Article

Migrating your data model automatically

Enable lightweight migrations to keep your data model and the underlying data in a consistent state.

Overview

Core Data can typically perform an automatic data migration, referred to as lightweight migration. Lightweight migration infers the migration from the differences between the source and the destination managed object models.

Generating an inferred mapping model

To perform automatic lightweight migration, Core Data needs to be able to find the source and destination managed object models at runtime. It looks for models in the bundles returned by the `allBundles` and `allFrameworks` methods of the `Bundle` class. Core Data then analyzes the schema changes to persistent entities and properties, and generates an inferred mapping model.

Generating an inferred mapping model requires changes to fit an obvious migration pattern, for example:

- Addition of an attribute
- Removal of an attribute
- A nonoptional attribute becoming optional
- An optional attribute becoming non-optional, and defining a default value
- Renaming an entity or property

Managing changes to entities and properties

If you rename an entity or property, you can set the renaming identifier in the destination model to the name of the corresponding property or entity in the source model. Use the Xcode Data Modeling tool's property inspector (for either an entity or a property) to set the renaming identifier in the managed object model. For example, you can:

- Rename a Car entity to Automobile
- Rename a Car's color attribute to paintColor

The renaming identifier creates a canonical name, so set the renaming identifier to the name of the property in the source model (unless that property already has a renaming identifier). This means you can rename a property in version 2 of a model, then rename it again in version 3. The renaming will work correctly going from version 2 to version 3, or from version 1 to version 3.

Managing changes to relationships

Lightweight migration can also manage changes to relationships and to the type of relationship. You can add a new relationship or delete an existing relationship. You can also rename a relationship by using a renaming identifier, just like an attribute.

In addition, you can change a relationship from a to-one to a to-many, or a nonordered to-many to an ordered (and vice versa).

Managing changes to hierarchies

You can add, remove, and rename entities in the hierarchy. You can also create a new parent or child entity and move properties up and down the entity hierarchy. You can move entities out of a hierarchy, but you can't merge entity hierarchies. If two existing entities don't share a common parent in the source, they can't share a common parent in the destination.

Confirming whether Core Data can infer the model

If you want to determine in advance whether Core Data can infer the mapping model between the source and destination models without actually doing the work of migration, you can use `inferredMappingModel(forSourceModel:destinationModel:)` method. The method returns the inferred model if Core Data is able to create it; otherwise, it returns `nil`.

If your data change exceeds the capabilities of automatic migration, you can perform a heavyweight migration (often referred to as manual migration).

Requesting lightweight migration

You request automatic lightweight migration using the options dictionary that you pass into `addPersistentStore(ofType:configurationName:at:options:)`. Set values corresponding to both the `NSMigratePersistentStoresAutomaticallyOption` and the `NSInferMappingModelAutomaticallyOption` keys to `true`:

```
// Create a persistent store coordinator with a managed object model.
let coordinator = NSPersistentStoreCoordinator(managedObjectModel: managedObjectMode

// Create a URL to the data store.

do {
    // Set the options to enable lightweight data migrations.
    let options = [NSMigratePersistentStoresAutomaticallyOption: true,
                   NSInferMappingModelAutomaticallyOption: true]

    // Set the options when you add the store to the coordinator.
    _ = try coordinator.addPersistentStore(type: .sqlite,
                                           at: storeURL,
                                           options: options)
} catch {
    // Handle the error appropriately. It's useful to use
    // `fatalError(_:file:line:)` during development.
    fatalError("Failed to add persistent store: \(error.localizedDescription)")
}
```

With these settings in place, Core Data attempts a lightweight migration when it detects the persistent store no longer matches the current model.

See Also

Data model migration

- Staged migrations

  Migrate complex data models containing changes that are incompatible with lightweight migrations.

- Manual migrations

  Migrate elaborate data models with changes that go beyond the capabilities of both lightweight and staged migrations.