EBook Gratis

APRENDIZAJE core-data

Free unaffiliated eBook created from
Stack Overflow contributors.

#core-data

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: core-data

It is an unofficial and free core-data ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official core-data.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con los datos básicos

Observaciones

Core Data es un marco en varios SDK de SO de Apple, que incluye, entre otros, iOS y OS X. Tiene dos funciones principales: una capa de modelo y una capa de persistencia. La capa de modelo se utiliza en la gestión de objetos de modelo y datos persistentes. Simplemente puede almacenar y administrar datos en una interfaz orientada a objetos. Las características principales incluyen el filtrado, la consulta, la clasificación, la persistencia de datos y la creación de relaciones entre los datos. Otros temas de interés para los proyectos Core Data son NSPredicate, threading, y entre otros.

Una aplicación de ejemplo de Core Data podría ser una aplicación de catálogo para su biblioteca local. En la aplicación Catálogo, un bibliotecario podría agregar o eliminar libros. También podrían filtrar libros por género, ordenar libros por fecha de publicación o buscar un trabajo específico de autores. Un "Libro" de la entidad tendría varios atributos, como título, autor, fecha de publicación, isbn, número de llamada, etc. Los Datos básicos, incluido el ejemplo anterior, también pueden almacenar datos recopilados de un servidor.

Los componentes principales del marco incluyen:

- Modelos de datos (entidades, atributos y relaciones)
- Core Data Stack (NSPersistentStoreCoordinator, NSManagedObjectModel, NSManagedObjectContext)
- NSFetchRequest
- NSFetchedResultsController

Fuentes:

Documentación Marco

Guia de programacion

Notas de lanzamiento de datos básicos 2016

CoreData y Concurrencia

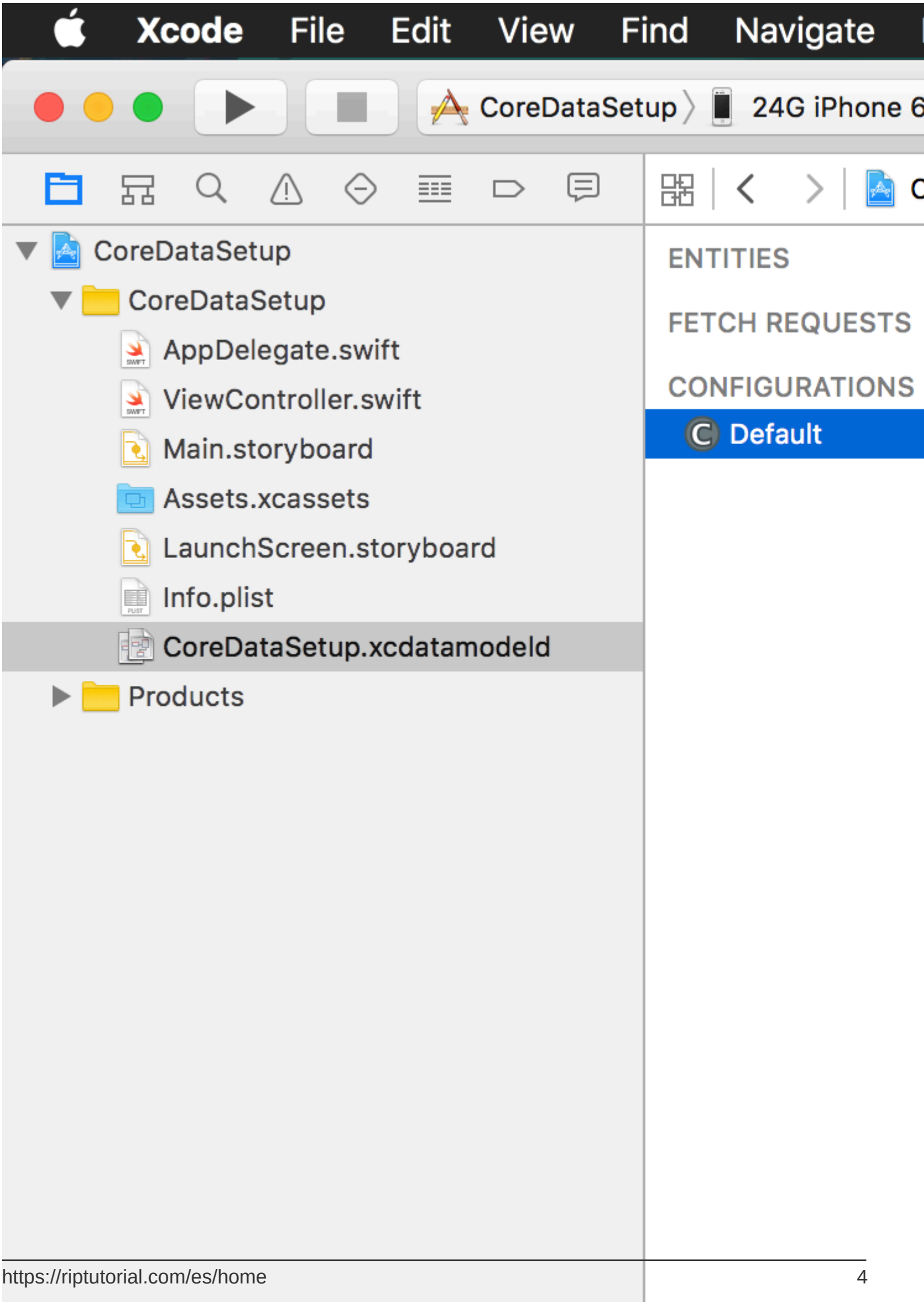
Es importante recordar que CoreData **NO** es seguro para subprocesos, lo que significa que si es necesario usar, por ejemplo, un subproceso de fondo para trabajar en ManagedObjects, hay cosas nuevas que se deben considerar, como *PrivateQueue- / MainQueue* - ManagedObjectContexts.

Desde el documental de Apple: *Core Data espera ser ejecutado en un solo hilo. Nunca debes compartir contextos de objetos gestionados entre hilos. Esta es una regla difícil que no debes romper.*

Examples

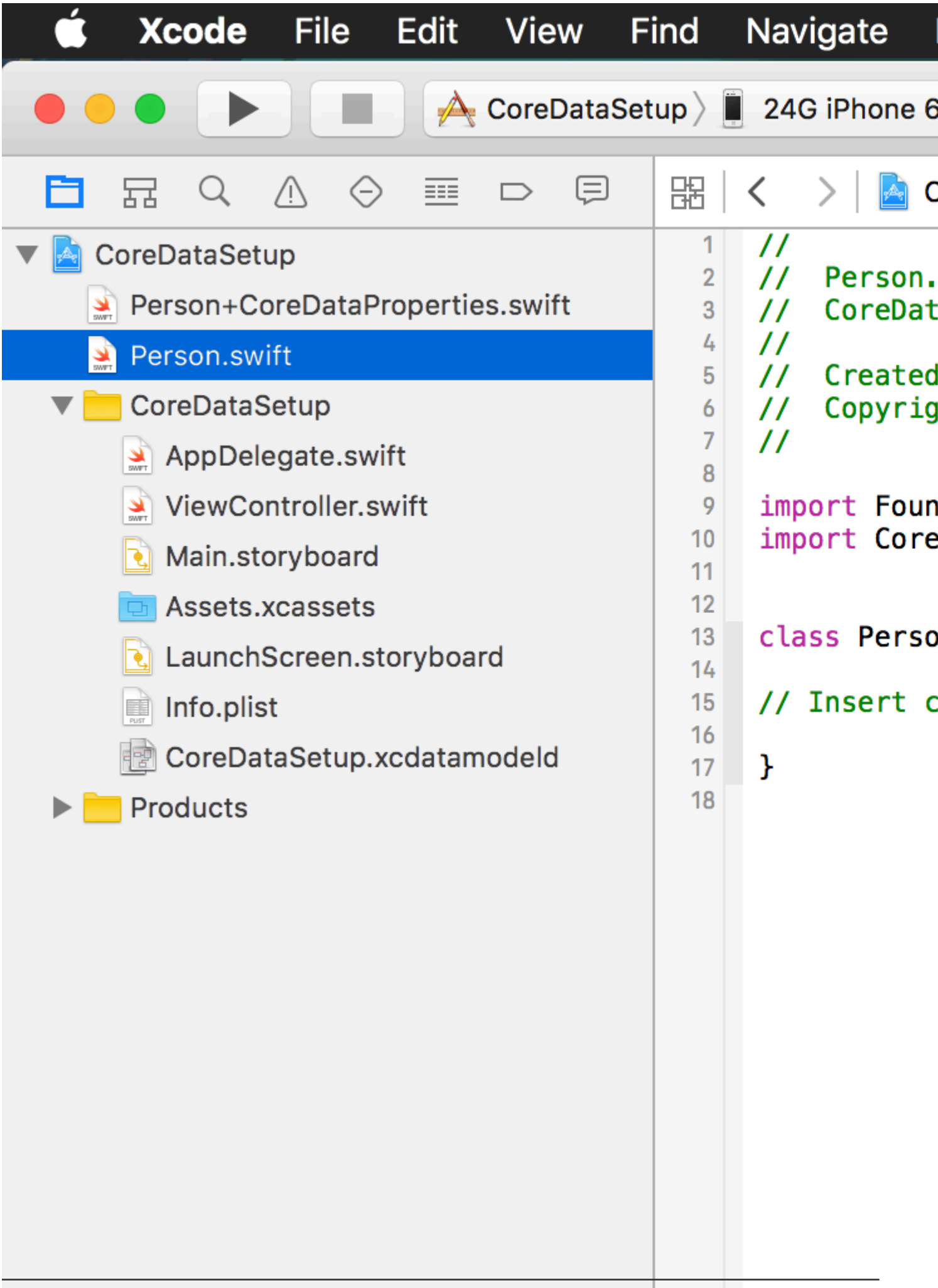
Creando tu primer modelo

- Seleccione el archivo `.xcdatamodeld` . Notarás que no tienes entidades. Tendrás que crear uno tú mismo. En la parte inferior de Xcode notará que un botón que dice "Agregar entidad" hace clic en él y tendrá una nueva entidad con la que podrá trabajar en el proyecto.


Xcode File Edit View Find Navigate
CoreDataSetup
24G iPhone 6
CoreDataSetup
CoreDataSetup
AppDelegate.swift
ViewController.swift
Main.storyboard
Assets.xcassets
LaunchScreen.storyboard
Info.plist
CoreDataSetup.xcdatamodeld
Products
ENTITIES
FETCH REQUESTS
CONFIGURATIONS
Default

- Todas las adiciones deben hacerse en `Person.swift` , ya que si alguna vez cambia su modelo y vuelve a ejecutar el generador de clases, se sobrescribirá todo en `Person+CoreDataProperties.swift` .

  , ya que si alguna vez cambia su modelo y vuelve a ejecutar el generador de clases, se sobrescribirá todo en `Person+CoreDataProperties.swift` .


Xcode File Edit View Find Navigate
CoreDataSetup
24G iPhone 6
CoreDataSetup
Person+CoreDataProperties.swift
Person.swift
CoreDataSetup
AppDelegate.swift
ViewController.swift
Main.storyboard
Assets.xcassets
LaunchScreen.storyboard
Info.plist
CoreDataSetup.xcdatamodeld
Products
1 //
2 // Person.
3 // CoreDat
4 //
5 // Created
6 // Copyrig
7 //
8
9 import Foun
10 import Core
11
12
13 class Perso
14
15 // Insert c
16
17 }
18

https://riptutorial.com/es/core-data/topic/1718/empezando-con-los-datos-basicos

Capítulo 2: Creación de un modelo de datos básicos

Observaciones

Los tipos de atributos incluyen: No definido, Entero 16, Entero 32, Entero 64, Decimal, Doble, Flotante, Cadena, Booleano, Fecha, Binario, Datos o Transformable

Al definir una `Entity` como abstracta, no creará ninguna instancia de esa entidad. Por ejemplo, una Persona sería abstracta y un Empleado o Cliente sería una subentidad concreta.

`Transient` atributos `Transient` son propiedades que define como parte del modelo, pero que no se guardan en el almacén persistente como parte de los datos de una instancia de entidad. Core Data realiza un seguimiento de los cambios que realiza en las propiedades transitorias, por lo que se registran para las operaciones de deshacer. Utiliza propiedades transitorias para una variedad de propósitos, incluyendo mantener valores calculados y valores derivados.

El campo `Destination` define qué objeto (u objetos) se devuelven cuando se accede a la relación en el código.

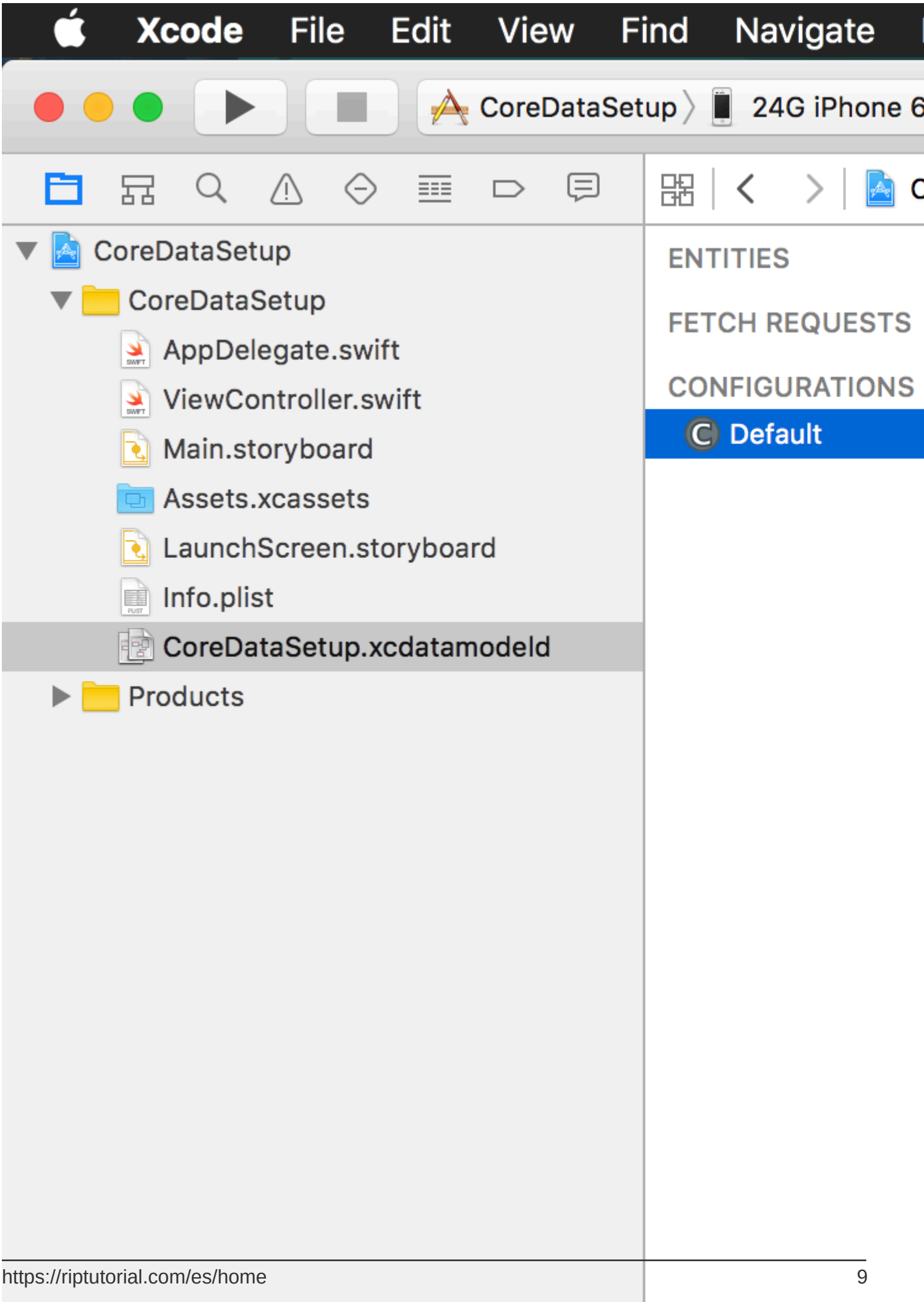
El campo `Inverse` define la otra mitad de una relación. Debido a que cada relación se define desde una dirección, este campo une dos relaciones para crear una relación completamente bidireccional.

Fuente: Guía de programación de datos básicos

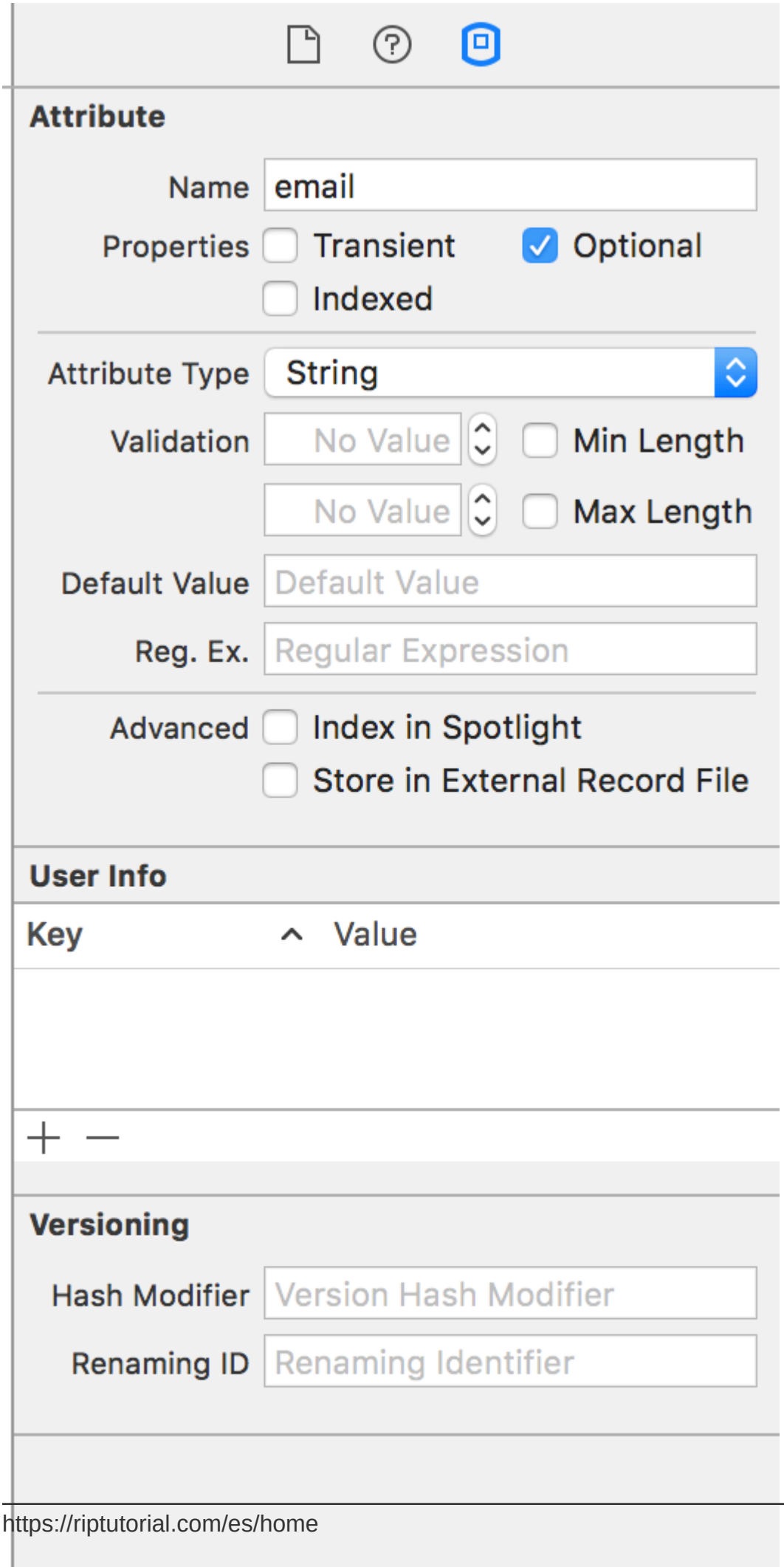
Examples

Agregar una entidad a un modelo de datos básico

1. Primero, es importante comprender que el modelo de datos principales es el archivo `*.xcdatamodeld` . Notarás que no tienes entidades. Tendrás que crear uno tú mismo. En la parte inferior de Xcode, verá un botón que dice "Agregar entidad", haga clic en él y tendrá una nueva entidad en el área del navegador con la que podrá trabajar en el proyecto.


Xcode
File
Edit
View
Find
Navigate
CoreDataSetup
24G iPhone 6
CoreDataSetup
CoreDataSetup
AppDelegate.swift
ViewController.swift
Main.storyboard
Assets.xcassets
LaunchScreen.storyboard
Info.plist
CoreDataSetup.xcdatamodeld
Products
ENTITIES
FETCH REQUESTS
CONFIGURATIONS
Default

por ejemplo, si estuviera agregando un correo electrónico, podría proporcionar una cadena de expresiones regulares `".+@([A-Za-z0-9-]+\\.)+[A-Za-z]{2}[A-Za-z]*"` para evitar que las direcciones postales se agreguen a su atributo de correo electrónico. La validación podría permitir un carácter mínimo y máximo para un número de teléfono.


Attribute
Name
email
Properties
Transient
Optional
Indexed
Attribute Type
String
Validation
No Value
Min Length
No Value
Max Length
Default Value
Default Value
Reg. Ex.
Regular Expression
Advanced
Index in Spotlight
Store in External Record File
User Info
Key
Value
Versioning
Hash Modifier
Version Hash Modifier
Renaming ID
Renaming Identifier

https://riptutorial.com/es/core-data/topic/2853/creacion-de-un-modelo-de-datos-basicos

Capítulo 3: NSFetchedResultsController

Introducción

NSFetchedResultsController es una conexión entre la tabla de datos de núcleo (entidad en datos de núcleo, tabla en sqlite) y UITableView. UITableView se puede adjuntar a cualquier entidad de datos centrales utilizando NSFetchedResultsController y UITableView se actualizará cuando la base de datos principales actualice esa entidad / tabla.

Examples

NSFetchedResultsController para UITableView

```
class ConversationsTableViewController: UITableViewController,
NSFetchedResultsControllerDelegate {

private var fetchedResultsController: NSFetchedResultsController<Conversation>!

override func viewDidLoad() {
    super.viewDidLoad()
    initializeFetchedResultsController()
}
private func initializeFetchedResultsController() {
    let request = NSFetchRequest<Conversation>(entityName: "Conversation")
    let timeSort = NSSortDescriptor(key: "lastMessageTime", ascending: false)
    request.sortDescriptors = [timeSort]

    let MOC = AppManagedObjectContext()//It should be main thread MOC
    fetchedResultsController = NSFetchedResultsController(fetchRequest: request,
managedObjectContext: MOC, sectionNameKeyPath: nil, cacheName: nil)
    fetchedResultsController.delegate = self

    do {
        try fetchedResultsController.performFetch()
    } catch {
        print("Failed to initialize FetchedResultsController: \(error)")
    }
}

//table view methods
override func numberOfSections(in tableView: UITableView) -> Int {
    if let n = fetchedResultsController?.sections!.count {
        return n
    }
    return 0
}
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    let sectionInfo = fetchedResultsController.sections![section]
    let n =  sectionInfo.numberOfObjects
    return n
}
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "identifier", for: indexPath)
```

```
    //configure cell
    return cell
}

//NSFetchedResultsController Delegates
func controllerWillChangeContent(_ controller:
NSFetchedResultsController<NSFetchRequestResult>) {
    tableView.beginUpdates()
}

func controller(_ controller: NSFetchedResultsController<NSFetchRequestResult>, didChange
sectionInfo: NSFetchedResultsSectionInfo, atSectionIndex sectionIndex: Int, for type:
NSFetchedResultsChangeType) {
    switch type {
    case .insert:
        tableView.insertSections(IndexSet(integer: sectionIndex), with: .fade)
    case .delete:
        tableView.deleteSections(IndexSet(integer: sectionIndex), with: .fade)
    case .move:
        break
    case .update:
        break
    }
}

func controller(_ controller: NSFetchedResultsController<NSFetchRequestResult>, didChange
anObject: Any, at indexPath: IndexPath?, for type: NSFetchedResultsChangeType, newIndexPath:
IndexPath?) {
    switch type {
    case .insert:
        tableView.insertRows(at: [newIndexPath!], with: .fade)
    case .delete:
        tableView.deleteRows(at: [indexPath!], with: .fade)
    case .update:
        tableView.reloadRows(at: [indexPath!], with: .none)
    case .move:
        tableView.deleteRows(at: [indexPath!], with: .fade)
        tableView.insertRows(at: [newIndexPath!], with: .fade)
    }
}

func controllerDidChangeContent(_ controller:
NSFetchedResultsController<NSFetchRequestResult>) {
    tableView.endUpdates()
}
}
```

Lea NSFetchedResultsController en línea: https://riptutorial.com/es/core-data/topic/9985/nsfetchedresultscontroller

Capítulo 4: Ordenar descriptores

Examples

Datos de pedido devueltos por solicitudes de recuperación

Establezca la propiedad NSFetchRequest sortDescriptors para determinar cómo se devuelven los datos.

```
let fetchRequest = NSFetchRequest(entityName: "NAME_OF_ENTITY")
let sortDescriptor = NSSortDescriptor(key: "NAME_OF_ATTRIBUTE", ascending: true)
fetchRequest.sortDescriptors = [sortDescriptor]
```

Descriptores de orden múltiple

También puede configurar múltiples descriptores de clasificación, para ordenar por un atributo dentro de otro. Por ejemplo, devuelva todas las entradas ordenadas por fecha y por nombre dentro de cada fecha:

```
let fetchRequest = NSFetchRequest(entityName: "NAME_OF_ENTITY")
let sortDescriptor1 = NSSortDescriptor(key: "name", ascending: true)
let sortDescriptor2 = NSSortDescriptor(key: "date", ascending: true)
fetchRequest.sortDescriptors = [sortDescriptor1, sortDescriptor2]
```

Lea Ordenar descriptores en línea: https://riptutorial.com/es/core-data/topic/5971/ordenar-descriptores

Capítulo 5: Pila de datos del núcleo

Observaciones

Esta es una implementación de la pila de datos principales que se coloca inicialmente en el archivo `AppDelegate` si el proyecto se crea con los datos principales cuando se crea el proyecto. Estas funciones también pueden implementarse en clases separadas para `CoreDataStack.swift` . Una de las funciones principales es obtener NSManagedObjectContext.

C objetivo

```
- (NSManagedObjectContext *)managedObjectContext {...}
```

Swift 2

```
lazy var managedObjectContext: NSManagedObjectContext = {...}
```

Swift 3

```
lazy var persistentContainer: NSPersistentContainer = {...)
let managedObjectContext = persistentContainer.viewContext
```

La pila de datos principales que se comunica entre los objetos en su aplicación y los almacenes de datos externos. La pila Core Data maneja todas las interacciones con los almacenes de datos externos para que su aplicación pueda centrarse en su lógica de negocios. La pila consta de tres objetos primarios: el contexto del objeto administrado (`NSManagedObjectContext`), el coordinador de tienda persistente (`NSPersistentStoreCoordinator`) y el modelo de objeto administrado (`NSManagedObjectModel`).

NSManagedObjectModel

La instancia de `NSManagedObjectModel` describe los datos a los que la pila Core Data accederá. `NSManagedObjectModel` (a menudo denominado "mamá") se carga en la memoria como el primer paso en la creación de la pila. Un ejemplo de `NSManagedObjectModel` es DataModel.momd. El `NSManagedObjectModel` define la estructura de los datos

NSPersistentStoreCoordinator

El `NSPersistentStoreCoordinator` realiza los objetos de los datos en el almacén persistente y pasa esos objetos al `NSManagedObjectContext` solicitante. Crea nuevas instancias de las entidades en el modelo y recupera instancias existentes de un almacén persistente (`NSPersistentStore`). El `NSPersistentStoreCoordinator` también verifica que los datos se encuentren en un estado coherente que coincida con las definiciones en `NSManagedObjectModel` .

NSManagedObjectContext

Cuando recuperas objetos de un almacén persistente, traes copias temporales al bloc de notas donde forman un gráfico de objetos (o una colección de gráficos de objetos). Luego puede modificar esos objetos, a menos que realmente guarde esos cambios, sin embargo, el almacén persistente permanece inalterado.

Todos los objetos gestionados deben registrarse con un contexto de objeto gestionado. Utiliza el contexto para agregar objetos al gráfico de objetos y eliminar objetos del gráfico de objetos. El contexto rastrea los cambios que realiza, tanto a los atributos de los objetos individuales como a las relaciones entre los objetos. Al realizar un seguimiento de los cambios, el contexto puede proporcionarle ayuda para deshacer y rehacer. También garantiza que si cambia las relaciones entre los objetos, se mantiene la integridad del gráfico de objetos.

Cuando guarda los cambios, el contexto garantiza que sus objetos se encuentren en un estado válido. Los cambios se escriben en el almacén (o almacenes) persistentes, se agregan nuevos registros para los objetos que creó y los registros se eliminan para los objetos que eliminó.

Fuente: Apple Core Data Programming: Inicializando la pila de datos Core

Examples

Objective-C Ejemplo

Esta es una configuración simple pero robusta de datos básicos para iOS 10+. Hay exactamente dos formas de acceder a los datos principales:

1. **viewContext** . El `viewContext` solo se puede usar desde el hilo principal, y solo para leer.
2. **EnqueueCoreDataBlock fuerte** . Toda la escritura debe hacerse utilizando `enqueueCoreDataBlock` . No hay necesidad de guardar al final, se guardará automáticamente. Todas las escrituras se ponen en cola en una OperationQueue para que nunca haya conflictos de escritura.

Asegúrese de que NUNCA use ningún objeto gestionado del contexto en otro contexto. También descarte todos los objetos que se crean o se recuperan en `enqueueCoreDataBlock` ya que el contexto que los respalda se destruirá después de que se ejecute el bloque.

// CoreDataManager.h

```
@interface CoreDataManager : NSObject
@property (nonatomic, readonly) NSManagedObjectContext * viewContext;
- (void)enqueueCoreDataBlock:(void (^)(NSManagedObjectContext* context))block;
@end
```

// CoreDataManager.m

```
@implementation NSManagedObjectContext(SaveIfNeeded)
-(BOOL) saveIfNeeded{
    BOOL toReturn = YES;
```

```
    if ([self hasChanges]) {
        NSError *error;
        toReturn = [self save:&error];
        if (toReturn == NO || error)
        {
            //Here you should log to your analytics service
            NSLog(@"--- Failed to commit data\n error: %@", error);
        }
    }
    return toReturn;
}
@end
@interface CoreDataManager ()
@property (nonatomic, strong) NSPersistentContainer* persistentContainer;
@property (nonatomic, strong) NSOperationQueue* persistentContainerQueue;
@end
@implementation CoreDataManager

- (id)init
{
    self = [super init]
    if (self)
    {
        self.persistentContainer = [[NSPersistentContainer alloc]
initWithName:@"PROJECT_NAME_ALSO_NAME_OF_MODEL" managedObjectModel:managedObjectModel];
        [self.persistentContainer
loadPersistentStoresWithCompletionHandler:^(NSPersistentStoreDescription * description,
NSError * error) {
        }];
        self.persistentContainer.viewContext.automaticallyMergesChangesFromParent = YES;
        _persistentContainerQueue = [[NSOperationQueue alloc] init];
        _persistentContainerQueue.maxConcurrentOperationCount = 1;
        _persistentContainerQueue.name = @"persistentContainerQueue";
        dispatch_queue_t queue =
dispatch_queue_create("persistentContainerQueue.dispatchQueue", DISPATCH_QUEUE_SERIAL);
        _persistentContainerQueue.underlyingQueue = queue;
    }
}

- (void)enqueueCoreDataBlock:(void (^)(NSManagedObjectContext* context))block{
    void (^blockCopy)(NSManagedObjectContext*) = [block copy];

    [self.persistentContainerQueue addOperation:[NSBlockOperation blockOperationWithBlock:^{
        NSManagedObjectContext* context =  self.persistentContainer.newBackgroundContext;
        [context performBlockAndWait:^{
            blockCopy(context);
            [context saveIfNeeded];
        }];
    }]];
}

-(NSManagedObjectContext*) viewContext{
    if (![NSThread mainThread]) {
        //here you should log to you analytics service. If you are in developer mode you
should crash to force you to fix this
        NSLog(@"access context on wrong thread!!");
    }
    return self.persistentContainer.viewContext;
}
```

Swift 2 Ejemplo

```
// Core Data stack

lazy var applicationDocumentsDirectory: NSURL = {
    let urls = NSFileManager.defaultManager().URLsForDirectory(.DocumentDirectory, inDomains:
.UserDomainMask)
    return urls[urls.count-1]
}()

lazy var managedObjectModel: NSManagedObjectModel = {
    let modelURL = NSBundle.mainBundle().URLForResource("ProjectName", withExtension: "momd")!
    return NSManagedObjectModel(contentsOfURL: modelURL)!
}()

lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {

    let coordinator = NSPersistentStoreCoordinator(managedObjectModel:
self.managedObjectModel)
    let url =
self.applicationDocumentsDirectory.URLByAppendingPathComponent("SingleViewCoreData.sqlite")
    var failureReason = "There was an error creating or loading the application's saved data."
    do {
        try coordinator.addPersistentStoreWithType(NSSQLiteStoreType, configuration: nil, URL:
url, options: nil)
    } catch {
        var dict = [String: AnyObject]()
        dict[NSLocalizedDescriptionKey] = "Failed to initialize the application's saved data"
        dict[NSLocalizedFailureReasonErrorKey] = failureReason

        dict[NSUnderlyingErrorKey] = error as NSError
        let wrappedError = NSError(domain: "YOUR_ERROR_DOMAIN", code: 9999, userInfo: dict)
        print("Unresolved error \(wrappedError), \(wrappedError.userInfo)")
        abort()
    }

    return coordinator
}()

lazy var managedObjectContext: NSManagedObjectContext = {
    let coordinator = self.persistentStoreCoordinator
    var managedObjectContext = NSManagedObjectContext(concurrencyType:
.MainQueueConcurrencyType)
    managedObjectContext.persistentStoreCoordinator = coordinator
    return managedObjectContext
}()

// Core Data Saving support

func saveContext () {
    if managedObjectContext.hasChanges {
        do {
            try managedObjectContext.save()
        } catch {
            let nserror = error as NSError
            print("Unresolved error \(nserror), \(nserror.userInfo)")
            abort()
        }
    }
}
```

Ejemplo de iOS 10 en Swift

```
lazy var persistentContainer: NSPersistentContainer = {

    let container = NSPersistentContainer(name: "ProjectName")
    container.loadPersistentStores(completionHandler: { (storeDescription, error) in

        if let error = error {
            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })
    return container
}()

func saveContext () {
    let context = persistentContainer.viewContext

    do {
        try context.save()
    } catch {
        let nserror = error as NSError
        fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
    }

    if context.hasChanges {
        print("changes occured")
    }else {
        print("no changes occured")
    }

}
```

Lea Pila de datos del núcleo en línea: https://riptutorial.com/es/core-data/topic/2596/pila-de-datos-del-nucleo

Capítulo 6: Uso de predicados

Examples

Coincidiendo con una cadena exacta

```
let fetchRequest = NSFetchRequest(entityName: "Foo")
var thePredicate: NSPredicate?
thePredicate = NSPredicate(format: "message == 'example'")
```

> La entidad `Foo` tiene un atributo de cadena de `message`

Sustituciones

En lugar de pasar una cadena estática como criterio de un predicado. Es posible sustituir valores mediante el uso de especificadores de formato. Hay cinco especificadores de formato:

- `%K` es una sustitución var arg para una ruta clave.
- `%@` es una sustitución var arg para un valor de objeto, a menudo una cadena, un número, una fecha o una matriz.
- `%ld` es una sustitución var arg para un valor int.
- `%la` es una sustitución var arg para un doble.
- `%a` es una sustitución var arg para un flotador.

En el siguiente ejemplo, el especificador de formato `%K` sirve como el argumento de la izquierda que pasa dinámicamente en la propiedad *"mensaje"* . El especificador de formato `%@` sirve como el argumento de la mano derecha para pasar dinámicamente en una cadena que contiene la palabra *"ejemplo"* .

```
let predicate = NSPredicate(format:"%K == %@", "message", "example")
```

Lea Uso de predicados en línea: https://riptutorial.com/es/core-data/topic/4283/uso-de-predicados

Creditos

S. No	Capítulos	Contributors
1	Empezando con los datos básicos	Asdrubal, Community, Josh Caswell, rgeorge, Santa Claus, TMob
2	Creación de un modelo de datos básicos	Asdrubal
3	NSFetchedResultsController	D4ttatraya
4	Ordenar descriptores	Jonas
5	Pila de datos del núcleo	Asdrubal, Jon Rose, Santa Claus
6	Uso de predicados	Dan Beaulieu, David