Relationships {  
    static let tickets = "tickets"  
    static let venue = "venue"  
}
```

mogenerator

Simplest case:

```
$ mogenerator --model Model.xcdatamodeld/ --swift  
3 machine files and 3 human files generated.
```



- _Event.swift
- _Ticket.swift
- _Venue.swift
- Event.swift
- Ticket.swift
- Venue.swift

mogenerator

Two files per entity:

- `_EntityName.swift`: All generated code.

```
public extension Event {  
    ...  
}
```

- `EntityName.swift`: Put your code here

```
objc(Event)  
open class Event: NSObject {  
    ...  
}
```

Event.swift

```
import Foundation
```

```
import CoreData
```

```
@objc(Event)
```

```
open class Event: NSManagedObject {
```

```
    // Custom code for Event goes here.
```

```
}
```

_Event.swift

```
// DO NOT EDIT. This file is machine-generated and constantly overwritten.  
// Make changes to Event.swift instead.
```

```
import CoreData
```

```
public extension Event {
```

```
    @objc public class var entityName: String {  
        return "Event"  
    }
```

```
    @objc public class func entity(managedObjectContext: NSManagedObjectContext) -> NSEntityDescription? {  
        return NSEntityDescription.entity(forEntityName: entityName, in: managedObjectContext)  
    }
```

```
    // Attribute names for Event
```

```
    public struct Attributes {  
        static let capacity = "capacity"  
        static let eventType = "eventType"  
        static let eventTypeString = "eventTypeString"  
        static let name = "name"  
        static let organizer = "organizer"  
        static let timestamp = "timestamp"  
    }
```

```
    ....
```


mogenerator

Enumerations with raw values

```
enum EventType: Int {  
    case conference  
    case party  
    case meeting  
    case concert  
}
```

Generated declaration:

```
public var eventType: EventType?
```

Attribute

Name

Properties ☐ Transient ☒ Optional

Attribute Type

Validate ☐ Minimum ☐ Maximum ☒ Default ☒ Use Scalar Type

Advanced ☐ Index in Spotlight ☐ Preserve After Deletion

Deprecated

Spotlight ☐ Store in External Record File

User Info

Key	Value
attributeRawValueEnumType	EventType

```
public var eventType: EventType? {
    get {
        let key = Event.Attributes.eventType
        willAccessValue(forKey: key)
        defer { didAccessValue(forKey: key) }

        guard let primitiveValue = primitiveValue(forKey: key) as? EventType.RawValue
            else { return nil }
        return EventType(rawValue: primitiveValue)
    }
    set {
        let key = Event.Attributes.eventType
        willChangeValue(forKey: key)
        defer { didChangeValue(forKey: key) }

        guard let value = newValue else {
            setPrimitiveValue(nil, forKey: key)
            return
        }
        setPrimitiveValue(value.rawValue, forKey: key)
    }
}
```

mogenerator

Support for `Codable` properties.

```
public struct OrganizerContactInfo: Codable {  
    let name: String  
    let email: String  
    let twitter: String  
}
```

Generated declaration:

```
public var organizer: OrganizerContactInfo?
```

- Uses `JSONEncoder/JSONDecoder` internally.

Attribute

Name

Properties ☐ Transient ☒ Optional

Attribute Type

Options ☐ Allows External Storage

Advanced ☐ Index in Spotlight
☐ Preserve After Deletion

Deprecated
Spotlight ☐ Store in External Record File

User Info

Key	Value
attributeCodableTypeName	OrganizerContactInfo

Coming soon: Codable managed objects

```
public class Event: NSManagedObject: Codable {
    required convenience public init(from decoder: Decoder) throws {
        guard let context = decoder.userInfo[.context] as? NSManagedObjectContext else {
            throw [...]
        }
        ...
    }

    ...
}
```

Coming soon: Codable managed objects

```
public class Event: NSManagedObject, Codable {  
    required convenience public init(from decoder: Decoder) throws {  
        guard let context = decoder.userInfo[.context] as? NSManagedObjectContext else {  
            throw [...]  
        }  
        ...  
    }  
    ...  
}
```

Coming soon: Codable managed objects

```
public class Event: NSManagedObject: Codable {  
    required convenience public init(from decoder: Decoder) throws {  
        guard let context = decoder.userInfo[.context] as? NSManagedObjectContext else {  
            throw [...]  
        }  
        ...  
    }  
    ...  
}
```

mogenerator

Other options:

- output-dir**
- base-class-import TEXT**
- machine-dir**
- human-dir**

github.com/rentzsch/ mogenerator/

- swift42 branch merging soon

rentzsch / mogenerator		Unwatch ▾	112	★ Unstar	2,984	🍴 Fork	403
Core Data code generation		http://rentzsch.github.com/mogenerator					
7 commits		4 releases		67 contributors		MIT	
Branch: master ▾		New pull request		Create new file	Upload files	Find file	Clone or download ▾
👤	atomicbird	Add entity level "additionalImports", a comma-separated list of modul...					Latest commit aaea23b 5 days ago
📁	.github	Added pull request & issue templates.					2 years ago
📁	MiscMerge	Don't check for failure by looking at the NSError					2 years ago
📁	categories	Replaced RegexKitLite with NSRegularExpression.					3 years ago
📁	contributed templates	Reverted PonsoTest to version on rentzsch/master					7 years ago
📁	ddcli	instancetype support.					3 years ago
📁	installer	Bump version to 1.29, copyright to 2015.					3 years ago
📁	mogenerator.xcodeproj	Remove user-specific command line args					9 days ago
📁	momcom	Support "Custom Class" for Transformable attributes:					9 days ago
📁	ponso	instancetype support.					3 years ago
📁	templates	Add entity level "additionalImports", a comma-separated list of modul...					5 days ago
📁	test	Specify path to 10.13 SDK					11 months ago
📄	.gitignore	check-in `mogenerator` shared scheme					3 years ago

Other Core Data questions?

See me after class



Office hours, Noon - 12.25

Effective Core Data with Swift

Tom Harrington

@atomicbird