


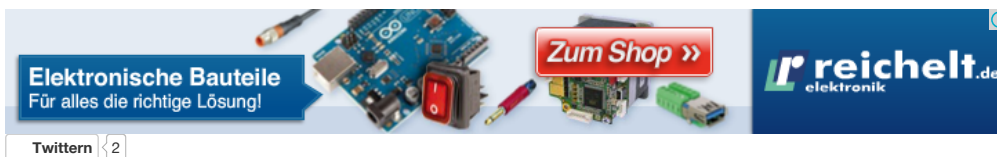
## :: Jörn Hameister ::



- 
- [Mac-Programmierung](#)
- [iPhone-Programmierung](#)
- [Java](#)
- [Publikationen](#)
- [Projekte](#)
- [Fotografie](#)
- [Blog](#)
- [Impressum](#)
- [Objective-C und Cocoa](#)
  - [Grundlagen](#)
  - [Memory Management](#)
  - [Objekt-Orientierung](#)
  - [Strings](#)
  - [Datum und Zeit](#)
  - [Arrays](#)
  - [Dictionaries](#)
  - [Dateipfade](#)
  - [Lesen von Dateien](#)
  - [plist-Dateien lesen und schreiben](#)
  - [Einlesen von XML und Zugriff mit XQuery](#)
  - [Custom Views](#)
  - [Programmatisch Views erstellen](#)
  - [Zeichnen mit appendBezierPathWith ArcFromPoint und NSBezierPath](#)
  - [Grafik](#)
  - [Table View](#)
  - [Outline View](#)
- [Beispiele](#)
  - [Vokabel-Trainer](#)
  - [Sierpinski-Dreieck](#)
- [Mac-Frameworks](#)
  - [Core Data](#)
  - [CorePlot](#)

## Simple Core Data Command Line Importer

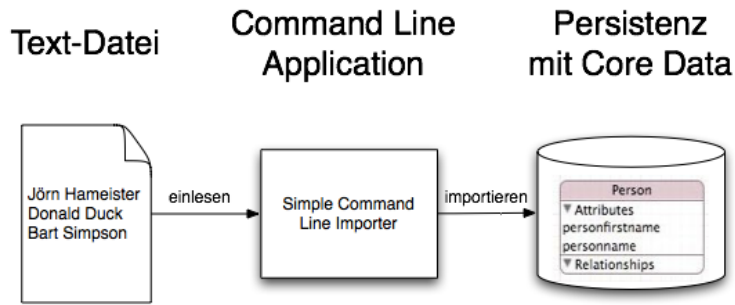
- Core Data
  - [Simple Core Data Command Line Importer](#) (-> für Xcode 3.2)
  - [Auslesen von Core Data-Daten mit NSFetchRequest](#) (-> für Xcode 3.2)
  - [Löschen von Core Data Einträgen](#) (-> für Xcode 3.2)
  - [Suchergebnisse mit NSSortDescriptor sortieren](#) (-> für Xcode 3.2)
  - [Binärdaten \(BLOBs-Binary Large Objects\) lesen und schreiben](#) (-> für Xcode 3.2)



12.03.2011

## Übersicht

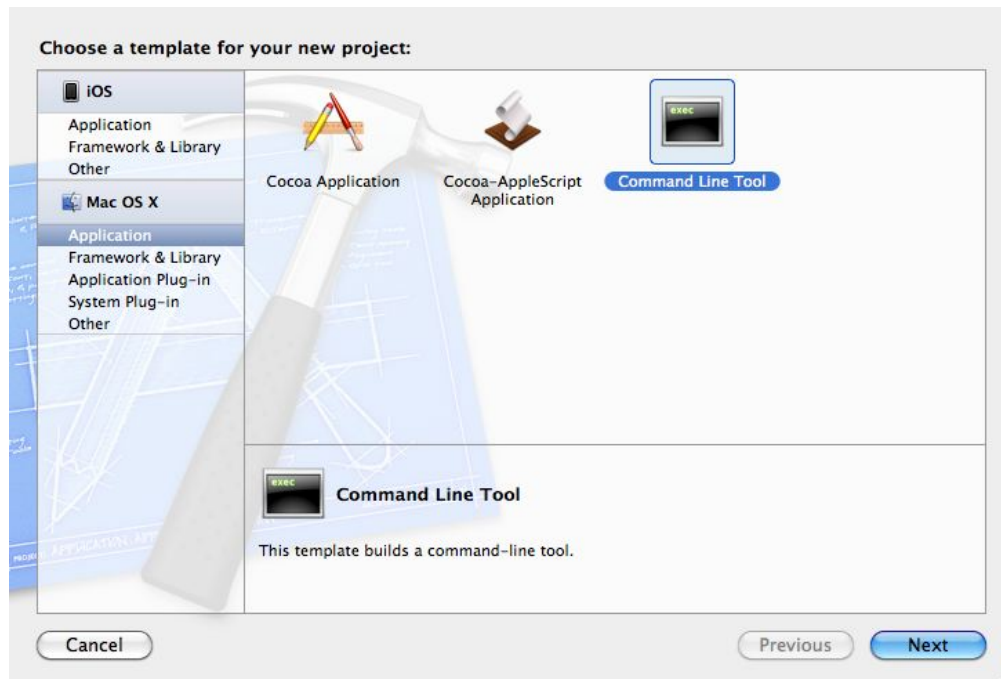
In dem Beispielprojekt wird mit Xcode 4 ein einfaches Datenmodell mit *Core Data* angelegt, das Personen mit Vor- und Nachnamen speichern kann. Danach wird eine Funktionalität ergänzt, die das Importieren von Personen aus einer Textdatei in das Datenmodell erlaubt. Dazu wird eine Textdatei mit Vor- und Nachnamen zeilenweise eingelesen und für die darin enthaltenen Personen werden *ManagedObjects* angelegt und in Core Data gespeichert.



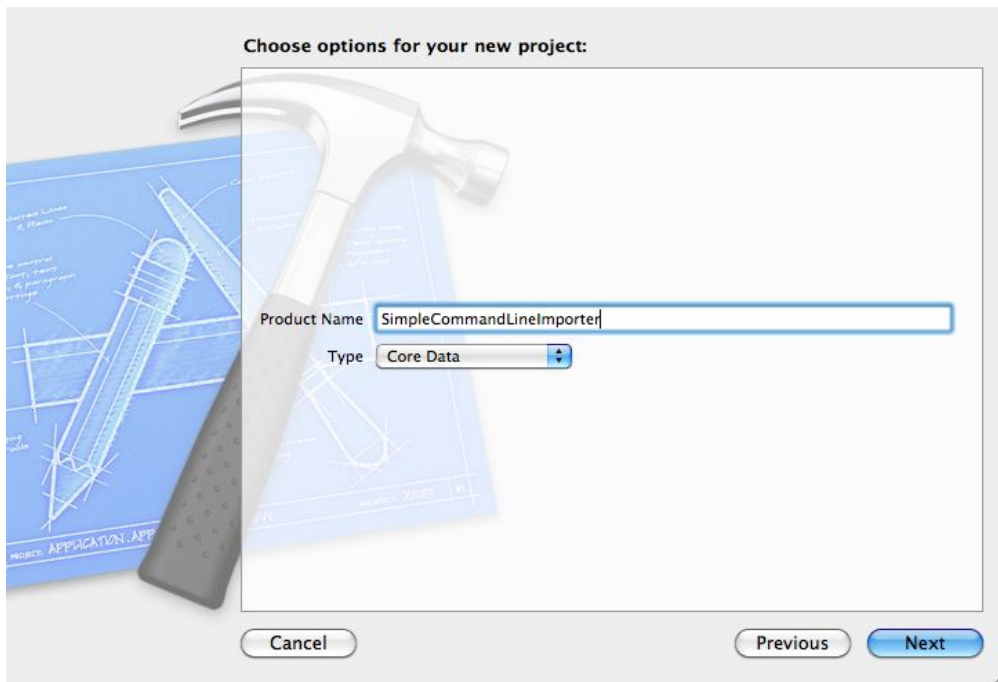
Außerdem gibt es ein paar Erklärungen zu den Methoden die Xcode beim Erstellen einer *Core Data Anwendung* anlegt. Beispielsweise wird erläutert, wie man zwischen *SQLite*, *Binary*, *in Memory* und *XML* als Speicherformat wechselt. Oder wo die Datei mit den abgespeicherten Daten zu finden ist.

## Projekt anlegen

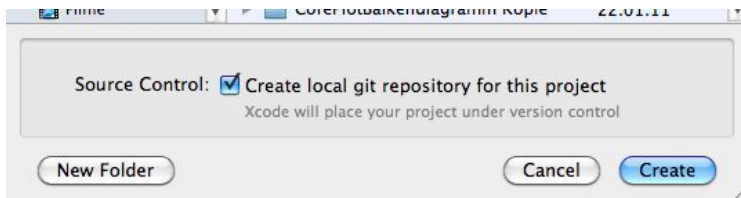
Als erstes wird ein Projekt angelegt indem im Hauptmenü *File->New Project* angeklickt wird. Im *New Project*-Dialog wird zuerst *Command Line Tool* ausgewählt und mit *Next* in den nächsten Dialog gewechselt.



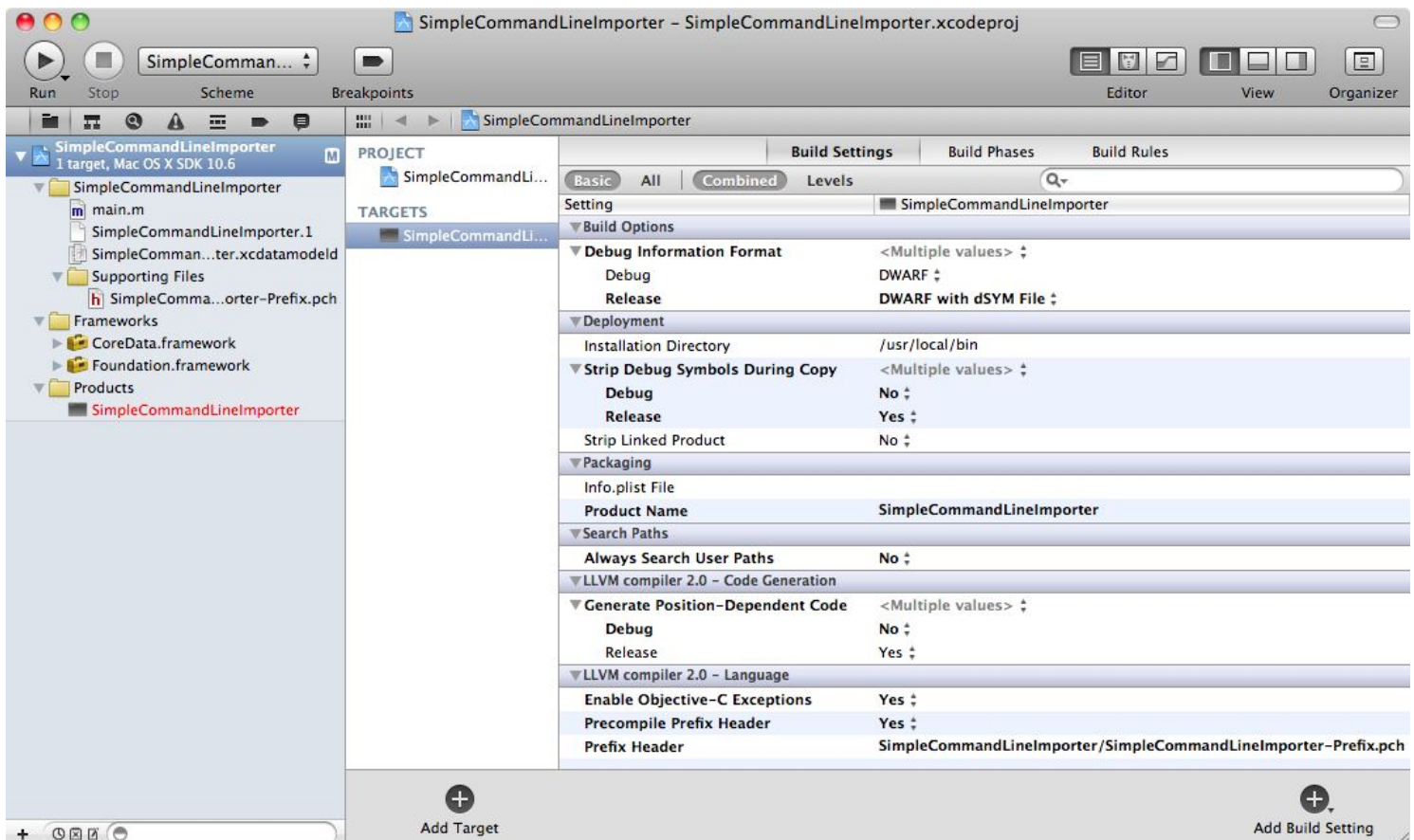
Dort wird als Type *Core Data* ausgewählt und als Name *SimpleCommandLineImporter* angegeben. Mit *Next* kommt man zum nächsten Dialog.



Abschließend wird im Datei-Dialog das Verzeichnis festgelegt, in dem das Projekt abgelegt wird. Außerdem kann in dem Dialog bei Xcode 4 ein lokales Git-Repository für das Projekt angegeben werden.



Danach sollte in Xcode ein neues Projekt angelegt worden sein, das ungefähr so aussieht:



Unter *Source* befindet sich die Hauptklasse (*main.m*) der Anwendung und unter *Model* wurde schon ein leeres Core Data Modell (*SimpleCommandLineImporter.xcdatamodeld*) angelegt.

## ManagedObjectModel

Wenn man die Datei *main.m* öffnet, findet man mehrere Methoden. Einmal die Methode `NSManagedObjectModel *managedObjectModel()` in der das *ManagedObjectModel* geladen wird.

```

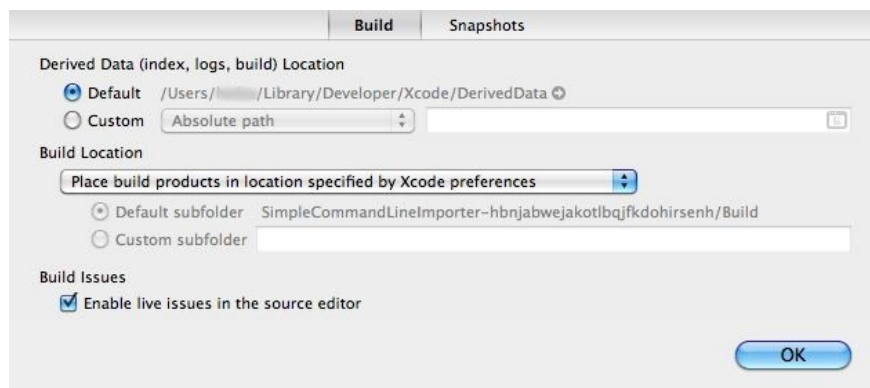
1  NSManagedObjectModel *managedObjectModel() {
2
3      static NSManagedObjectModel *model = nil;
4
5      if (model != nil) {
6          return model;
7      }
8
9      NSString *path =
10         [[[NSProcessInfo processInfo] arguments] objectAtIndex:0];
11     path =
12         [path stringByDeletingPathExtension];
13     NSURL *modelURL =
14         [NSURL fileURLWithPath:[path stringByAppendingPathExtension:@"mom"]];
15     model = [[NSManagedObjectModel alloc] initWithContentsOfURL:modelURL];
16
17     return model;
18 }
```

### Wichtig

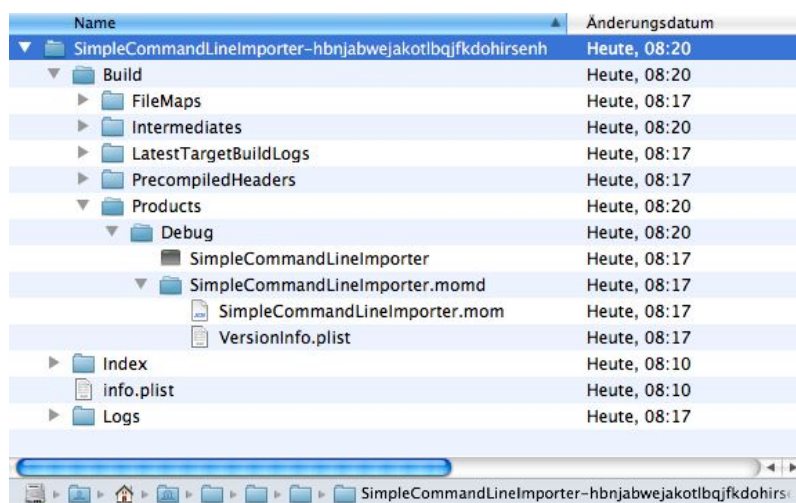
In meiner Xcode4-Version ist es so, daß ich den generierten Code der Methode *managedObjectModel* anpassen mußte. Als *pathExtension* darf nicht *mom* sondern es muß ***momd*** verwendet werden. Sonst kann die Datei nicht gefunden werden und die Applikation stürzt zur Laufzeit ab.

Diese *mom*-Datei wird beim Builden der Datei *SimpleCommandLineImporter.xcdatamodelId* erstellt. Seit Xcode4 ist diese Datei mit dem *Finder* nicht mehr ganz so einfach zu finden. Xcode 4 bietet allerdings mehrere Möglichkeiten das Build-Verzeichnis zu finden in dem sich auch die *mom*-Datei befindet.

Eine Möglichkeit ist über das Hauptmenü *Project Settings...* zu gehen.

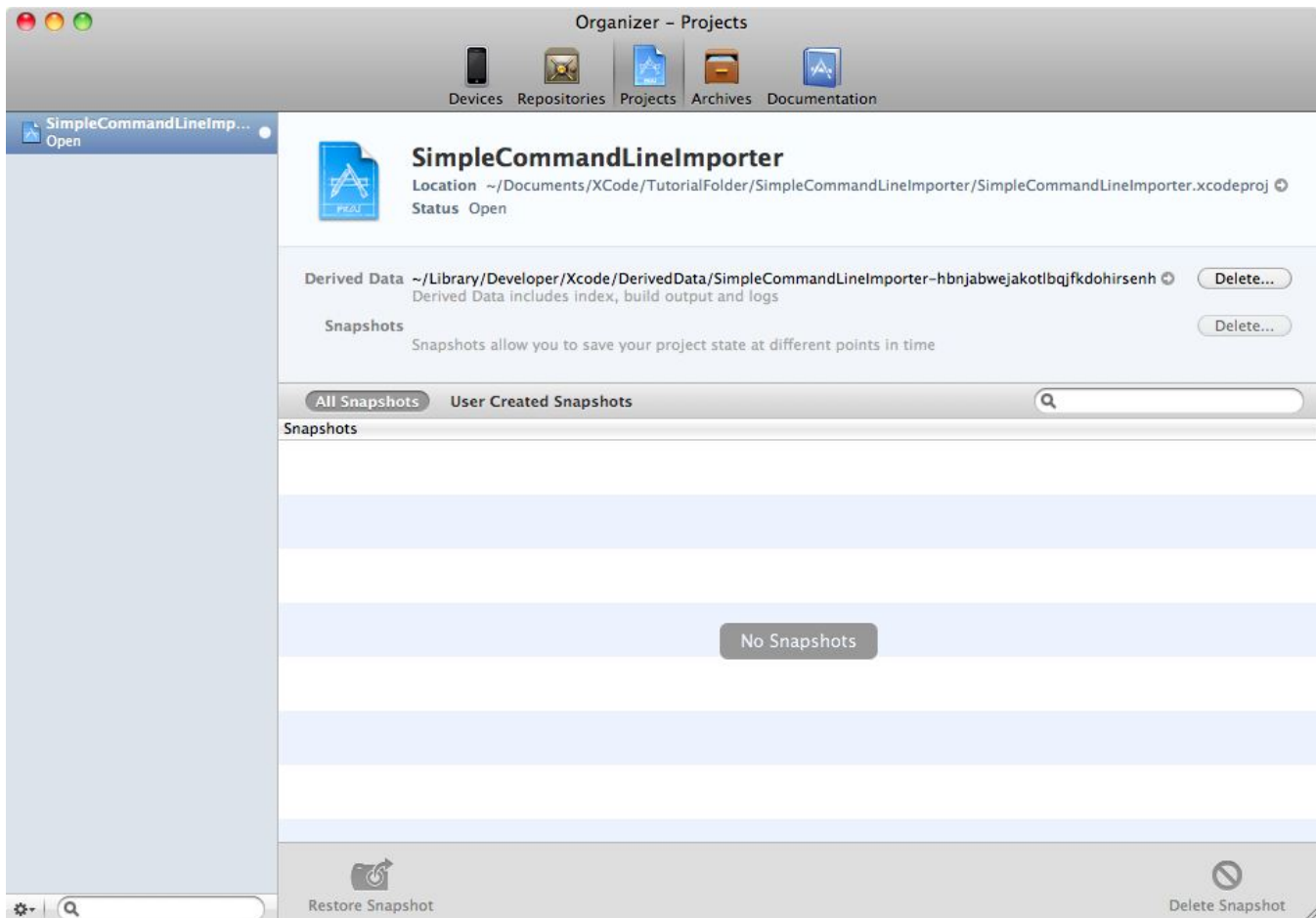


Dort findet man unter *Default* den Pfad zu *Derived Data*. Klickt man auf den kleinen grauen Pfeil am Ende der Zeile, dann öffnet sich der *Finder* und zeigt das Build-Verzeichnis, inklusive *mom*-Datei an.



Eine weitere Möglichkeit zu dem Build-Verzeichnis zu kommen ist, den *Organizer* zu öffnen. Dieser befindet sich in der Toolbar oben rechts. Nach einem Klick

darauf öffnet er sich.



In dem Dialog findet man einen Verweis auf *Derived Data*. Auch hier öffnet ein Klick auf den grauen Pfeil, den *Finder* mit dem Build-Verzeichnis.

Falls die mom-Datei einen anderen Namen hat oder sich an einer anderen Stelle befindet, dann kann die Methode `managedObjectModel` einfach angepaßt werden. Beispielsweise so:

```

1  NSManagedObjectModel *managedObjectModel() {
2
3      static NSManagedObjectModel *model = nil;
4
5      if (model != nil) {
6          return model;
7      }
8
9
10     NSString *filename =
11         [@"~/Documents/Xcode/SimpleCommandLineImporter/build/Debug/SimpleCommandLineImporter.mom" string];
12
13     model = [[NSManagedObjectModel alloc] initWithContentsOfURL:[NSURL fileURLWithPath:filename]];
14
15     return model;
16 }
    
```

## ManagedObjectContext

Die andere generierte Methode `NSManagedObjectContext *managedObjectContext()` ist für das Laden der Daten zuständig.

```

1  NSManagedObjectContext *managedObjectContext() {
2
3      static NSManagedObjectContext *context = nil;
4      if (context != nil) {
5          return context;
6      }
7
8      context = [[NSManagedObjectContext alloc] init];
9
10     NSPersistentStoreCoordinator *coordinator = [[NSPersistentStoreCoordinator alloc] initWithManagedObjc
11     [context setPersistentStoreCoordinator: coordinator];
12
13     NSString *STORE_TYPE = NSSQLiteStoreType;
    
```



```

14
15     NSString *path = [[[NSProcessInfo processInfo] arguments] objectAtIndex:0];
16     path = [path stringByDeletingPathExtension];
17     NSURL *url = [NSURL fileURLWithPath:[path stringByAppendingPathExtension:@"sqlite"]];
18
19     NSError *error;
20     NSPersistentStore *newStore = [coordinator addPersistentStoreWithType:STORE_TYPE configuration:nil UI
21
22     if (newStore == nil) {
23         NSLog(@"Store Configuration Failure\n%@",
24             ([error localizedDescription] != nil) ?
25             [error localizedDescription] : @"Unknown Error");
26     }
27
28     return context;
29 }
    
```

In ihr kann auch das Format in dem die Daten gespeichert werden sollen verändert werden. Folgende Formate stehen auf dem Mac zur Verfügung:

```

1     NSString * const NSSQLiteStoreType;
2     NSString * const NSXMLStoreType;
3     NSString * const NSBinaryStoreType;
4     NSString * const NSInMemoryStoreType;
    
```

Standardmäßig ist der erste Wert gesetzt, der angibt, daß SQLite verwendet wird. Durch den Austausch der Konstante läßt sich das Format ändern. Benutzt man beispielsweise den zweiten Wert, dann werden die Daten als XML gespeichert. Dies ist besonders für Debug-Zwecke geeignet oder wenn die XML-Datei mit einem anderen Tool weiterverarbeitet werden soll.

Wenn man die Konstante für das Format anpaßt, dann sollte auch die Dateieindung geändert werden. Falls XML gewählt wurde, sollte auch als Dateieindung xml verwendet werden:

```

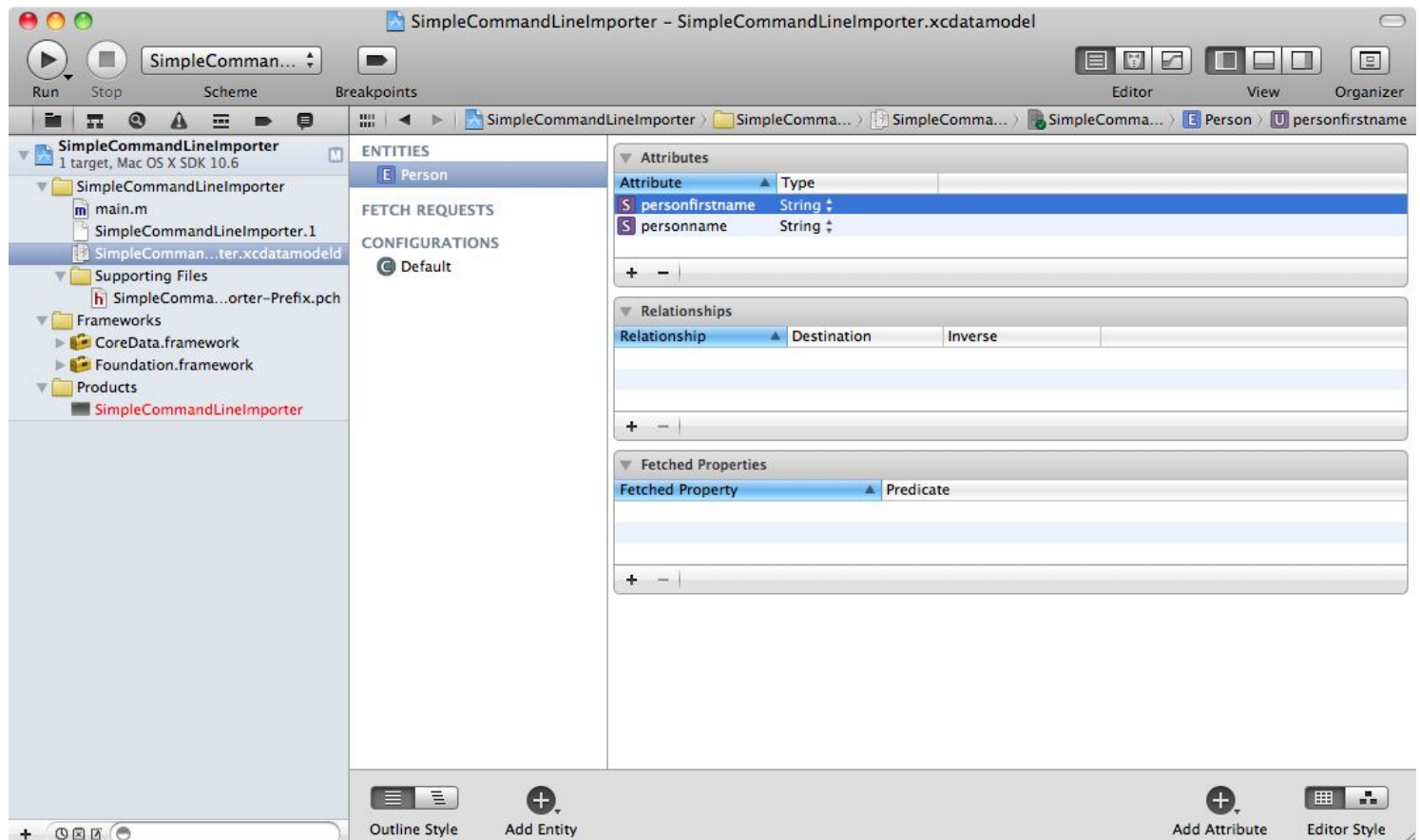
1     NSURL *url =
2     [NSURL fileURLWithPath:[path stringByAppendingPathExtension:@"xml"]];
    
```

Diese Datei liegt übrigens in dem gleichen Verzeichnis, wie die mom-Datei.

## Core Data Model erstellen

Wenn die Datei SimpleCommandLineImporter.xcdatamodelId selektiert wird, dann wechselt die Ansicht in den Core Data Design Modus.

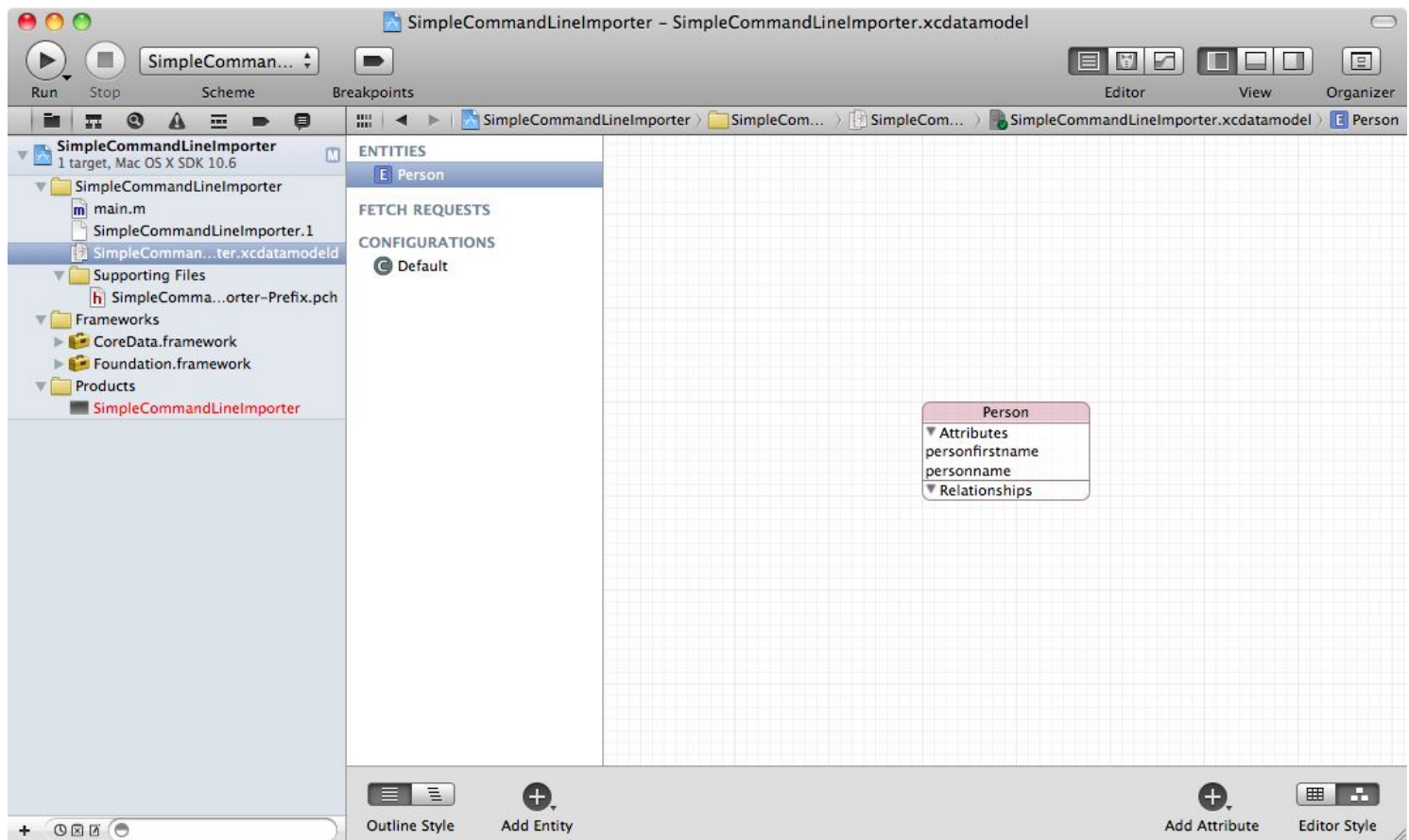
Als erstes soll jetzt eine neue Entity Person mit den Attributen Vorname und Name in dem Modell ergänzt werden.



Es gibt mehrere Möglichkeiten eine neue Entity anzulegen. Einmal über das Hauptmenü Editor->Add Entity oder über das Plus-Zeichen unten (Add Entity).

Nach dem Anlegen einer neuen Entity läßt sich ein Name wählen. In dem Beispiel wird Person verwendet. Wenn die Entity Person selektiert ist, lassen sich über das Plus-Zeichen (*Add Attribute*) zusätzliche Attribute hinzufügen. Dies kann aber auch über das Hauptmenü gemacht werden, indem *Editor->Add Attribute* ausgewählt wird. In dem Beispiel werden zwei Attribute hinzugefügt. Beide sind vom Type *String*.

Seit Xcode 4 kann zwischen zwei *Editor-Styles* gewechselt werden. Unten rechts läßt sich zwischen dem oben gezeigten *Style* und einer Modell-Ansicht wechseln.



## Einlesen von Textdateien

Die Personen, die in Core Data gespeichert werden sollen, liegen in einer Textdatei (Personen.txt) vor. In dem Beispiel wird davon ausgegangen, daß sich die Datei im Verzeichnis *Dokumente* des aktuell angemeldeten Benutzers befindet. Sie hat folgendes Format:

```
Jörn Hameister
Donald Duck
Bart Simpson
```

Eine genauere Erklärung, wie man so eine Datei einliest findet man auf der Seite [Einlesen von Dateien mit Objective-C](#)

In der Datei `SimpleCommandLineImporter.m` wird also eine Methode zum Einlesen dieser Textdatei ergänzt. Sie sieht folgendermaßen aus:

```
1  NSArray* readPersons() {
2      //Zeilenweise Textdatei einlesen:
3      NSString *filename =
4      [@"~/Documents/Personen.txt" stringByResolvingSymlinksInPath];
5      NSError *error = nil;
6
7      NSString *filecontent =
8      [[NSString alloc] initWithContentsOfFile:filename
9                                     encoding:NSUTF8StringEncoding
10                                    error:&error];
11
12      if(error!=nil) {
13          NSLog(@"Cannot read file: %@", filename);
14          return nil;
15      }
16
17      //Parse filecontent
18      NSArray *lines = [filecontent componentsSeparatedByString:@"\n"];
19      for(id names in lines) {
20          NSLog(@"%@", names);
21      }
22
23      return lines;
24  }
```

Die Personen werden zeilenweise eingelesen und in einem NSArray als NSString gespeichert.

## Importieren von Personen

Zum Importieren wird in der main-Methode der Datei `simpleCommandLineImporter.m` folgender Quellcode an der Stelle `// Custom code here...` ergänzt:

```
1  // Read persons
2  NSArray* persons = readPersons();
3
4  for(id person in persons) {
5      NSArray *firstnameLastname = [person componentsSeparatedByString:@" "];
6
7      // Add a new Person
8      NSManagedObject *personMO = [NSEntityDescription insertNewObjectForEntityForName:@"Person" inManagedC
9      [personMO setValue:[firstnameLastname objectAtIndex:0] forKey:@"personfirstname"];
10     [personMO setValue:[firstnameLastname objectAtIndex:1] forKey:@"personname"];
11 }
```

Zuerst wird die Methode zum Einlesen der Personen aufgerufen. Danach wird jeder Personen-String (z.B. *Jörn Hameister*) am Leerzeichen aufgesplittet und in einem NSArray abgelegt. Dadurch wurde der Vorname vom Nachnamen getrennt.

Für die Personen, welche in Core Data gespeichert werden sollen, wird ein *NSManagedObject* erzeugt. Als Entity-Name wird `person` angegeben. Anschließend werden die Attribute mit Werten gefüllt. Zuerst der Vorname, dann der Nachname. Es wird jeweils als `key` der Attributname verwendet.

In dem Beispiel wird der Einfachheit halber davon ausgegangen, daß eine Person immer einen Vor- und Nachnamen hat.

Nach dem Kompilieren und Starten werden die Daten aus der Datei eingelesen und in Core Data abgelegt. Wie die Daten ausgelesen werden, wird in dem Artikel: [Auslesen mit NSFetchRequest](#) erklärt.



- [Impressum](#)
- [Kontakt](#)