Core Data

Core Data is an object graph and persistence framework provided by Apple in the macOS and iOS operating systems.

Use Core Data to save your application's permanent data for offline use, to cache temporary data, and to add undo functionality to your app on a single device.

Core Data abstracts the details of mapping your objects to a store, making it easy to save data from Swift and Objective-C without administering a database directly.

Core Data also helps keep your views and data synchronized by providing data sources for table and collection views.

Core Data is the **M** in **MVC**, the model layer of your application. Even though Core Data can persist data to disk, data persistence is actually an optional feature of the framework.

Core Data can only do its magic because it keeps the object graph it manages in memory. This means that it can only operate on records once they are in memory. This is very different from performing a SQL query on a database. If you want to delete thousands of records, Core Data first needs to load each record into memory.

Core Data allows us to store data in the device and make them accessible again even after the app terminates. **Core Data** is **NOT** an **SQLite database**. It is a powerful framework that can use **SQLite** to store data.

Create A New Core Data Object

After defining a new **UserEntity** entity in the **.xcdatamodeld** file, **UserEntity** can be used as a type in Swift. The **UserEntity** constructor takes in a **NSManagedObjectContext** and returns an instance of **UserEntity** with all properties set to default values.

New NSManagedObject Example

```
// Create a new UserEntity in the
// NSManagedObjectContext context
let user = UserEntity(context: context)

// Assign values to the entity's
properties
user.name = "User Name"
user.email = "user@domain.com"
user.age = 32

// To save the new entity to the
persistent store, call
// save on the context
do {
    try context.save()
}
```

```
catch {
    // Handle Error
}
```

Update A Core Data Object

```
// Update entity properties as needed
user.age = 33

// To save new entity updates to the
persistent store,
// call save on the context
do {
    try context.save()
}
catch {
    // Handle Error
}
```

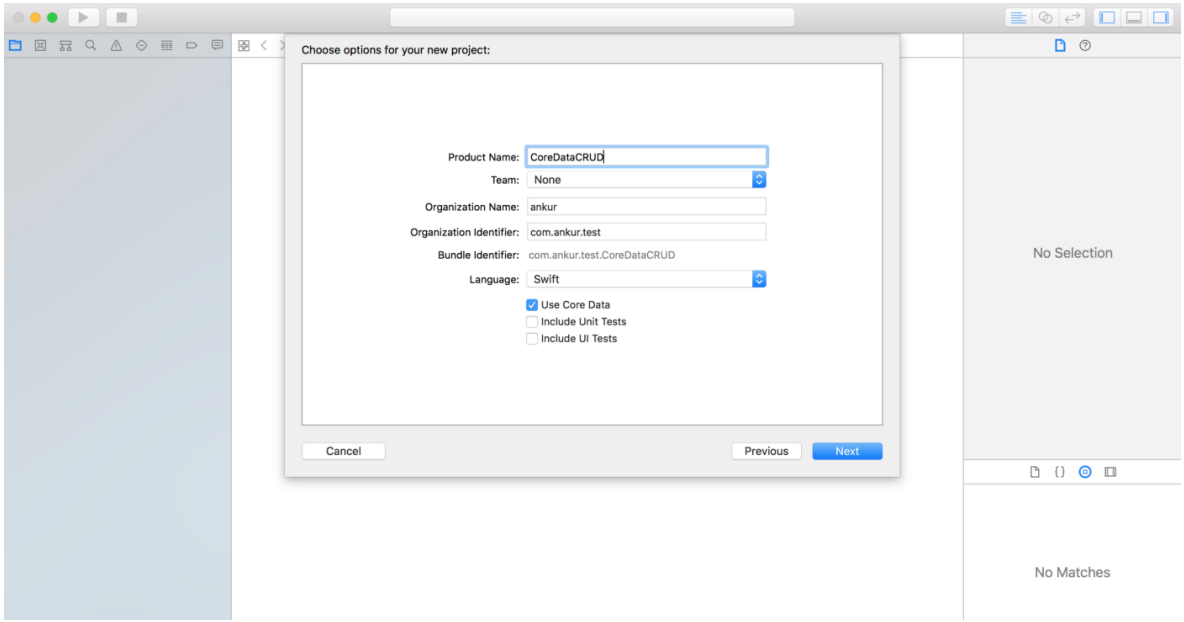
Delete A Core Data Object

```
// Delete the entity from the context
context.delete(user)

// To delete the entity from the
persistent store, call
// save on the context
do {
    try context.save()
}
catch {
    // Handle Error
}
```

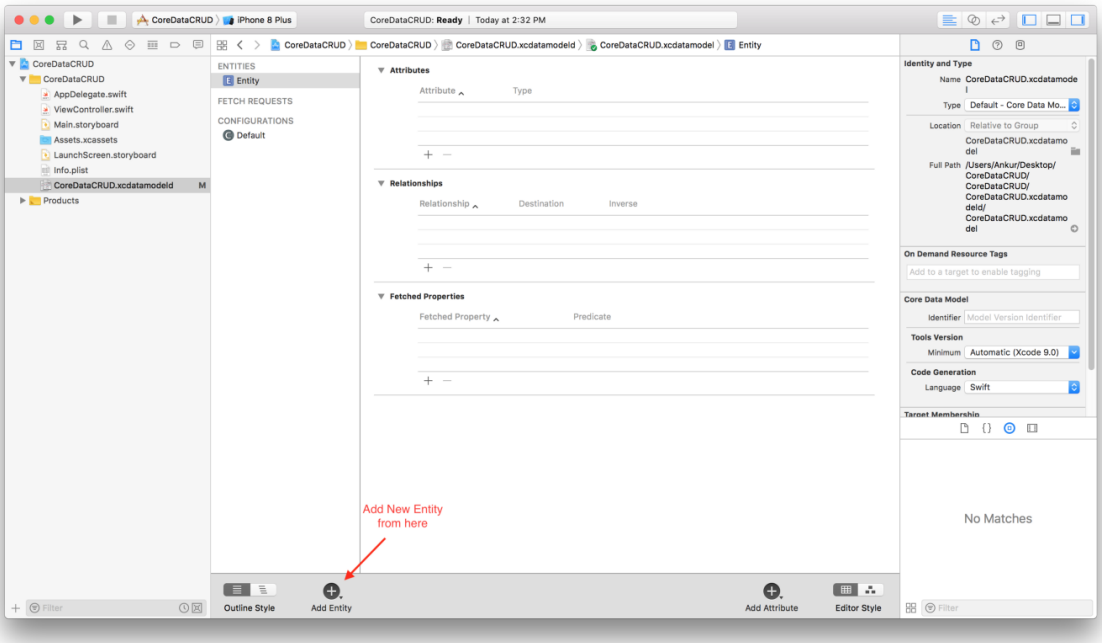
Core Data Demo (Create, Retrieve, Update and Delete)

In order to demonstrate basics of the Core Data, let's create single view iOS app with Core Data module selected.

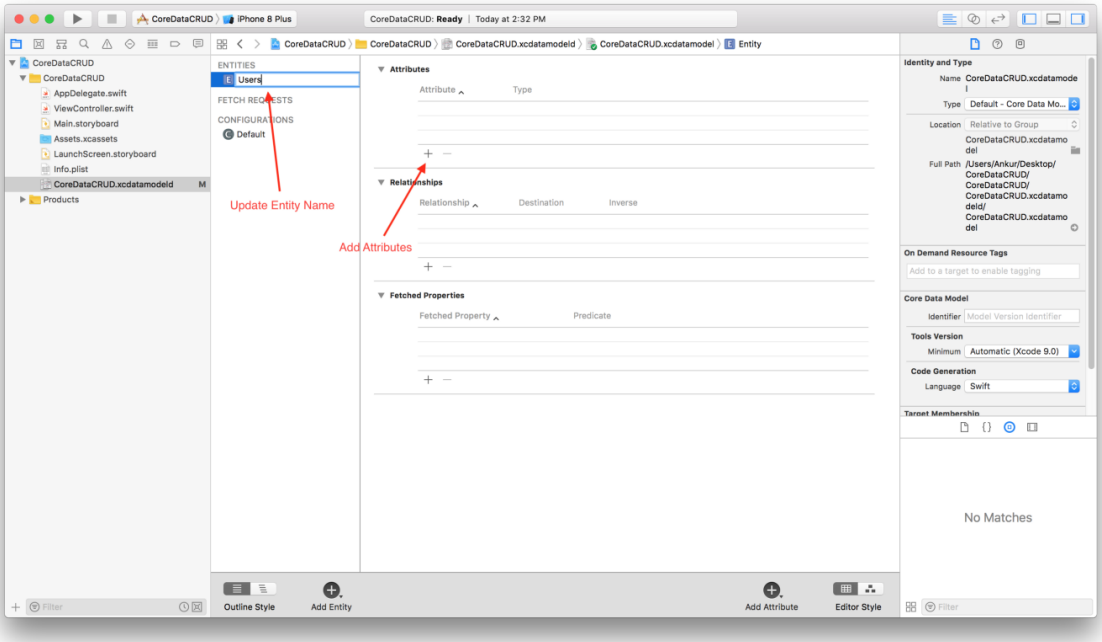


First, let's create a new project and let's select "**Use Core Data**". Although you can add that framework for your existing project, it's easier to add it from that place as everything will be wired up already for you.Once the project is created, you will see a file like ***CoreDataTest.xcdatamodeld*** added.

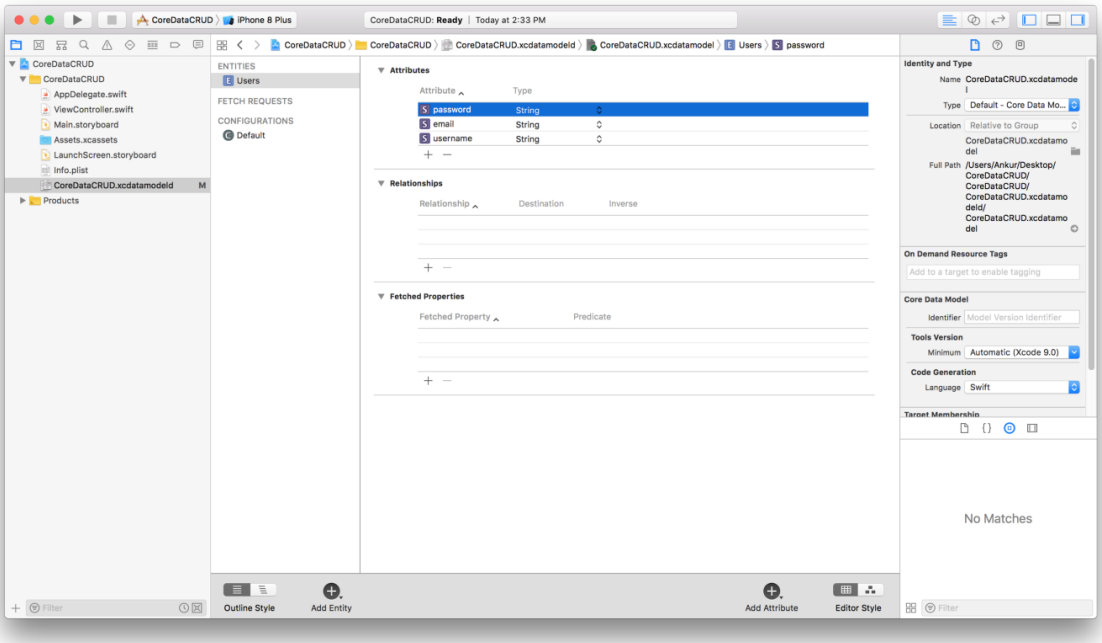
When you click it, you will see a tool that allows you to configure entities which represents data models. You can define few things for each entity there but for us **Attributes** and **Relationships** will be most important.



Add Entity

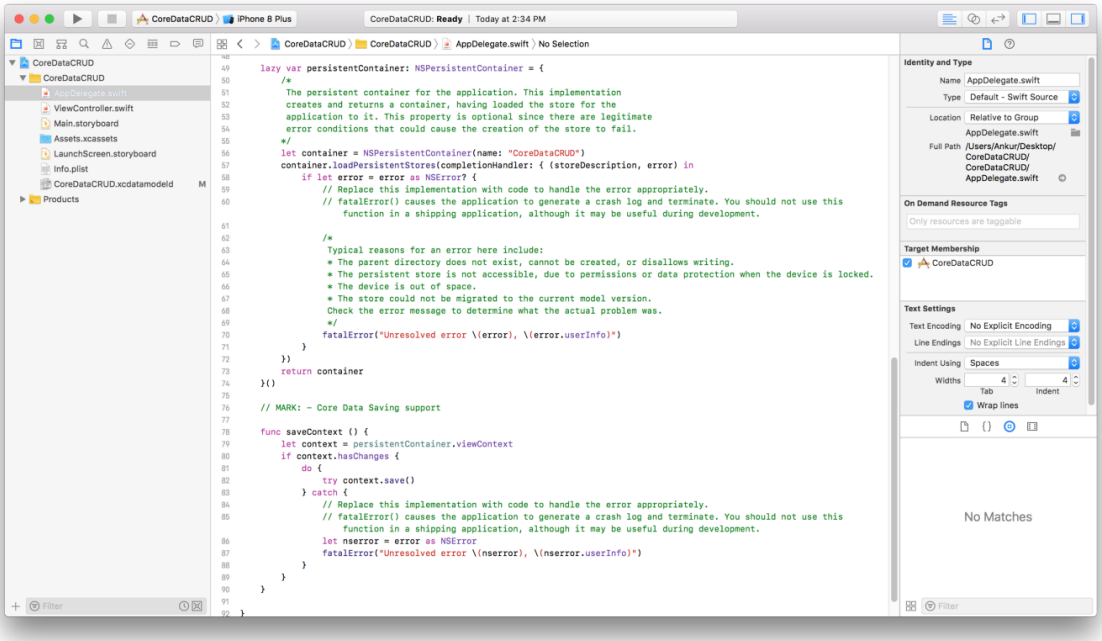


Configure Entity Name and Attributes



Now that, we have created a demo project with Core Data support. There are two notable changes in this Xcode template we can easily observe which are

- The new file **CoreDataCRUD.xcdatamodeld**
- The **AppDelegate.swift** file with Core Data Stack code



Core Data Stack

The Core Data Stack code inside the AppDelegate.swift has clear documentation in form of comments but in short, it set up the persistentContainer and save the data if there are any changes. As AppDelegate is the first file that executes as soon as app launched, we can save and fetch the context the from the Core Data Stack.

Now that, we have modelled our data in the User entity. It's time to add some records and save it into the CoreData

Start with **import CoreData to our viewcontroller.swift**

Create Records to Core Data

The process of adding the records to Core Data has following tasks

- Refer to persistentContainer from appdelegate
- Create the context from persistentContainer
- Create an entity with User
- Create new record with this User Entity
- Set values for the records for each key

```swift
func createData(){

    //As we know that container is set up in the AppDelegates so we need to refer that container.
    guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else { return }

    //We need to create a context from this container
    let managedContext = appDelegate.persistentContainer.viewContext

    //Now let's create an entity and new user records.
    let userEntity = NSEntityDescription.entity(forEntityName: "User", in: managedContext)!

    //final, we need to add some data to our newly created record for each keys using
    //here adding 5 data with loop

    for i in 1...5 {

        let user = NSManagedObject(entity: userEntity, insertInto: managedContext)
        user.setValue("Ankur\(i)", forKeyPath: "username")
        user.setValue("ankur\(i)@test.com", forKey: "email")
        user.setValue("ankur\(i)", forKey: "password")
    }

    //Now we have set all the values. The next step is to save them inside the Core Data

    do {
        try managedContext.save()

    } catch let error as NSError {
        print("Could not save. \(error), \(error.userInfo)")
    }
}
```

Retrieve Data

The process of fetching the saved data is very easy as well. It has the following task

- Prepare the request of type NSFetchRequest for the entity (User in our example)
- if required use predicate for filter data
- Fetch the result from context in the form of array of [NSManagedObject]
- Iterate through an array to get value for the specific key

We can fetch the data from our Users entity using following code.

```
    func retrieveData() {

        //As we know that container is set up in the AppDelegates so we need to refer that container.
        guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else { return }

        //We need to create a context from this container
        let managedContext = appDelegate.persistentContainer.viewContext

        //Prepare the request of type NSFetchRequest  for the entity
        let fetchRequest = NSFetchRequest<NSFetchRequestResult>(entityName: "User")

'/        fetchRequest.fetchLimit = 1
'/        fetchRequest.predicate = NSPredicate(format: "username = %@", "Ankur")
'/        fetchRequest.sortDescriptors = [NSSortDescriptor.init(key: "email", ascending: false)]
'/
        do {
            let result = try managedContext.fetch(fetchRequest)
            for data in result as! [NSManagedObject] {
                print(data.value(forKey: "username") as! String)
            }

        } catch {

            print("Failed")
        }
    }
```

Update Data

For update record first we have to fetch/Retrieve data with predicate as same as above Retrieve data process. Then below few steps to follow

Prepare the request with predicate for the entity (User in our example)

Fetch record and Set New value with key

And Last Save context same as create data.

```swift
func updateData(){

    //As we know that container is set up in the AppDelegates so we need to refer that container.
    guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else { return }

    //We need to create a context from this container
    let managedContext = appDelegate.persistentContainer.viewContext

    let fetchRequest:NSFetchRequest<NSFetchRequestResult> = NSFetchRequest.init(entityName: "User")
    fetchRequest.predicate = NSPredicate(format: "username = %@", "Ankur1")
    do
    {
        let test = try managedContext.fetch(fetchRequest)

            let objectUpdate = test[0] as! NSManagedObject
            objectUpdate.setValue("newName", forKey: "username")
            objectUpdate.setValue("newmail", forKey: "email")
            objectUpdate.setValue("newpassword", forKey: "password")
            do{
                try managedContext.save()
            }
            catch
            {
                print(error)
            }
        }
    catch
    {
        print(error)
    }

}
```

Delete Data

For delete record first we have to find object which we want to delete by fetchRequest. then follow below few steps for delete record

- Prepare the request with predicate for the entity (User in our example)
- Fetch record and which we want to delete
- And make context.delete(object) call (ref image attached below)

```swift
func deleteData(){

    //As we know that container is set up in the AppDelegates so we need to refer that container.
    guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else { return }

    //We need to create a context from this container
    let managedContext = appDelegate.persistentContainer.viewContext

    let fetchRequest = NSFetchRequest<NSFetchRequestResult>(entityName: "User")
    fetchRequest.predicate = NSPredicate(format: "username = %@", "Ankur3")

    do
    {
        let test = try managedContext.fetch(fetchRequest)

        let objectToDelete = test[0] as! NSManagedObject
        managedContext.delete(objectToDelete)

        do{
            try managedContext.save()
        }
        catch
        {
            print(error)
        }

    }
    catch
    {
        print(error)
    }
}
```

Well, this isn’t enough to core data, there are many complex things we can do with core data tracking data changes, adding Predicates and complex relationships of databases.