

CoreData in ios

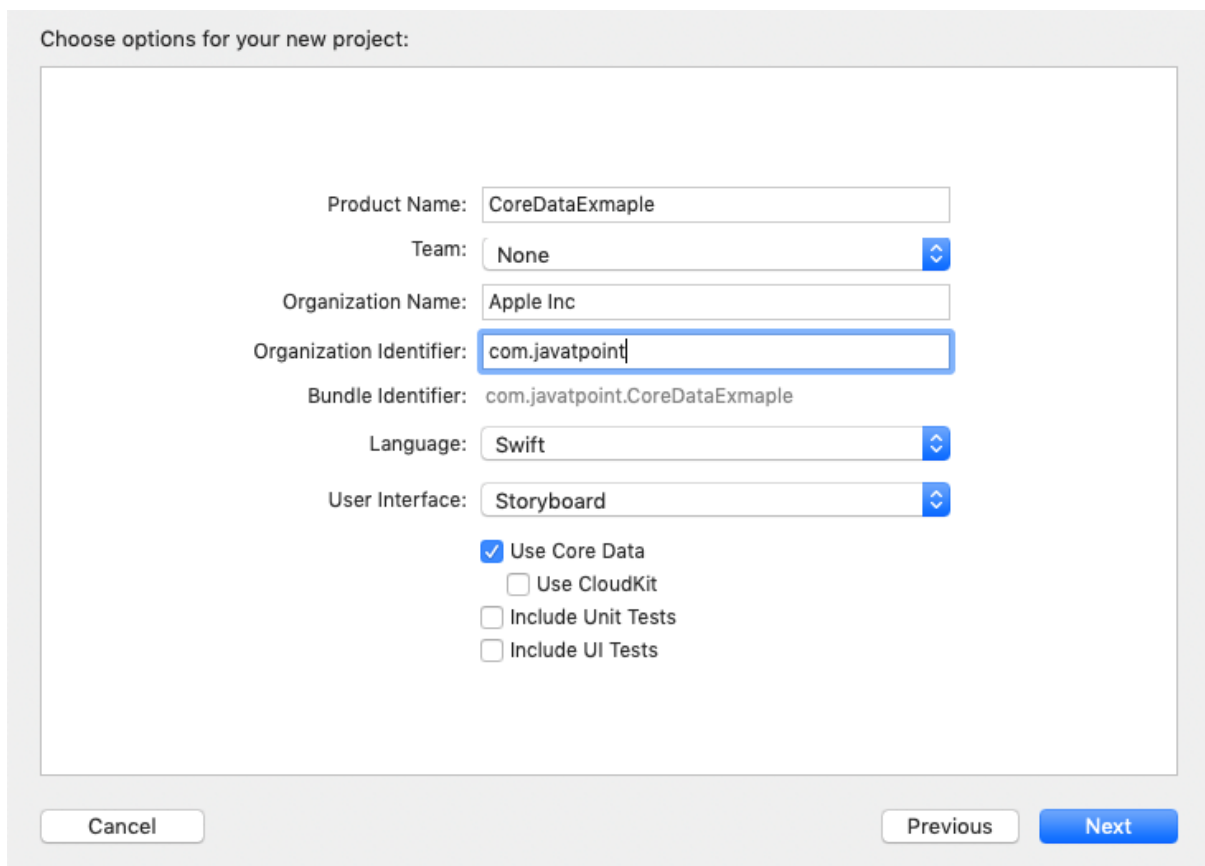
CoreData is the framework provided by Apple to save, track, filter, and modify the data within the iOS applications. It is not the database, but it uses SQLite as its persistent store. It is used to manage the model layer object in our application. It manages the object graphs, tracks the changes in the data, and modifies the data on the user interactions.

In the previous section of the tutorial, we have seen how we can use user defaults to store the short pieces of data. However, in this section of the tutorial, we will discuss how to use the actual database framework like CoreData to store, modify, and filter the user's data in the application's database.

How to use CoreData in iOS application

Here, we will create an iOS application that includes CoreData as a framework to store and persist the data objects.

Let's create a single view iOS application to demonstrate the basics of CoreData. To enable the app to use CoreData, we must check the Use CoreData option displayed at the bottom.



Choose options for your new project:

Product Name: CoreDataExmaple

Team: None

Organization Name: Apple Inc

Organization Identifier: com.javatpoint

Bundle Identifier: com.javatpoint.CoreDataExmaple

Language: Swift

User Interface: Storyboard

Use Core Data
 Use CloudKit
 Include Unit Tests
 Include UI Tests

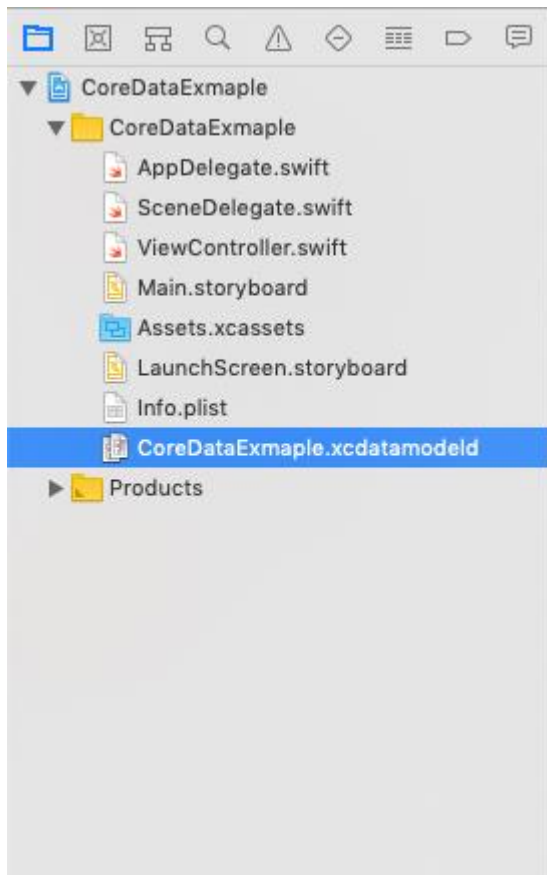
Cancel Previous Next

Now, we have successfully created the CoreDataExample project with CoreData support. There are two notable changes in the XCode project if we observe. First, The Core Data stack code has been added to the AppDelegate file. The CoreData stack code comes with clear documentation in the form of short comments. It sets up the persistentContainer and saves the data if there are any changes. The CoreData Stack Code is shown in the below image.

```
38     lazy var persistentContainer: NSPersistentContainer = {
39         /*
40         The persistent container for the application. This implementation
41         creates and returns a container, having loaded the store for the
42         application to it. This property is optional since there are legitimate
43         error conditions that could cause the creation of the store to fail.
44         */
45         let container = NSPersistentContainer(name: "CoreDataExmample")
46         container.loadPersistentStores(completionHandler: { (storeDescription, error) in
47             if let error = error as NSError? {
48                 // Replace this implementation with code to handle the error appropriately.
49                 // fatalError() causes the application to generate a crash log and terminate. You should not use this function
                    in a shipping application, although it may be useful during development.

51                 /*
52                 Typical reasons for an error here include:
53                 * The parent directory does not exist, cannot be created, or disallows writing.
54                 * The persistent store is not accessible, due to permissions or data protection when the device is locked.
55                 * The device is out of space.
56                 * The store could not be migrated to the current model version.
57                 Check the error message to determine what the actual problem was.
58                 */
59                 fatalError("Unresolved error \(error), \(error.userInfo)")
60             }
61         })
62         return container
63     }()
64
65 // MARK: - Core Data Saving support
66
67 func saveContext () {
68     let context = persistentContainer.viewContext
69     if context.hasChanges {
70         do {
71             try context.save()
72         } catch {
73             // Replace this implementation with code to handle the error appropriately.
74             // fatalError() causes the application to generate a crash log and terminate. You should not use this function
                    in a shipping application, although it may be useful during development.
75             let nerror = error as NSError
76             fatalError("Unresolved error \(nerror), \(nerror.userInfo)")
77         }
78     }
79 }
```

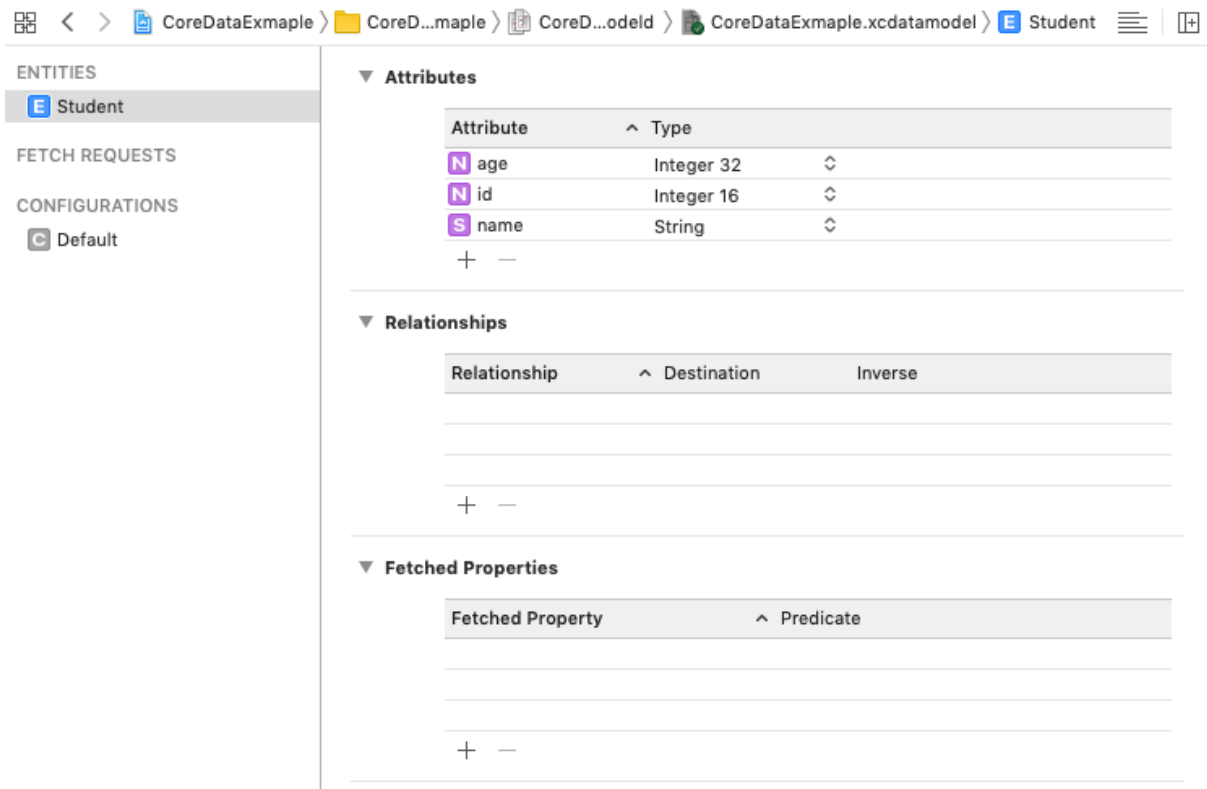
Second, the new file CoreDataExample.xcdatamodeld has been added to the Application's package. It acts as the model layer for the data. We can add the entity, attributes, and relations into the model layer.



Adding the Entity to CoreData Model

We can add an entity into the xcdatamodeld file by selecting an option Add Entity given at the bottom of the file. In the right pane of the file, we can add attributes, relationships, and fetched properties to the Entity.

Here, we have created a Student entity and added three attributes id, age, and name in the model, as shown in the below image.



Now, we have created our model Student. Let's add some records into this model. The model will be saved to the CoreData.

To add records to the model, we need to follow the following steps.

- Instantiate the persistentContainer.
- Create the context object.
- Create an entity object.
- Create a new record object.
- Set values for the records for each key.

To refer to the persistentContainer, we need to instantiate the AppDelegate. The instance of the AppDelegate is formed using the following code.

1. `let delegate = UIApplication.shared.delegate as! AppDelegate`

To create the context, we can use the persistentContainer reference using the below code.

1. `let context = delegate.persistentContainer.viewContext`

Now, we need to create the entity using the context reference, we just created.

1. `let entity = NSEntityDescription.entity(forEntityName: "Student", in: context)`

Now, we need to create the new Student record as the entity object.

1. `let newStudent = NSManagedObject(entity: entity!, insertInto: context)`

Now, let's add some records into the newly created entity object.

1. `newStudent.setValue("John", forKey: "name")`

2. `newStudent.setValue(23, forKey: "age")`

3. `newStudent.setValue(1, forKey: "id")`

We have created an AppDelegate object, the context, entity, and entity object. We have also set the values for the newly created entity object.

Now, we need to save the data inside CoreData. To save the data, we use the context object to save the context. We have to wrap this code with the try-catch block.

Now, if we run our application, our data will be saved inside the CoreData.

The AppDelegate.swift and ViewController.swift file contains the following code.

AppDelegate.swift

1. `import UIKit`

2. `import CoreData`

- 3.

4. `@UIApplicationMain`

5. `class AppDelegate: UIResponder, UIApplicationDelegate {`

- 6.

7. `func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {`

8. `// Override point for customization after application launch.`

9. `return true`

10. `}`

- 11.

- 12.

13. `// MARK: UISceneSession Lifecycle`

- 14.

- 15.

```

16. func application(_ application: UIApplication, configurationForConnecting connectingSceneSession: UISceneSession, options: UIScene.ConnectionOptions) -
    > UISceneConfiguration {
17.     // Called when a new scene session is being created.
18.     // Use this method to select a configuration to create the new scene with.
19.     return UISceneConfiguration(name: "Default Configuration", sessionRole: connectingSceneSession.role)
20. }
21.
22.
23. func application(_ application: UIApplication, didDiscardSceneSessions sceneSessions: Set<UISceneSession>) {
24.     // Called when the user discards a scene session.
25.     // If any sessions were discarded while the application was not running, this will be called shortly after application:didFinishLaunchingWithOptions.
26.     // Use this method to release any resources that were specific to the discarded scenes, as they will not return.
27. }
28.
29.
30. // MARK: - Core Data stack
31.
32.
33. lazy var persistentContainer: NSPersistentContainer = {
34.     /*
35.     The persistent container for the application. This implementation creates and returns a container, having loaded the store for the application to it. This property is optional since there are legitimate error conditions that could cause the creation of the store to fail.
36.     */
37.     let container = NSPersistentContainer(name: "CoreDataExample")
38.     container.loadPersistentStores(completionHandler: { (storeDescription, error) in
39.         if let error = error as NSError? {
40.             // Replace this implementation with code to handle the error appropriately.

```

```

44.         // fatalError() causes the application to generate a crash log and terminate.
           You should not use this function in a shipping application, although it may be useful
           during development.
45.
46.         /*
47.         Typical reasons for an error here include:
48.         * The parent directory does not exist, cannot be created, or disallows writin
           g.
49.         * The persistent store is not accessible, due to permissions or data protecti
           on when the device is locked.
50.         * The device is out of space.
51.         * The store could not be migrated to the current model version.
52.         Check the error message to determine what the actual problem was.
53.         */
54.         fatalError("Unresolved error \(error), \(error.userInfo)")
55.     }
56. })
57.     return container
58. })
59.
60.
61. // MARK: - Core Data Saving support
62.
63.
64. func saveContext () {
65.     let context = persistentContainer.viewContext
66.     if context.hasChanges {
67.         do {
68.             try context.save()
69.         } catch {
70.             // Replace this implementation with code to handle the error appropriately.
71.
           // fatalError() causes the application to generate a crash log and terminate.
           You should not use this function in a shipping application, although it may be useful
           during development.
72.             let nsetError = error as NSError
73.             fatalError("Unresolved error \(nsetError), \(nsetError.userInfo)")

```

```
74.     }
75.   }
76. }
77. }
```

ViewController.swift

```
1. import UIKit
2. import CoreData
3.
4.
5. class ViewController: UIViewController {
6.
7.
8.     override func viewDidLoad() {
9.         super.viewDidLoad()
10.        // Do any additional setup after loading the view.
11.
12.        let appDelegate = UIApplication.shared.delegate as! AppDelegate
13.        let context = appDelegate.persistentContainer.viewContext
14.        let entity = NSEntityDescription.entity(forEntityName: "Student", in: context)
15.        let newStudent = NSManagedObject(entity: entity!, insertInto: context)
16.        newStudent.setValue("John", forKey: "name")
17.        newStudent.setValue(23, forKey: "age")
18.        newStudent.setValue(1, forKey: "id")
19.
20.        do{
21.            try context.save()
22.        }catch{
23.            debugPrint("Can't save")
24.        }
25.
26.    }
27. }
```

Fetching the records

Fetching the records from the CoreData is very simple. We need to instantiate the `NSFetchRequest` class and create a request object. We can pass this request object into the `fetch()` method for `NSManagedObjectContext` reference.

1. `let result = try context.fetch(request)`

The following code can be used to fetch the data from the CoreData.

```
let request = NSFetchRequest<NSFetchRequestResult>(entityName: "Student")
request.returnsObjectsAsFaults = false
do{
    let result = try context.fetch(request)
    for data in result{
        debugPrint((data as AnyObject).value(forKey: "name") as! String)
        debugPrint((data as AnyObject).value(forKey: "id") as! Int16)
        debugPrint((data as AnyObject).value(forKey: "age") as! Int32)
    }
}catch{

}
```

Now, the `ViewController.swift` file contains the following code.

```
import UIKit
import CoreData

class ViewController: UIViewController {

    let appDelegate = UIApplication.shared.delegate as! AppDelegate
    var context = NSManagedObjectContext()
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.

        context = appDelegate.persistentContainer.viewContext

        let entity = NSEntityDescription.entity(forEntityName: "Student", in: context)
        let newStudent = NSManagedObject(entity: entity!, insertInto: context)
        newStudent.setValue("John", forKey: "name")
    }
}
```

```
newStudent.setValue(23, forKey: "age")
newStudent.setValue(1, forKey: "id")
```

```
do{
    try context.save()
fetchData()
}catch{
    debugPrint("Can't save")
}
}
func fetchData(){
    let request = NSFetchedRequest<NSFetchedRequestResult>(entityName: "Student")
    request.returnsObjectsAsFaults = false
do{
    let result = try context.fetch(request)
    for data in result{
        debugPrint((data as AnyObject).value(forKey: "name") as! String)
        debugPrint((data as AnyObject).value(forKey: "id") as! Int16)
        debugPrint((data as AnyObject).value(forKey: "age") as! Int32)
    }
}catch{
}
}
}
```