Core Data

Dr.-Ing. Thomas Springer
M.Sc. Martin Weißbach
https://www.rn.inf.tu-dresden.de/martin/iOS-Programming-17/
Core-Data-Example.zip
Core Data – Key Characteristics

- introduced in OS X 10.4 in 2005
- object graph management and persistence framework
  - model layer helps to represent the application state
  - persistence layer stores the application state to disk
- neither an O/RM nor an SQL-wrapper
Core Data – API Stack

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObjectContext

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObjectContext

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObject

NSManagedObjectContext

NSManagedObjectContext

model layer

persistence layer

graphic adapted from [1]
NSManagedObject

• Model objects
  • represent the application state
  • loaded from and stored to persistent storage

• must belong to a managed object context

• related to an Entity
tracks changes made to managed objects
  - offers undo and redo functionality
tracks changes in relationships between managed objects

objects added to a context form an object graph or a collection of object graphs
  - manages the object graph in the model layer of Core Data

changes made to the model are not automatically saved to the database
  - save: message has to be sent explicitly to the context
• abstract base class for all persistent stores supported by Core Data

• responsible for mapping data in the store to corresponding objects in the managed object context
• **Binary**
  - atomic store – must be written and read in its entirety

• **XML**
  - not available on iOS
  - atomic

• **SQLite**
  - can be modified piecemeal
  - private format – not compatible with native SQLite API

• **In-Memory**
  - persistent store is kept in memory and never saved to disk
<table>
<thead>
<tr>
<th>Store Types</th>
<th>XML</th>
<th>Binary</th>
<th>SQLite</th>
<th>In-Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>Slow</td>
<td>Fast</td>
<td>Fast</td>
<td>Fast</td>
</tr>
<tr>
<td>Object Graph</td>
<td>Whole</td>
<td>Whole</td>
<td>Partial</td>
<td>Whole</td>
</tr>
<tr>
<td>Other Factors</td>
<td>Externally parseable</td>
<td>No backing required</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NSPersistentStoreCoordinator

• ties model layer and persistence layer together

• acts as a facade of the persistence layer for the model layer

• in most setups, exactly one NSPersistentStore is attached to the coordinator
  • attaching multiple stores is equally possible for more advanced use cases
Defining the model

- **Entity**
- **to-one relationship**
- **to-many relationship**

Inverse relationships are set automatically when the inverse relation is set.
Entity vs. NSManagedObject

- Entity lives in the persistence layer
- Managed Object lives in the model layer

- Entity describes the type of the managed object

- Entity provides additional information relevant for persistence
  - transient, optional properties and relationships
  - default values for properties and relationships
  - delete rules for relationships
Deny
• if at least one object at the relationship’s destination, the source will not be deleted

Nullify
• remove the relationship between two objects, but do not delete either

Cascade
• delete all objects at the relationship’s destination when the source is deleted

No Action
• Do nothing to the object at the destination of the relationship
• as well as classes, entities can inherit from other entities
• entity models look like class diagrams
Inheritance

• yet, inheritance should be avoided and entity hierarchies should be flattened
  • albeit entities are matched to managed objects, the inheritance hierarchy is independent

• performance deterioration possible
  • super-entities and sub-entities stored in the same table, if SQLite is the chosen persistence type [4]
**Fetch Request**

- Entity Name: "Employee"
- Predicate: 
- Sort Orderings:

**Predicate**

- Format: "department.name = 'IT'"

**Array**

- **Sort Descriptor**
  - Key: "salary"
  - Ascending: YES

- **Sort Descriptor**
  - Key: "lastName"
  - Ascending: YES

*graphic adapted from [3]*
NSFetchRequest

• returns model objects from a managed object context

• consists of three parts:
  • Entity Name: Name of entity for which model objects are to be retrieved
  • NSPredicate: used to filter the result set
  • NSSortDescriptor: modify the order of elements in the result set

• NSPredicate & NSSortDescriptor may be omitted
  • with no predicate set, all entries for the requested entities are fetched
• introduced with iOS10
• provides a default initialization of the Core Data stack that had to be set up manually before
• uses SQLite store type by default
References
