Chapter 23
Core Date

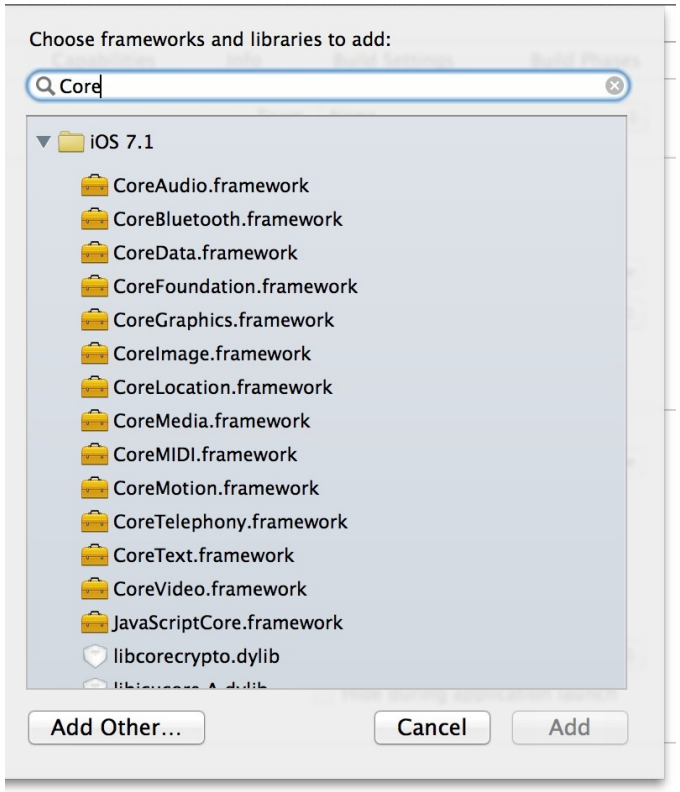
- Object-Relational Mapping
- Moving Homeowner to Core Data

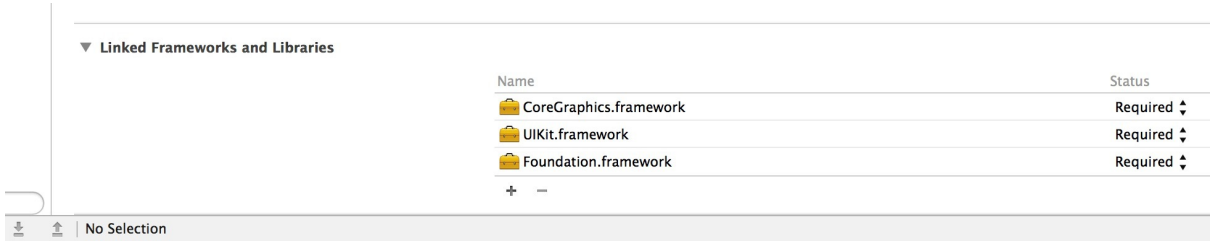
Core Data

- Core Data is a framework that provides object-relational mapping.
- Core Data can turn Objective-C objects into data that is stored in a SQLite database file and vice-versa.
- SQLite is a relational database that is stored in a single file.
- SQLite is the library that manages the database file
- SQLite is not a full-fledged relational database server
- Core Data gives us the ability to fetch and store data in a relational database without having to know SQL.
- We do have to understand a bit about how relational databases work.

Moving Homeowner to Core Data

- Copy the latest version of Homeowner application to a new location
- Core Data is a framework that needs to be added to your application





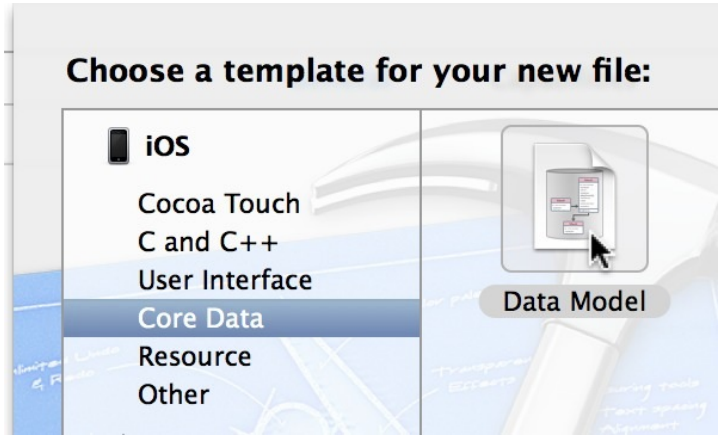
The Model File

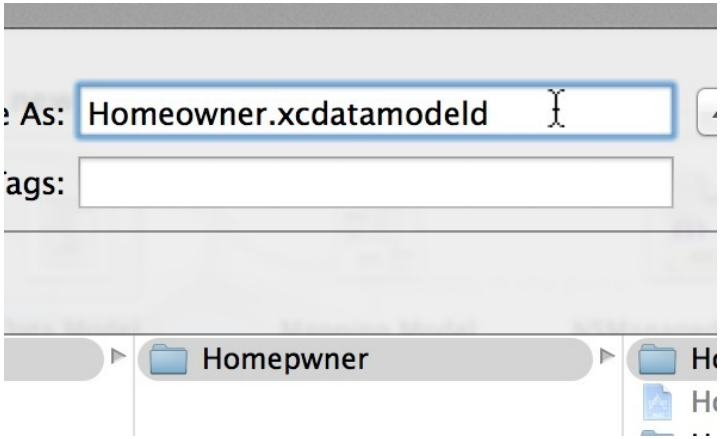
- In relational database we use the terms entity and attribute as the conceptual names for the information that we store in tables and columns, Objective-C uses the Entity/Attribute and Relation terminology too.
- A Core Date model file is the description of every entity with all its attributes.
- Entities in Core Data, as in relational databases, can be related to each other.
- Core Data supports one-to-one and one-to-many:

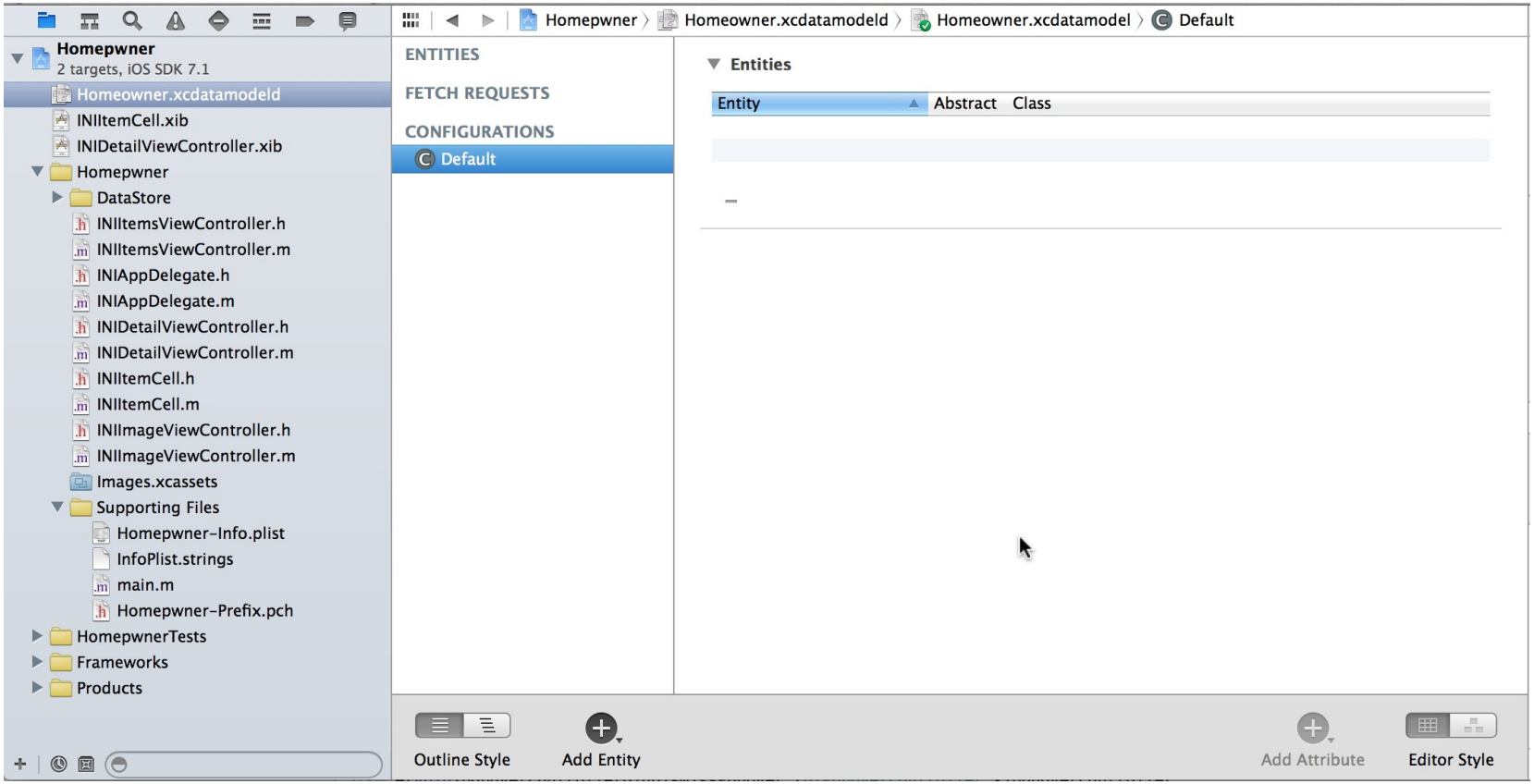
  every instance in INIItem will map to one instance of INIAssetType

  every instance in INIAssetType will map to many instances of INIItem, that is to an instance on NSSet of INIItems

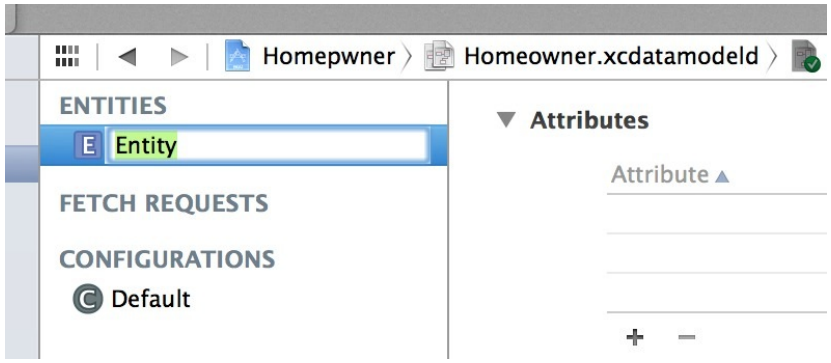
Creating Model Files in Core Data

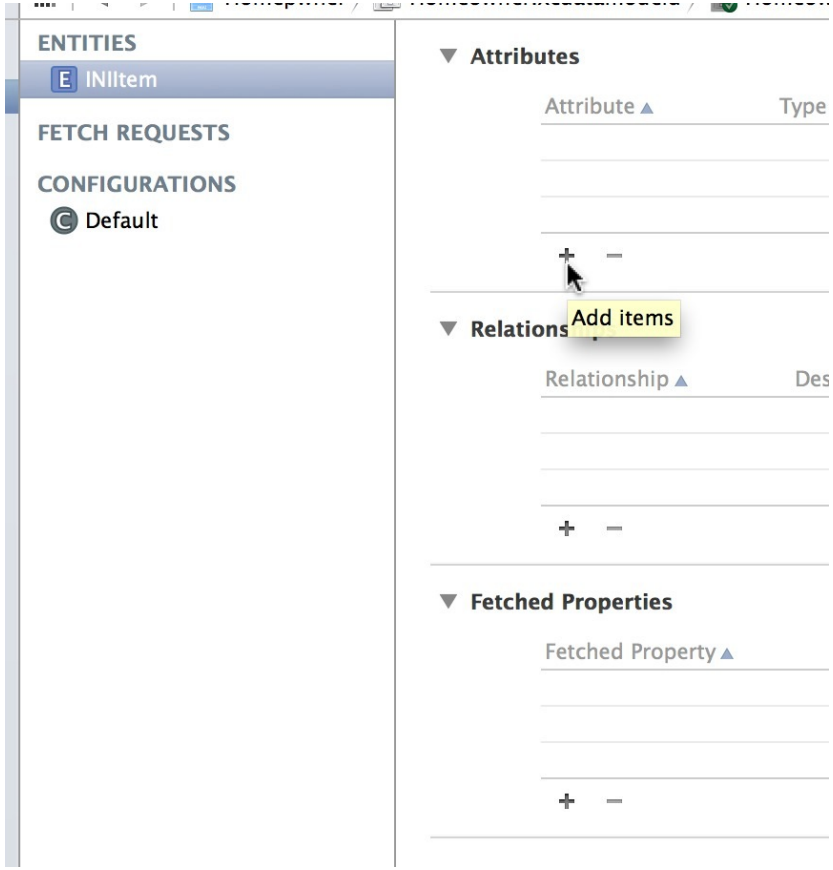


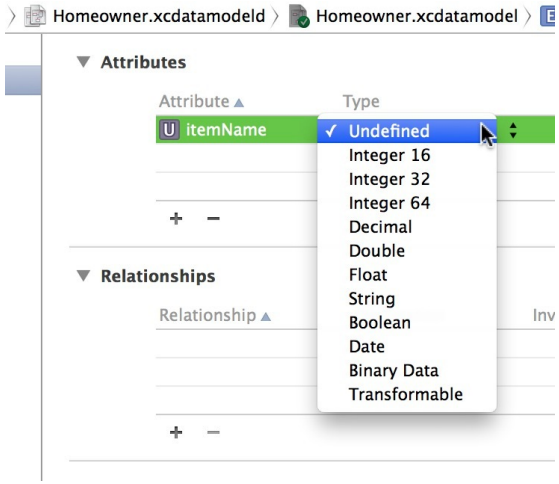


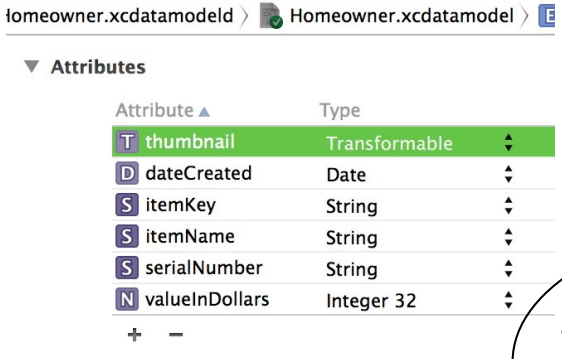


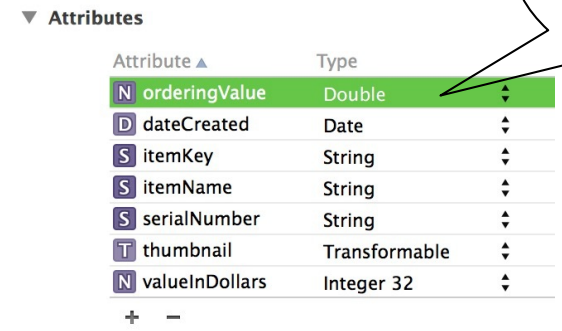
The model file INIItem for Homeowner







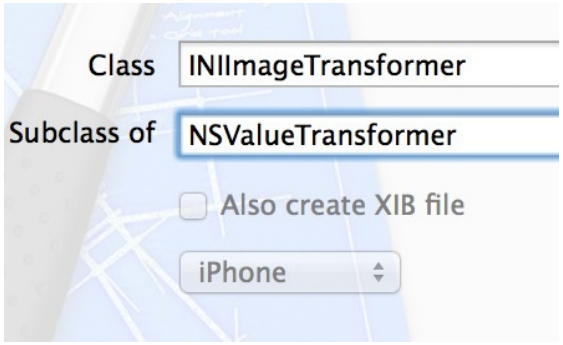




orderingValue used to keep track of the wa items are ordered in th table

Handling The thumbnail Attribute

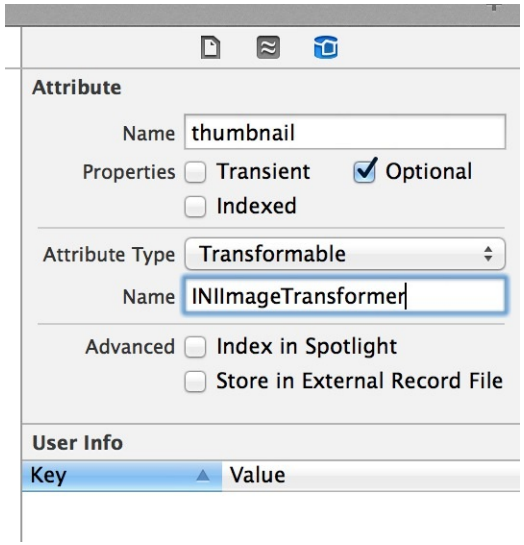
- Core Data can only store certain data types in its store.
- UIImage is not one of these types that is why we declared the thumbnail as transformable.
- With a transformable attribute, Core Data will convert the object into NSData when saving, and convert the NSData back into the original object when loading it from the file system.
- For Core Data to do this for the thumbnail attribute we need to supply it with an NSValueTransformer subclass to handles these conversions.

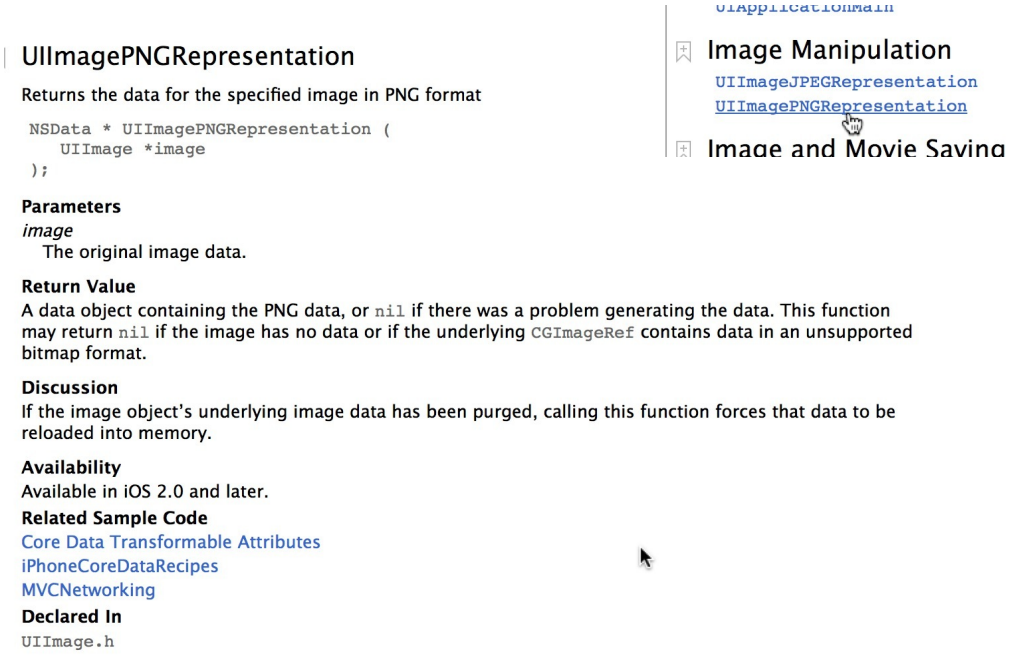



```
#import "INIImageTransformer.h"

@implementation INIImageTransformer

+(Class)transformedValueClass
{
    return [NSData class];
}

-(id)transformedValue:(id)value
{
    if (!value){
        return nil;
    }
    if ([value isKindOfClass:[NSData class]]){
        return value;
    }
    return UIImagePNGRepresentation(value);
}

-(id)reverseTransformedValue:(id)value
{
    return [UIImage imageWithData: value];
}

@end
```

```objc
#import "INIImageTransformer.h"

@implementation INIImageTransformer

+(Class)transformedValueClass
{
    return [NSData class];
}

-(id)transformedValue:(id)value
{
    if (!value){
        return nil;
    }
    if ([value isKindOfClass:[NSData class]]){
        return value;
    }
    return UIImagePNGRepresentation(value);
}

-(id)reverseTransformedValue:(id)value
{
    return [UIImage imageWithData: value];
}

@end
```

The class method transformedValueClass tells the transformer what type of object it will receive from the transformedValue: method, namely NSData.
The transformedValue: method will be called when your transformable variable is to be saved to the file system. It expects a UIImage as input and it will return an instance of NSData that can be written to the filesystem.
reverseTransformedValue: method is called when the thumbnail data is loaded from the file system, and your implementation will create the UIImage from the NSData that was stored.





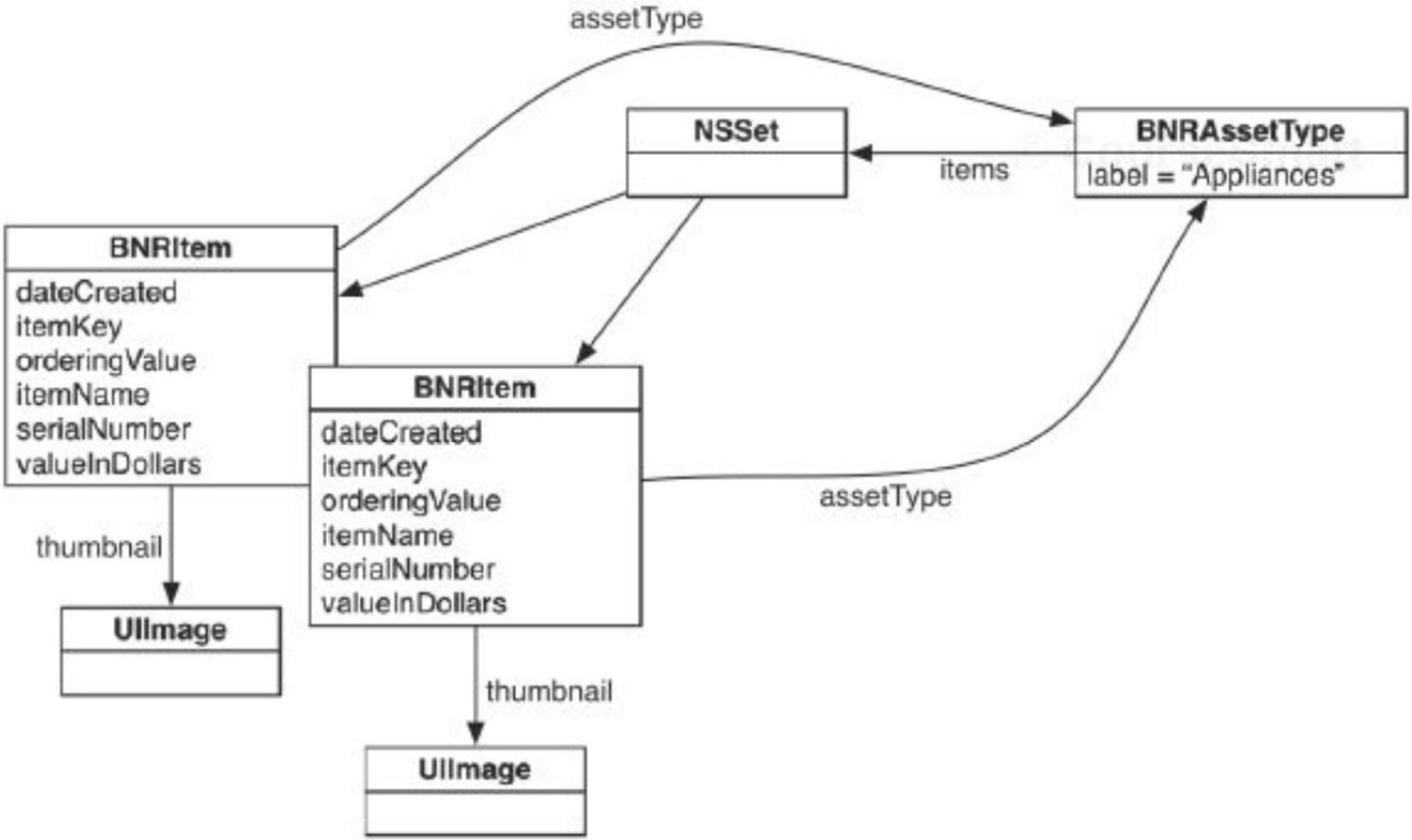
Entity Relationships

- One of the benefits to using Core Data is that entities can be related to one another.
- We will add a new entity called INIAssetType that describes a category of items.
- INIAssetType will be an entity in the model file, and each row of that table will be mapped to an Objective- C object at runtime.
- We need to establish relationships between INIAssetType and INIItem.
- Relationships between entities are represented by pointers between objects.
- There are two kinds of relationships:

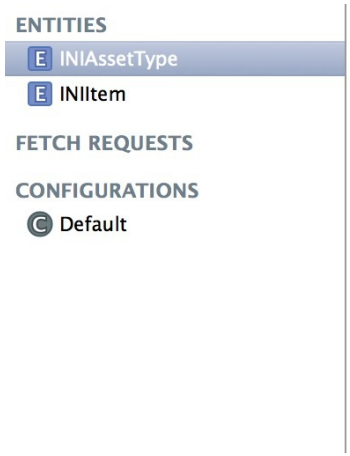
  to-one, each instance of this entity will have a pointer to an instance in the entity it has a relationship to. An item will have a relationship to one type (painting points to art)

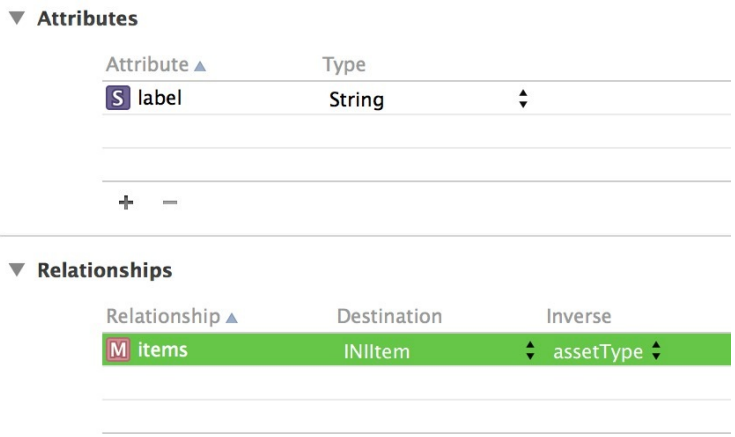
  to-many, each instance of this entity has a pointer to an NSSet that contains the instances of the entity that it has a relationship with (art points to a set of all paintings, sculptures …)

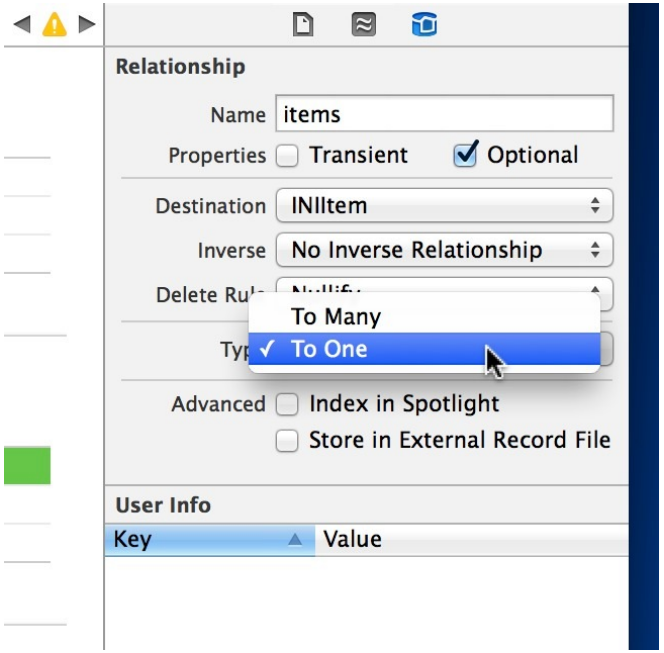
Entities in Homeowner

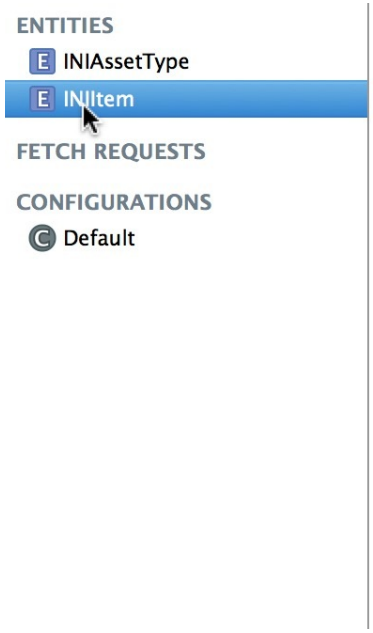


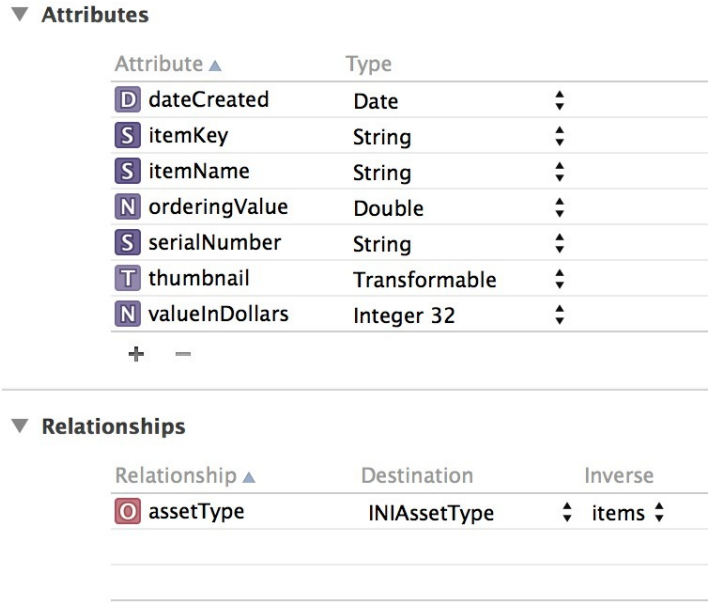
Assigning Relationships

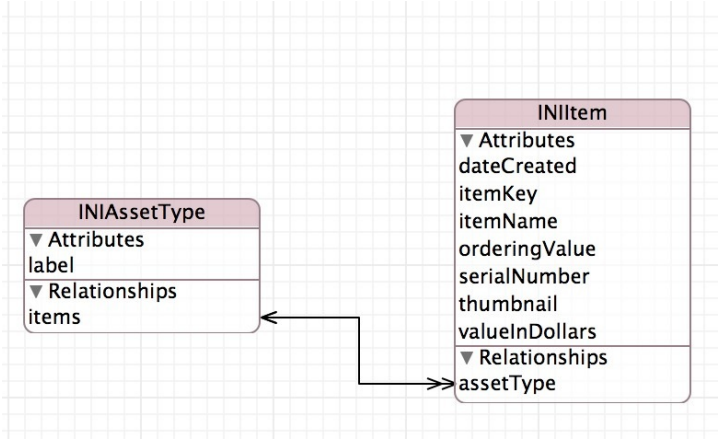






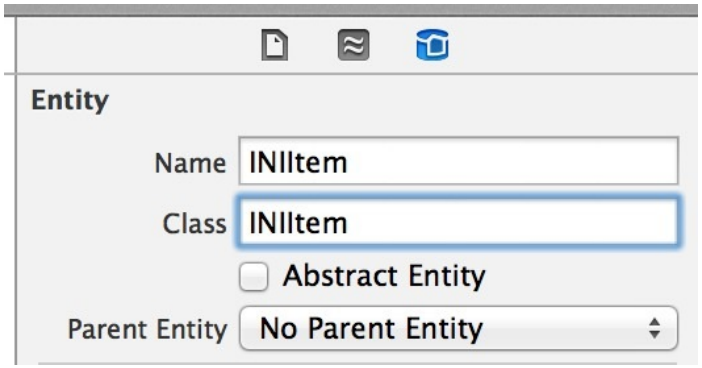


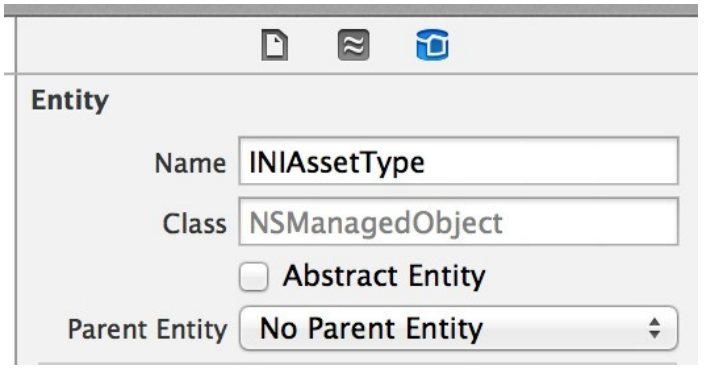




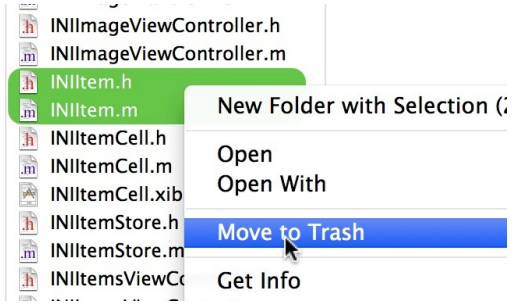
NSManagedObject

- When Core Data fetches objects from a table, it stores the information in an instance of NSManagedObject.
- NSManagedObject is a smart class that knows how to cooperate with Core Data and it holds a key-value pair for every property (attribute or relationship) in the entity.
- If you want to add additional behavior to what is retrieved by Core Data, you should subclass NSManagedObject and have the returned object by Core Data be an instance of this subclass.
- In our example, we let Xcode generate a subclass of NSManagedObject as our new INIItem class then we add our desired behavior to this new class.

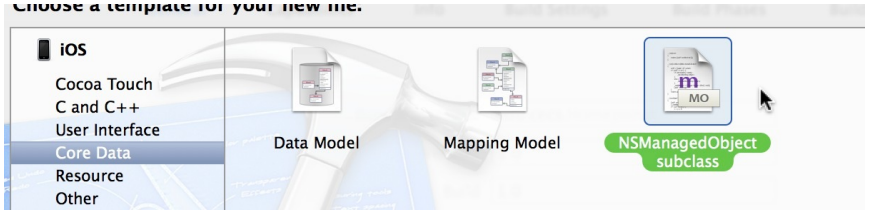


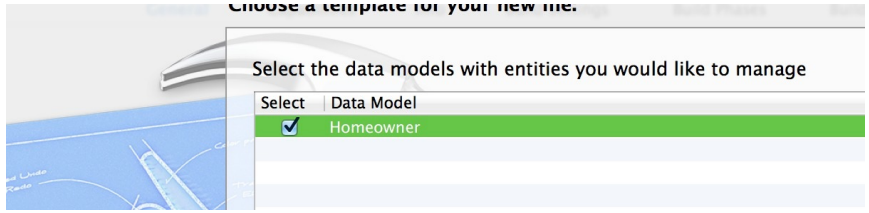


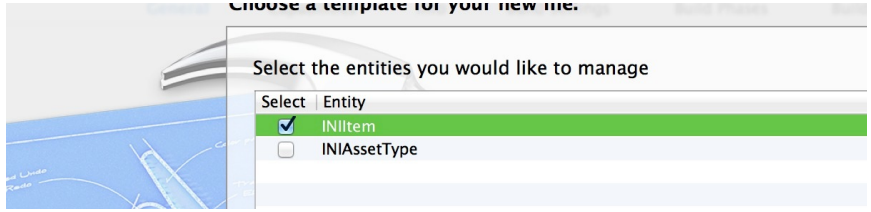
NSManagedObject and subclasses




```objc
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>


@interface INIItem : NSManagedObject

@property (nonatomic, retain) NSDate    *dateCreated;
@property (nonatomic, retain) NSString  *itemKey;
@property (nonatomic, retain) NSString  *itemName;
@property (nonatomic)         double    orderingValue;
@property (nonatomic, retain) NSString  *serialNumber;
@property (nonatomic, strong) UIImage   *thumbnail;
@property (nonatomic)         int       valueInDollars;
@property (nonatomic, retain) NSManagedObject *assetType;

- (void) setThumbnailFromImage:(UIImage *) image;

@end
```








```objc
#import "INIItem.h"


@implementation INIItem

@dynamic dateCreated;
@dynamic itemKey;
@dynamic itemName;
@dynamic orderingValue;
@dynamic serialNumber;
@dynamic thumbnail;
@dynamic valueInDollars;
@dynamic assetType;

-(void)setThumbnailFromImage:(UIImage *)image
{
    CGSize origImageSize = image.size;
    //The rectangle of the thumbnail
    CGRect newRect = CGRectMake(0, 0, 40, 40);

    //Figure out a scaling ratio to make sure we maintain the same
    float ratio = MAX(newRect.size.width /origImageSize.width, new
```

awakeFromInsert

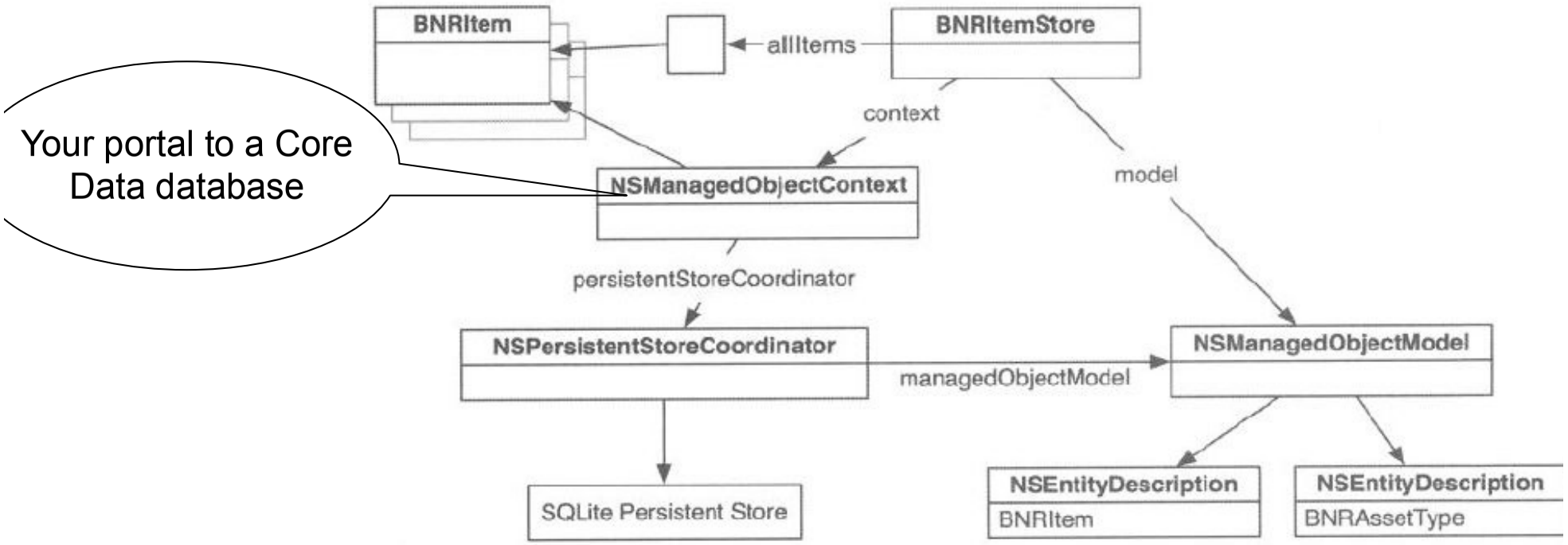
- Of course, when you first launch an application, there are no saved items or asset types.
- When the user creates a new INIItem instance, it will be added to the database.
- When objects are added to the database, they are sent the message awakeFromInsert.
- In awakeFromInsert is where you will set the dateCreated and itemKey properties of a INIItem.

```
59
60  -(void)awakeFromInsert
61  {
62      [super awakeFromInsert];
63      self.dateCreated = [NSDate date];
64      //Create an NSUUID object -and get its string representation
65      NSUUID *uuid = [[NSUUID alloc] init];
66      NSString *key = [uuid UUIDString];
67      self.itemKey = key;
68  }
```

Updating INIItemStore

- The portal through which you talk to the database is the NSManagedObjectContext.
- The NSManagedObjectContext uses a persistence store coordinator named NSPersistentStoreCoordinator.
- We request the persistent store coordinator to open a SQLite database at a particular filename (connect to the database).
- The persistent store coordinator uses the model file in the form of an instance of NSManagedObjectModel.

Talking to the database in Core Date



A context without a coordinator is not fully functional as it cannot access a model except through a coordinator.

The coordinator needs to know:

- The entities, attributes and relations which are in the Model
- Where to store the data in the persistence store

Updating INIItemStore

```
#import "INIItemStore.h"
#import "INIImageStore.h"
#import <CoreData/CoreData.h>

@interface INIItemStore ()

@property (nonatomic) NSMutableArray *privateItems;
@property (nonatomic, strong) NSMutableArray *allAssetTypes;
@property (nonatomic, strong) NSManagedObjectContext *context;
@property (nonatomic, strong) NSManagedObjectModel *model;

@end
```

```
-(NSString *)itemArchivePath
{

    NSArray *documentDirectories = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                                                       NSUserDomainMask,
                                                                       YES);


    NSString *documentDirectory = [documentDirectories firstObject];
    return [documentDirectory stringByAppendingPathComponent:@"store.data"];
}
```

```
@im
```

```
    // Here is the real (secret) initializer
- (instancetype) initPrivate
{
    self = [super init];
    if ( self) {
        NSString *path = [self itemArchivePath];
        _privateItems = [NSKeyedUnarchiver unarchiveObjectWithFile: path];

        // If the array hadn't been saved previously, create a new empty one
        if (!_privateItems) {
            _privateItems = [[NSMutableArray alloc] init];
        }
    }
    return self;
}
```

```
    // Here is the real (secret) initializer
- (instancetype) initPrivate
{
    self = [super init];
    if ( self) {
        //Read in Homepwner.xcdatamodeld
        _model = [NSManagedObjectModel mergedModelFromBundles: nil];
        NSPersistentStoreCoordinator *psc = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel:_model];
        //Where does the SQLite file go?
        NSString *path = self.itemArchivePath;
        NSURL *storeURL = [NSURL fileURLWithPath: path];
        NSError *error;
        if (![psc addPersistentStoreWithType: NSSQLiteStoreType
                                configuration: nil
                                          URL: storeURL
                                      options: nil
                                        error:&error]) {
            [NSException raise:@"Open Failure"
                        format:@"Reason: %@",[error localizedDescription]];
        }
        //Create the managed object context
        _context = [[NSManagedObjectContext alloc] init];
        _context.persistentStoreCoordinator = psc;
        [self loadAllItems];
    }
    return self;
}
```

```
- (BOOL) saveChanges
{
    NSString *path = [self itemArchivePath];
    // Returns YES on success
    return [NSKeyedArchiver archiveRootObject: self.privateItems toFile: path];
}
```

```
- (BOOL) saveChanges
{
    NSError *error;
    BOOL successful = [self.context save:&error];
    if (!successful){
        NSLog(@"Error saving: %@", [error localizedDescription]);
    }
    return successful;
}
```

NSFetchRequest and NSPredicate

- In this application we will fetch all of the items in store.data the first time you need them.
- To get objects back from the NSManagedObjectContext we must prepare and execute an NSFetchRequest (SQL Select).
- After a fetch request is executed, we will get an array of all the objects that match the parameters of that request.
- A fetch request needs an entity description that defines which entity you want to get objects from.
- To fetch INIItem instances, we specify the INIItem entity.
- We can also set the request's sort descriptors to specify the order of the objects in the array.
- A sort descriptor has a key that maps to an attribute of the entity and a BOOL that indicates if the order should be ascending or descending.

Implementing the NSFetchRequest and NSPredicate

```
-(void)loadAllItems
{
    if (!self.privateItems){
        NSFetchRequest *request = [[NSFetchRequest alloc] init];
        NSEntityDescription *e = [NSEntityDescription entityForName:@"INIItem"inManagedObjectContext: self.context];
        request.entity = e;
        NSSortDescriptor *sd = [NSSortDescriptor sortDescriptorWithKey:@"orderingValue"ascending: YES];
        request.sortDescriptors = @[sd];
        NSError *error;
        NSArray *result = [self.context executeFetchRequest: request error:& error];
        if (! result){
            [NSException raise:@"Fetch failed"
                        format:@"Reason:%@",[error localizedDescription]];
        }
        self.privateItems = [[NSMutableArray alloc] initWithArray: result];
    }
}
```

- A predicate contains a condition that can be true or false.

  NSPredicate * p = [NSPredicate predicateWithFormat:@" valueInDollars > 50"];

  [request setPredicate: p];
- The format string for a predicate can be very long and complex.
- Predicates can also be used to filter the contents of an array

  NSArray * expensiveStuff = [allItems filteredArrayUsingPredicate: p];

Adding and Deleting Items

```
- (INIItem *) createItem
{
    double order;
    if ([self.allItems count] == 0){
        order = 1.0;
    } else {
        order = [[self.privateItems lastObject] orderingValue] + 1.0;
    }
    NSLog(@"Adding after %lu items, order = %.2f", (unsigned long)[self.privateItems count], order);

    INIItem *item = [NSEntityDescription insertNewObjectForEntityForName:@"INIItem"
                                                   inManagedObjectContext: self.context];
    item.orderingValue = order;
    [self.privateItems addObject: item];
    return item;
}
```

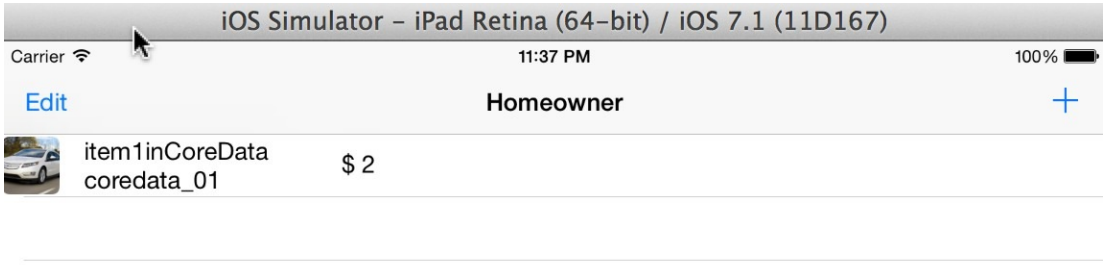
```
- (void) removeItem:(INIItem *) item
{
    NSString *key = item.itemKey;
    [[INIImageStore sharedStore] deleteImageForKey: key];
    [self. context deleteObject: item];
    [self.privateItems removeObjectIdenticalTo: item];
}
```

```
(void) moveItemAtIndex:(NSUInteger) fromIndex toIndex:(NSUInteger) toIndex

   if ( fromIndex == toIndex) {
       return;
   }
   INIItem *item = self.privateItems[fromIndex];
   [self.privateItems removeObjectAtIndex: fromIndex];
   [self.privateItems insertObject: item atIndex: toIndex];

   //Computing a new orderValue for the object that was moved

   double lowerBound = 0.0;
   //Is there an object before it in the array?
   if (toIndex > 0){
       lowerBound = [self.privateItems[(toIndex -1)] orderingValue];
   } else {
       lowerBound = [self.privateItems[1] orderingValue] -2.0;
   }

   double upperBound = 0.0;
   //Is there an object after it in the array?
   if (toIndex < [self.privateItems count] -1){
       upperBound = [self.privateItems[(toIndex + 1)] orderingValue];
   } else {
       upperBound = [self.privateItems[(toIndex -1)] orderingValue] + 2.0;
   }
   double newOrderValue = (lowerBound + upperBound)/2.0;
   NSLog(@"moving to order % f", newOrderValue);
   item.orderingValue = newOrderValue;
```

The orderingValue is a double to make calculating new order value easier.

Running The App


```
2014-07-15 23:34:45.293 Homepwner[1280:60b] Adding after 0 items, order = 1.00
2014-07-15 23:34:45.344 Homepwner[1280:60b] Error: unable to find /Users/inimam/Library/Application Support/iPhone Simulator/7.1-64/Applications/
4D69E3AD-3BA5-4F52-BFF4-7440BE5086E0/Documents/E4D2F59F-6FA8-4B0F-BF91-4CE98B81CD5A
2014-07-15 23:35:34.330 Homepwner[1280:60b] I am in showImage
2014-07-15 23:35:34.332 Homepwner[1280:60b] Going to show image for <INIItem: 0x10d926470> (entity: INIItem; id: 0x10d929a40 <x-coredata:///INIItem/
t4C45B155-31E7-4B18-BE55-E3252AC8D7352> ; data: {
    assetType = nil;
    dateCreated = "2014-07-16 03:34:45 +0000";
    itemKey = "E4D2F59F-6FA8-4B0F-BF91-4CE98B81CD5A";
    itemName = item1inCoreData;
    orderingValue = 1;
    serialNumber = "coredata_01";
    thumbnail = "<UIImage: 0x10d9ec050>";
    valueInDollars = 1;
})
```



Adding INIAssetTypes to Homeowner

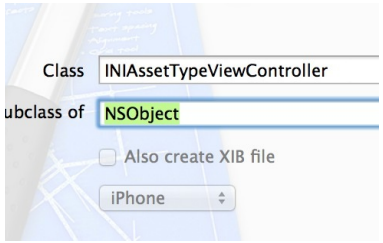
- We need to create the missing entity INIAssetType and to set the asset type for items in our INIItems entity.
- INIItemStore needs to be able to retrieve assets' types
- Modify INIItemStore:

```
- (NSString *)itemArchivePath;
- (BOOL)saveChanges;

- (NSArray *)allAssetTypes;

@end
```

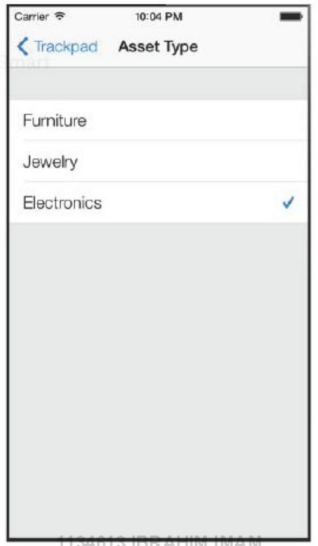
```
- (NSArray *)allAssetTypes
{
    if (!_allAssetTypes) {
        NSFetchRequest *request = [[NSFetchRequest alloc] init];

        NSEntityDescription *e = [NSEntityDescription entityForName:@"INIAssetType"
                                              inManagedObjectContext:self.context];
        request.entity = e;

        NSError *error = nil;
        NSArray *result = [self.context executeFetchRequest:request error:&error];
        if (!result) {
            [NSException raise:@"Fetch failed"
                        format:@"Reason: %@", [error localizedDescription]];
        }
        _allAssetTypes = [result mutableCopy];
    }

    // Is this the first time the program is being run?
    if ([_allAssetTypes count] == 0) {
        NSManagedObject *type;

        type = [NSEntityDescription insertNewObjectForEntityForName:@"INIAssetType"
                                              inManagedObjectContext:self.context];
        [type setValue:@"Furniture" forKey:@"label"];
        [_allAssetTypes addObject:type];

        type = [NSEntityDescription insertNewObjectForEntityForName:@"INIAssetType"
                                              inManagedObjectContext:self.context];
        [type setValue:@"Jewelry" forKey:@"label"];
        [_allAssetTypes addObject:type];

        type = [NSEntityDescription insertNewObjectForEntityForName:@"INIAssetType"
                                              inManagedObjectContext:self.context];
        [type setValue:@"Electronics" forKey:@"label"];
        [_allAssetTypes addObject:type];

    }
    return _allAssetTypes;
}
```

Creating a view to list assets type

- We want to create a table view controller to show a list of the available asset types.
- This view will be shown when we tap a button on the INIDetailViewController's view.




```
#import <Foundation/Foundation.h>

@class INIItem;

@interface INIAssetTypeViewController : UITableViewController

@property (nonatomic, strong) INIItem *item;

@end
```



INIAssetTypeViewController

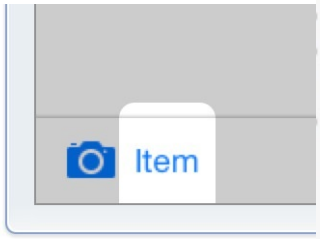
```objc
#import "INIAssetTypeViewController.h"
#import "INIItemStore.h"
#import "INIItem.h"

@implementation INIAssetTypeViewController

- (instancetype)init
{
    return [super initWithStyle:UITableViewStylePlain];
}

- (instancetype)initWithStyle:(UITableViewStyle)style
{
    return [self init];
}
```

```objc
- (void)viewDidLoad
{
    [super viewDidLoad];

    [self.tableView registerClass:[UITableViewCell class]
           forCellReuseIdentifier:@"UITableViewCell"];
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return [[[INIItemStore sharedStore] allAssetTypes] count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"UITableViewCell"
                                                            forIndexPath:indexPath];

    NSArray *allAssets = [[INIItemStore sharedStore] allAssetTypes];
    NSManagedObject *assetType = allAssets[indexPath.row];

    // Use key-value coding to get the asset type's label
    NSString *assetLabel = [assetType valueForKey:@"label"];
    cell.textLabel.text = assetLabel;

    // Checkmark the one that is currently selected
    if (assetType == self.item.assetType) {
        cell.accessoryType = UITableViewCellAccessoryCheckmark;
    } else {
        cell.accessoryType = UITableViewCellAccessoryNone;
    }

    return cell;
}
```

```objc
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView cellForRowAtIndexPath:indexPath];

    cell.accessoryType = UITableViewCellAccessoryCheckmark;

    NSArray *allAssets = [[INIItemStore sharedStore] allAssetTypes];
    NSManagedObject *assetType = allAssets[indexPath.row];
    self.item.assetType = assetType;

    [self.navigationController popViewControllerAnimated:YES];
}
```

Connecting Item's Details to Asset Type

- In INIDetailViewController.xib, we drag a UIBarButtonItem onto the toolbar.
- Create an outlet named assetTypeButton to this button by Control- dragging to the class extension of INIDetailViewController.m.
- Create an action from this button in the same way by dragging implementation section and name it showAssetTypePicker.




```
23 @property (weak, nonatomic) IBOutlet UIImageView *imageView;
24 @property (weak, nonatomic) IBOutlet UIToolbar *toolBar;
25 @property (weak, nonatomic) IBOutlet UIBarButtonItem *cameraButton;
26 @property (strong, nonatomic) UIPopoverController *imagePickerPopover;
27 @property (weak, nonatomic) IBOutlet UIBarButtonItem *assetTypeButton;
28
29 @end
30
31 @implementation INIDetailViewController
32 - (IBAction)showAssetTypePicker:(id)sender {
33 }
34
```

Finally

```
- (IBAction)showAssetTypePicker:(id)sender
{
    [self.view endEditing:YES];

    INIAssetTypeViewController *avc = [[INIAssetTypeViewController alloc] init];
    avc.item = self.item;

    [self.navigationController pushViewController:avc
                                         animated:YES];
}
```