INTERMEDIATE

CORE DATA

HANDS-ON CHALLENGES

Intermediate Core Data

Luke Parham

Notice of Rights

Notice of Liability

Trademarks

Challenge #2: Custom Manual Migrations

By Luke Parham

In the demo, you saw how to create a custom mapping model. Your challenge is to migrate the sample apps data to a new version yet again. Your new data model will have a generalized `Attachment` entity along with an `ImageAttachment` entity which will have image specific data such as height and width which future `Attachment` types wouldn't necessarily need.

Since you can't infer these attributes for images just by setting up a mapping model like last time, you'll be required to write your own `NSEntityMigrationPolicy` to programatically fill in these attributes for the existing images.

Creating the Mapping Model

Just like in the demo, go ahead and create a new version of the data model. In case you forgot how, just click on **UnCloudNotesDataModel.xcdatamodeld** and then use **Editor > Add Model Version...**.

Set the version name to be **UnCloudNotesDataModel v 4** and base it off of **UnCloudNotesDataModel v 3**.

Then use the **File Inspector** and set the current model version to the V4 you just created.

Open the v4 data model and add a new entity named ImageAttachment. Set the class to ImageAttachment, and the module to UnCloudNotes.

Make the following changes to ImageAttachment:
1. Set the Parent Entity to Attachment.

3. Set the Name field in the Class section to **ImageAttachment**.

4. Set the Module to **Current Product Module**.
   2. Add a required String attribute named caption.
   3. Add a required Float attribute named width.
   4. Add a required Float attribute name height.

5. Add an optional Transformable attribute named image.
6. Set the Value Transformer Name to ImageTransformer, and set the Module value to UnCloudNotes.

Next, inside the Attachment entity:
1. Delete the image attribute.
2. If a newRelationship has been automatically created, delete it.

A parent entity is similar to having a parent class, which means ImageAttachment will inherit the attributes of Attachment. When you set up the managed object subclass later, you'll see this inheritance made explicit in the code.

Fixing the Managed Object Subclasses

Next, you need to get your managed object subclasses set up to match the new data model you've set up.

First, create a new file named **ImageAttachment.swift** and replace the contents with

```
import UIKit
import CoreData

class ImageAttachment: Attachment {
  @NSManaged var image: UIImage?
  @NSManaged var width: Float
  @NSManaged var height: Float
  @NSManaged var caption: String
}
```

Next, go to **Attachment.swift** and remove the `image` property declaration since that's covered by `ImageAttachment` now.

Then, go to **Note.swift** and replace the image property with

```
var image: UIImage? {
let imageAttachment = latestAttachment as? ImageAttachment
  return imageAttachment?.image
}
```

Finally, go to **AttachPhotoViewController.swift**, find `imagePickerController(_:didFinishPickingMediaWithInfo:)` and replace the definition of `attachment` with the following.

```
let attachment = ImageAttachment(context: context)
```

Creating the Mapping Model

Next, create a mapping model for this migration and name it **UnCloudNotesMappingModel_v3_to_v4**.

Like last time, you'll see mappings for each of the entities in the destination data model. The **NoteToNote** mapping is fine as is since you won't be changing anything about that entity.

The **AttachmentToAttachment** mapping on the other hand is trying to map old attachments to new attachments, but you want old attachments to be mapped to image attachments so delete this mapping.

Finally, choose the **ImageAttachment**. Set the **Source** field to be **Attachment**. This will cause Xcode to realize what you're trying to do and rename the mapping to **AttachmentToImageAttachment** as well as fill in some of the value expressions for you. As you can see, `caption`, `width` and `height` can't be inferred from any existing information. This is where a custom migration policy comes in.

A Custom Migration Policy

Create a new Swift file called **AttachmentToImageAttachmentMigrationPolicyV3toV4** and replace its contents with the following:

```
import CoreData
import UIKit
let errorDomain = "Migration"

class AttachmentToImageAttachmentMigrationPolicyV3toV4:
NSEntityMigrationPolicy {

}
```

This `NSEntityMigrationPolicy` subclass is where you can override certain methods and hook into the migration process as its happening to help things out along the way.

Now, go back to the **v3-to-v4 mapping model file** and select the `AttachmentToImageAttachment` entity mapping. In the **Entity Mapping Inspector** on the right, find the **Custom Policy** field and fill it in with **UnCloudNotes.AttachmentToImageAttachmentMigrationPolicyV3toV4**. And yes, you do need the module name there. ;]

Ok, go back to the custom migration policy class and get ready for some fun.

First, add the following override:

```
override func createDestinationInstances(forSource sInstance:
NSManagedObject,
               in mapping: NSEntityMapping,
               manager: NSMigrationManager) throws {

}
```

This method is where Core Data takes instances of managed objects from the old

data model, and creates corresponding instances that work with the new data model.

Next, add the following helper method that performs the task of iterating over the property mappings if they are present in the migration. If they aren't that means your mappings file has been specified incorrectly.

```
func traversePropertyMappings(mapping:NSEntityMapping,
    block: (NSPropertyMapping, String) -> ()) throws {
    if let attributeMappings = mapping.attributeMappings {
      for propertyMapping in attributeMappings {
        if let destinationName = propertyMapping.name {
          block(propertyMapping, destinationName)
        } else {
          let message = "Attribute destination not configured properly"
          let userInfo = [NSLocalizedFailureReasonErrorKey: message]
          throw NSError(domain: errorDomain, code: 0, userInfo: userInfo)
        }
      }
    } else {
      let message = "No Attribute Mappings found!"
      let userInfo = [NSLocalizedFailureReasonErrorKey: message]
      throw NSError(domain: errorDomain,
                    code: 0, userInfo: userInfo)
    }
}
```

Now, go back to `createDestinationInstances(forSource:in:manager:)` and create a new instance of `ImageAttachment` as the destination object.

```
let description = NSEntityDescription.entity(forEntityName:
"ImageAttachment", in: manager.destinationContext)
let newAttachment = ImageAttachment(entity: description!, insertInto:
manager.destinationContext)
```

Next, even though it's a custom manual migration, most of the attribute migrations should be performed using the expressions you defined in the mapping model. To do so, use the traversal function from the previous step and apply the value expression to the source instance and set the result to the new destination object.

```
try traversePropertyMappings(mapping: mapping) {
  propertyMapping, destinationName in
  if let valueExpression = propertyMapping.valueExpression {
    let context: NSMutableDictionary = ["source": sInstance]
    guard let destinationValue =
      valueExpression.expressionValue(with: sInstance,
                                      context: context) else {
                                        return
    }
    newAttachment.setValue(destinationValue,
                           forKey: destinationName)
  }
}
```

After that, its time to grab an image from the source object if you can, and then use that image to set the width and height values for the destination `ImageAttachment`.

```
if let image = sInstance.value(forKey: "image") as? UIImage {
  newAttachment.setValue(image.size.width, forKey: "width")
  newAttachment.setValue(image.size.height, forKey: "height")
}
```

To fill in the caption, simply grab the note's body text and take the first 80 characters.

```
let body = sInstance.value(forKeyPath: "note.body") as? NSString ?? ""
newAttachment.setValue(body.substring(to: 80), forKey: "caption")
```

Finally, associate theE source object with the destination object you just set up. Failing to do so will mean missing data in the resulting persistent store!

```
manager.associate(sourceInstance: sInstance,
                  withDestinationInstance: newAttachment,
                  for: mapping)
```

Build, run, and cross your fingers. If all went according to plan, your old attachment objects should all be `ImageAttachments` now.

Where To Go From Here

This was quite the involved challenge, but migrations and mapping models are a very complex area in the Core Data ecosystem. Even with all you've done, what would happen if your user went from version 1 to version 4 because they skipped a few updates? This is bound to happen at some point and to learn how to deal with it, I suggest you check out Chapter 6 of our book Core Data by Tutorials.