Extracted from:

iPhone SDK Development

Building iPhone Applications


The Pragmatic Programmers
Add Book
Save
iPhone SDK Development
Bill Dudney and
Chris Adamson
Edited by Daniel H Steinberg
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
space
return
.?123

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at

http://www.pragprog.com

Printed in the United States of America.

ISBN-10: 1-934356-25-5

ISBN-13: 978-1-934356-25-8

Printed on acid-free paper.

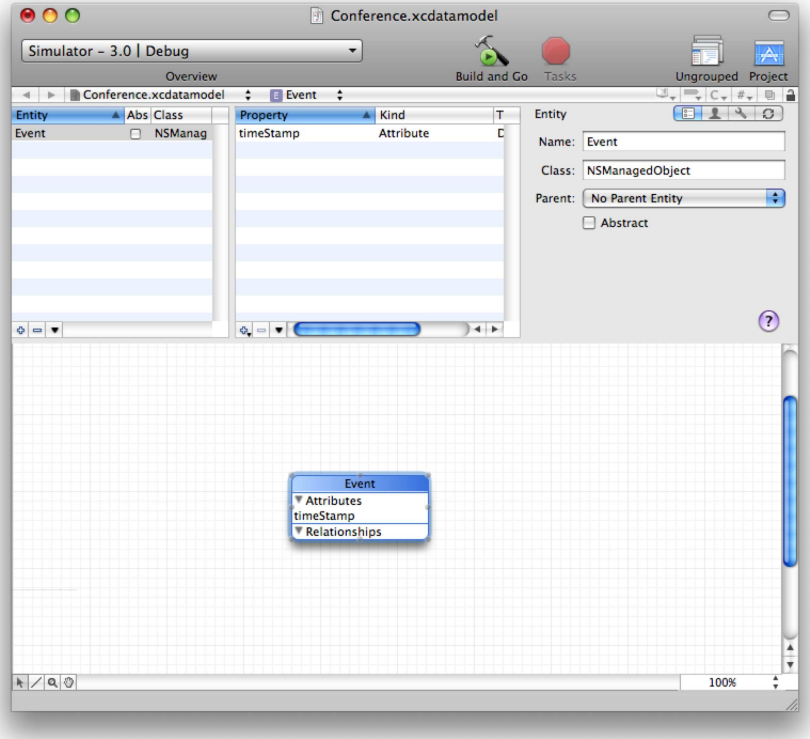
P2.0 printing, December 2009

Version: 2010-2-3

11.4 Modeling

Now that we have seen the way the template has set up Core Data for us, let's start fleshing out our application by building our managed object model.

The managed object model is a central part of how Core Data interacts with your model. Core Data uses the entity when it translates between the world of Objective-C and the world of SQLite. The model is similar to the schema of your database with a little bit of extra information about how to translate from Objective-C to SQL and back. Roughly speaking, each entity corresponds to a table in the database, and each attribute corresponds to a column in that table. On the Objective-C side, each entity corresponds to a class, and each attribute corresponds to a property on that class. There is, of course, a ton of detail behind how a MOM is used to map objects into rows, and vice versa, but you don't have to fully understand all that thanks to the great tools available in Xcode. Let's take a quick look at the UI. Double-click Conference.xcdatamodel in Xcode. You should see something that looks more or less like this screenshot:

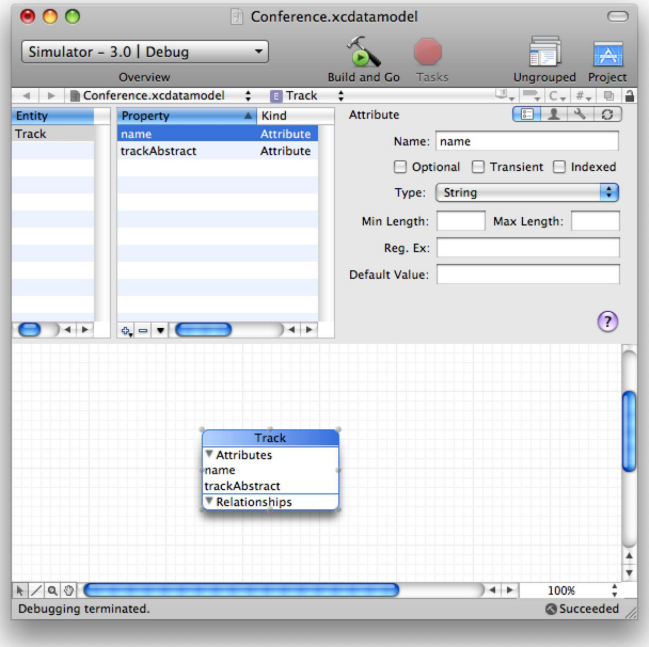


We are going to modify this template data model so it captures our Track and Session entities. Roughly speaking, we need to accomplish these tasks:

- Change the template-provided Event entity to the Track entity.
- Add the name and trackAbstract attributes and the sessions relationship to the Track entity.
- Create the Session entity.
- Add the name, sessionAbstract, and sessionID attributes and the track relationship to the Session entity.
- Generate the Track and Session classes and add them to the project.

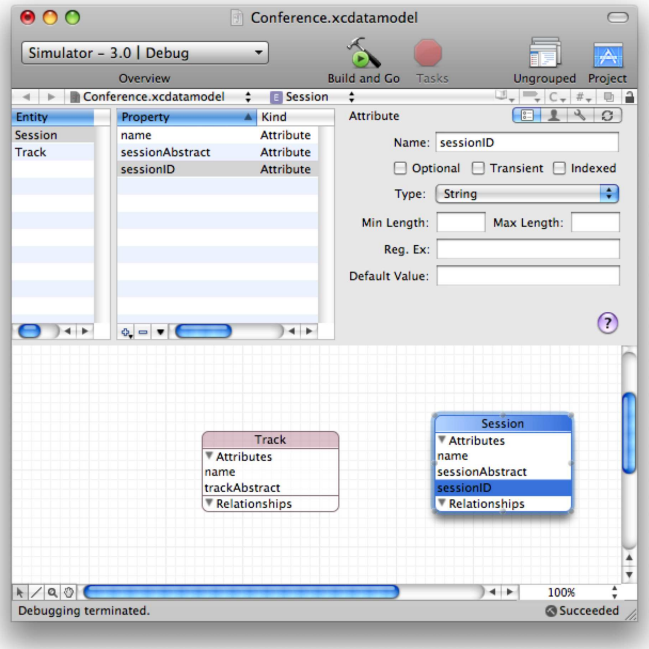
With these two entities and the relationship between them modeled, Core Data will have the information it needs to make them persistent.

Start by renaming the Event entity to Track. Select the entity; then in the Entity Attributes inspector, change the name from Event to Track. Also, change the class name from NSManagedObject to Track. Track has two attributes, name and trackAbstract. Both of them are strings with no constraints (length, regex, and so on). Add them by selecting the + button under the properties list (the list to the right of the entity list) and choosing Add Attribute. When you have added them, you should have something that looks roughly like this:

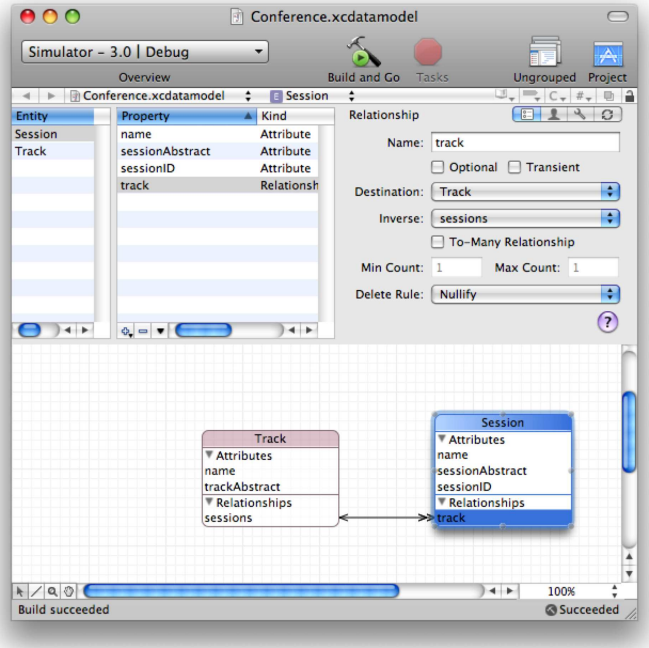


Make sure you turn off Optional for each of the attributes. Although this is not necessary, it will help make sure that no bogus data makes it into our database. As you have probably noticed, there are lots of other ways we can customize the constraints that Core Data will place on our attributes. For example, if we were to provide a regular expression (also known as a *regex*) in the Attributes inspector, Core Data would ensure that whatever value was placed in that attribute matches the regex before it saved the value. If the value fails any of the constraints, a validation error is raised. We can use the error to create a user visible/understandable message to show so the user knows what to fix. We are not going to take the time to go into detail on that, but the *Core Data documentation* [App09b] has more information on presenting errors to your users.

Now that we have the Track entity, we need to create the Session entity. Under the list of entities (where Track currently shows up on the top-left side), click the + button. Rename the new entity to Session, change its class name to Session, and add three string attributes: sessionID, name, and sessionAbstract. Again, all the Session attributes should have no constraints, and the Optional switch should be turned off. When you are done, you should have something that looks like this:

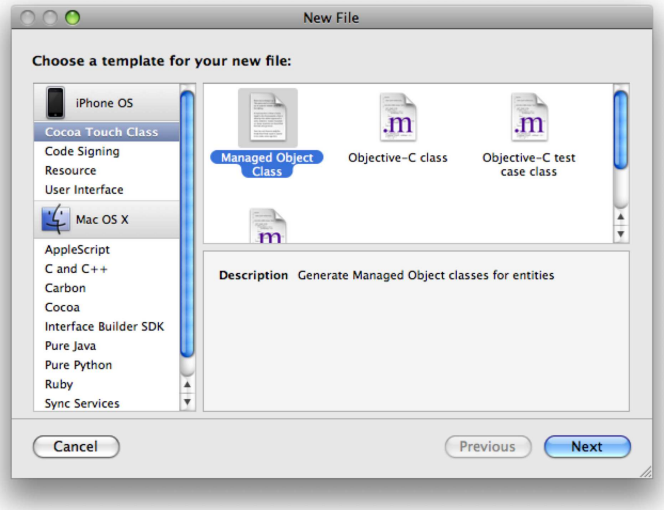


Now that we have the two entities set up, we need to create the relationship between them. Select the Track entity, and click the + button under the property list (top-left side). Choose Add Relationship, and name the new relationship sessions. Make the relationship's destination the Session entity, choose the to-many checkbox, and make sure the delete rule is Cascade. Next we need to add the inverse relationship to the Session entity. Select the Session entity, add a relationship, name it track, set its destination to Track, and choose its inverse to be sessions. Set the delete rule on the new relationship to Nullify, and turn Optional off. When you are done, the model should look something like this:

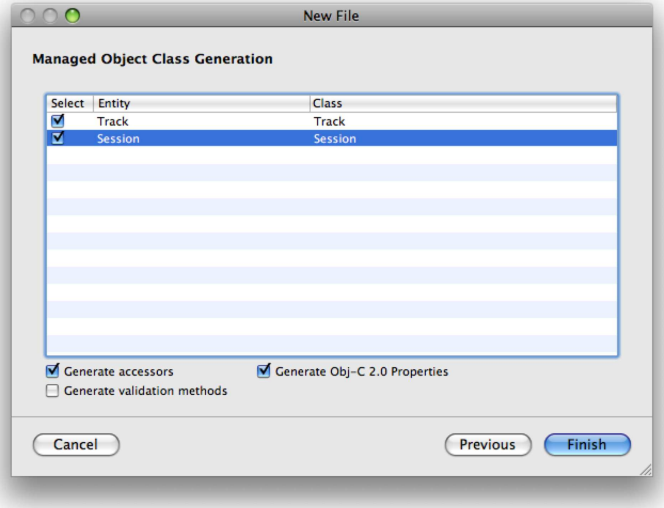


Our next step is to generate the classes that will be used to represent this model in our application. In Xcode, add a new group called Model Classes, and select it. Go back to the model, select all the entities with ⌘-a, and then choose File > New File from the menu. Select the Managed Object Class from the dialog box that pops up, as in the next screenshot.[4]

4. If you don't have the model editor as the front window when you select the File > New File menu item, you won't see the Managed Object Class option in the New File Wizard.



Click Next on the next page of the wizard. You don't need to change anything here, but make sure the Conference target is selected. On the final page of the wizard, make sure the checkbox next to Session and Track is selected so that the generator uses both our entities when it makes the classes. The page should look like this:



After you click the Finish button, you should have a header and implementation file for both the Track and Session classes.

Let's take a look at the code the generator made for us. Here is the header file for the Session class:

Download CoreData/Conference02/ModelClasses/Session.h

```
@class Track;

@interface Session :  NSManagedObject {
}

@property (nonatomic, retain) NSString * sessionID;
@property (nonatomic, retain) NSString * name;
@property (nonatomic, retain) NSString * sessionAbstract;
@property (nonatomic, retain) Track * track;

@end
```

Not much that is unexpected here. The generator created a property for each of the attributes and relationships in the model, all of them set to retain their object values. Next let's look at the implementation. Here is the code from the .m file:

Download CoreData/Conference02/ModelClasses/Session.m

```
@implementation Session

@dynamic sessionID;
@dynamic name;
@dynamic sessionAbstract;
@dynamic track;

@end
```

This file has very little substance too. The new and interesting piece is the use of @dynamic for the properties. This declaration says to the compiler that the properties will have get/set method pairs provided for them at runtime, so the compiler does not need to provide them. Core Data provides these methods for us, so we don't have to worry about them.

Let's look at the Track class next. Here is the header file:

Download CoreData/Conference02/ModelClasses/Track.h

```
@class Session;

@interface Track :  NSManagedObject {
}

@property (nonatomic, retain) NSString * trackAbstract;
@property (nonatomic, retain) NSString * name;
```

```
@property (nonatomic, retain) NSSet* sessions;

@end

@interface Track (CoreDataGeneratedAccessors)
- (void)addSessionsObject:(Session *)value;
- (void)removeSessionsObject:(Session *)value;
- (void)addSessions:(NSSet *)value;
- (void)removeSessions:(NSSet *)value;
@end
```

The top half is probably what you expected, which is a property for every attribute and relationship in the entity. The CoreDataGeneratedAccessors category might be new, however, so let's look at it in a bit more detail. An Objective-C category is a way for us to add methods to an object. On line 12, the CoreDataGeneratedAccessors category is declared for the Track class. All the methods declared in this category (until the @end) become part of the interface. Now let's look at the implementation:

Download CoreData/Conference02/ModelClasses/Track.m

```
@implementation Track

@dynamic trackAbstract;
@dynamic name;
@dynamic sessions;

@end
```

Not much here. . . the methods that are declared in the CoreDataGeneratedAccessors category are generated by Core Data at runtime (thus the name of the category). We need only the category in the header file so that we can call the methods without a compiler warning. This little bit of Objective-C magic is brought to you by Key-Value Coding (KVC). The Apple *Key-Value Coding* [App08e] documentation has a bunch of the detail. For now, though, you can ignore the detail and just know that you can call addSessionsObject: on any instance of Track.

Now that we have our model in place, Core Data will be able to persist it. Let's look at the code needed to make the Track table view work.

11.5 Track Table View

The good thing is you already know how table views work. They ask their data source for the number of sections, for the number of rows

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

iPhone SDK Development's Home Page
http://pragprog.com/titles/amiphd
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragprog.com/news
Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/amiphd.

Contact Us

Online Orders:	www.pragprog.com/catalog
Customer Service:	support@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com
Contact us:	1-800-699-PROG (+1 919 847 3884)