Scriptable Core Data

User's Guide

2008 June 14

Contents

Introduction to Scriptable Core Data

Overview of Scriptable Core Data

- The Scriptable Core Data Framework (SCD) is an open-source project for the Mac OS X (Leopard) developer community. The SCD distribution includes Xcode projects for the SCD Framework and three example applications.
  - See: Scriptable CoreData (SCD)
  - With the SCD Framework, adopting developers can implement AppleScript enabled Core Data Document-based applications,
    - starting with a new project;
    - from an existing Core Data Document-based application project; or
    - from an existing Cocoa Document-based application.
    - The resulting application can use Leopard's dynamic sdef feature to
      - calculate a useful AppleScript Dictionary at run time; or
      - resort to an sdef file composed in the usual way.
    - The SCD Framework can optionally provide persistent order among members of to-many relationships. This feature is independent of the AppleScript dictionary, or even whether AppleScript is enabled for the application.
- During the development cycle, the adopting developer can use AppleScript to test incrementally. Test scripts can use an AppleScript dictionary generated dynamically from the data model. The dynamic sdef tracks changes in the data model through the development cycle.
- The dynamic sdef simply exposes the Core Data Model and a document's graph of objects. Scripters can "peak and poke" at the object graph and at the attributes of the objects. They can also modify the graph of objects with "make new", "duplicate", and "delete" commands.
- Apple encourages Cocoa developers of new applications to provide at least a minimal AppleScript dictionary initially. Then, the users and scripters can guide the developers toward the additional AppleScript features they want in later versions. The AppleScript dictionary provided dynamically by the SCD Framework provides "just enough scriptability," and can serve as a useful initial dictionary.

Status and Directions

- Status
  - The SCD Framework is well beyond a mere feasibility study, but has not nearly reached maturity.
  - The first focus of efforts on SCD Framework was: "How can we do this at all?" The result was the Bricks+SCD example.
  - The next focus was: "Can we retrofit a simple existing Core Data Document-based app?" The result was the OutlineEdit+SCD example.
  - Then, the focus was: "Can we retrofit an existing Cocoa Document-based app for Core Data persistence and AppleScript? The result was the Sketch+SCD example.
  - No effort has yet been directed toward a set of intermediate but simpler problems that arise next: "Given a Core Data Document-based application that implements its own order among some of the to-many relationships in its Data Model, can we add Data Model user info annotations to support dynamic sdef generation seamlessly?" At this writing, it seems likely that to answer affirmatively, a first step would be to add some user info keys and values for entities and relationships. The SCD Framework should support:
    - an entity user info key to indicate that the entity needs no additional SCD support to implement AppleScript;
    - a relationship user info key to indicate the the relationship already orders its elements, and needs no additional SCD support; and
    - a relationship user info key to specify a custom subclass of SCDToManyArray.
- Directions
  - NSError/NSException management is sparse.
  - There is no direct support for Fetched Properties. Its development awaits a good example application. SCD does not prevent the use of Fetched Properties, it merely fails to provide a way to enable them as AppleScript properties.
  - The SCD build settings specify automatic garbage collection. No effort has been expended toward "manual" garbage collection in the SCD Framework. This work may wait until some adopting project needs it.
  - There is no support for cross-checking AppleScript four-character codes in a dynamically generated sdef. Some of

the related design choices remain unclear to the SCD collaboration at this writing.

Why AppleScript?

- Cocoa developers should see AppleScriptability as a way of helping their customers. Customers can extend an AppleScript enabled application in ways that the developer cannot foresee. Customers can build features tailored to their specific needs.
- For the Mac OS X user, AppleScript is more than just the language. It is also the set of scriptable applications suitable to the work at hand. Each application's AppleScript dictionary extends the AppleScript language in directions useful to its users.
- The "UNIX Way" for "the Rest of Us"
  - In some important ways, AppleScript—the language plus the applications—provides for Mac OS X scripters something comparable to the "UNIX Way." It provides lots of simple, interoperable, scriptable tools, and then lets users exploit and combine them in AppleScripts to solve their own complicated problems.
  - Apple's Automator application makes this process even more approachable and powerful.
- Customer Loyalty
  - Historically, customers who script an application have become very loyal customers. To put it crassly, once you've got 'em by their "Business Logic," their hearts and minds will follow.
  - To put it more tactfully, your application builds customer loyalty by enabling and empowering your customers.
- Developer Loyalty, too
  - AppleScript provides another avenue for regression testing. And of course, once you ship an AppleScript enabled app, it becomes a key regression testing mechanism.

Why Core Data?

- Core Data is a big advance in Cocoa.
  - Primarily, Core Data provides a persistence mechanism, but it also provides advanced support for:
    - Undo/Redo
    - More general change propagation and management, including maintaining consistency among relationships. Think "Undo/Redo for the graph of objects in your app's Data Model."
    - KVC and KVO, including optional support for bindings.
    - Fetching and Filtering.
  - The Core Data Model brings the Entity-Relationship formalism to the "Model" component of the "Model-View-Controller" design pattern. Xcode's Data Model editor makes your "Model" explicit and regular. The developer sees and edits a graph of entities, their attributes, and their relationships.
  - Core Data helps enormously to manage more complex projects.

Prior Work on Core Data and AppleScript

- Several earlier publication deserve recognition.
  - Bill Cheeseman's Wareroom Demo
    http://www.quecheesoftware.com/downloads/WareroomDemo.html
    See the extensive AppleScript test suite!
  - Red Sweater Blog, "We need a hero"
    http://www.red-sweater.com/blog/195/we-need-a-hero
  - mac geekery, "Adding Basic AppleScript to Core Data Applications"
    http://www.macgeekery.com/development/
- All these authors inspired and motivated the work in the SCD Framework, but of these, Bill Cheeseman's work provided the most in-depth example and the most valuable insight.

Goals of the SCD Framework

"Make the pain go away!"

- The primary goal of the SCD Framework is to reduce the threshold to AppleScript enable a Core Data Document-based application. As the reader can readily see from the prior work, this ain't so easy.
  - Part of that pain has included the additional tooling for composing an sdef, and the additional effort to keep the sdef and data model synchronized. The developer typically uses Xcode as the primary IDE, but must use some other tool to compose the sdef.
    - The author highly recommends: Sdef Editor.
    - However, if the Data Model changes, typically the sdef must change, too.
  - Instead, the SCD Framework lets the developer exploit a feature of Xcode to annotate the Data Model.
    - The sdef generated dynamically at run time automatically tracks the current Data Model:
      - Motto: "Data Model + codes + terms = sdef".
- A recent addition to the SCD Framework is support for dynamic KVO, per relationship. An object—a controller or a view, for example—can add itself as a observer to a to-many relationship in the Data Model. The SCD Framework then:
  - adds the observer to any objects already in the relationship;
  - adds the observer to any objects subsequently added to the relationship; and
  - removes the observer from objects subsequently removed from the relationship; while
  - mindfully respecting the managed object life cycle.
  - SCD's API is distinct from the normal Cocoa API for KVO, so the adopting developer won't invoke this behavior by accident.
- The SCD Framework provides support for optional persistent order among members of to-many relationships.
  - Each managed object can preserve the order in any of its to-many relationships.

Dynamic sdef Generated at Run Time from your Data Model

- Dynamic sdef generation is a new feature in Mac OS X 10.5 Leopard. When an AppleScript or Script Editor needs to use your app's sdef, it sends an event to your app, and SCD handles the event.
- By default, the SCD framework generates a useful sdef at run time from the current version of the Data Model, using the developer's annotations. In your Info.plist, just add:

```
<key>NSAppleScriptEnabled</key>
<string>YES</string>
<key>OSAScriptingDefinition</key>
<string>dynamic</string>
```

- You can override SCD's support to return a different sdef, perhaps conventionally composed and stored in your app's Resources as usual.

"Opt-in"

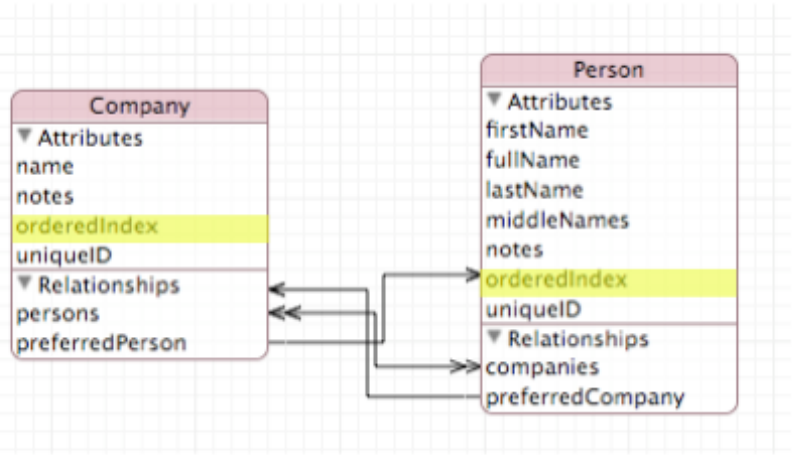
- The adopting developer can "opt in" for the features of the SCD Framework
  - per entity
  - per attribute
  - per relationship
- You can bypass SCD in any entity. Just let it inherit from NSManagedObject directly or through some different hierarchy.

Make the first 90% easy

- The SCD Framework makes it easy to AppleScript enable a new or an existing Core Data Document-based application. The first 90% may be all you need for an initial AppleScript enabled version.
- It does NOT make the remaining 10% impossible. The remaining 10% amounts to providing support for AppleScript commands, events, and types. The developer achieves that last 10% in the same way as before, namely by composing and testing a more extensive sdef. See Sketch+SCD. However, with SCD, the developer has the advantage of starting with a working AppleScript enabled application and a basic, working sdef.

Come for the AppleScript, Stay for the Persistent Order

- The SCD Framework manages order among members of to-many relationships at the framework level, and the developer can "opt in" for persistent order, per relationship, with or without AppleScript enabling the relationship.
- Core Data's lack of persistent order amounts to a delayed failure of WYSIWYG, at least for "readably small" to-many relationships. It just "looks wrong" to make a Core Data document with a prominent to-many relationship, save the document, re-open it, and discover a different order.
  - Consider the Core Data example project, OutlineEdit, at: /Developer/Examples/CoreData/OutlineEdit
  - Compile and run the application. Build a non-trivial outline document. Save it, and re-open it. You'll discover that the document forgets the order of the entries. That failure renders the application useless for its nominal purpose, to "edit outlines."
  - See also OutlineEdit+SCD.
- AppleScript requires indexed access to to-many relationships, and some implementers have added special index attributes to their entities to support this. This technique works well enough for one-to-many relationships, but does not scale well in the case of many-to-many relationships where every container maintains its own order of "containees."



Data Model, Ware Room Demo

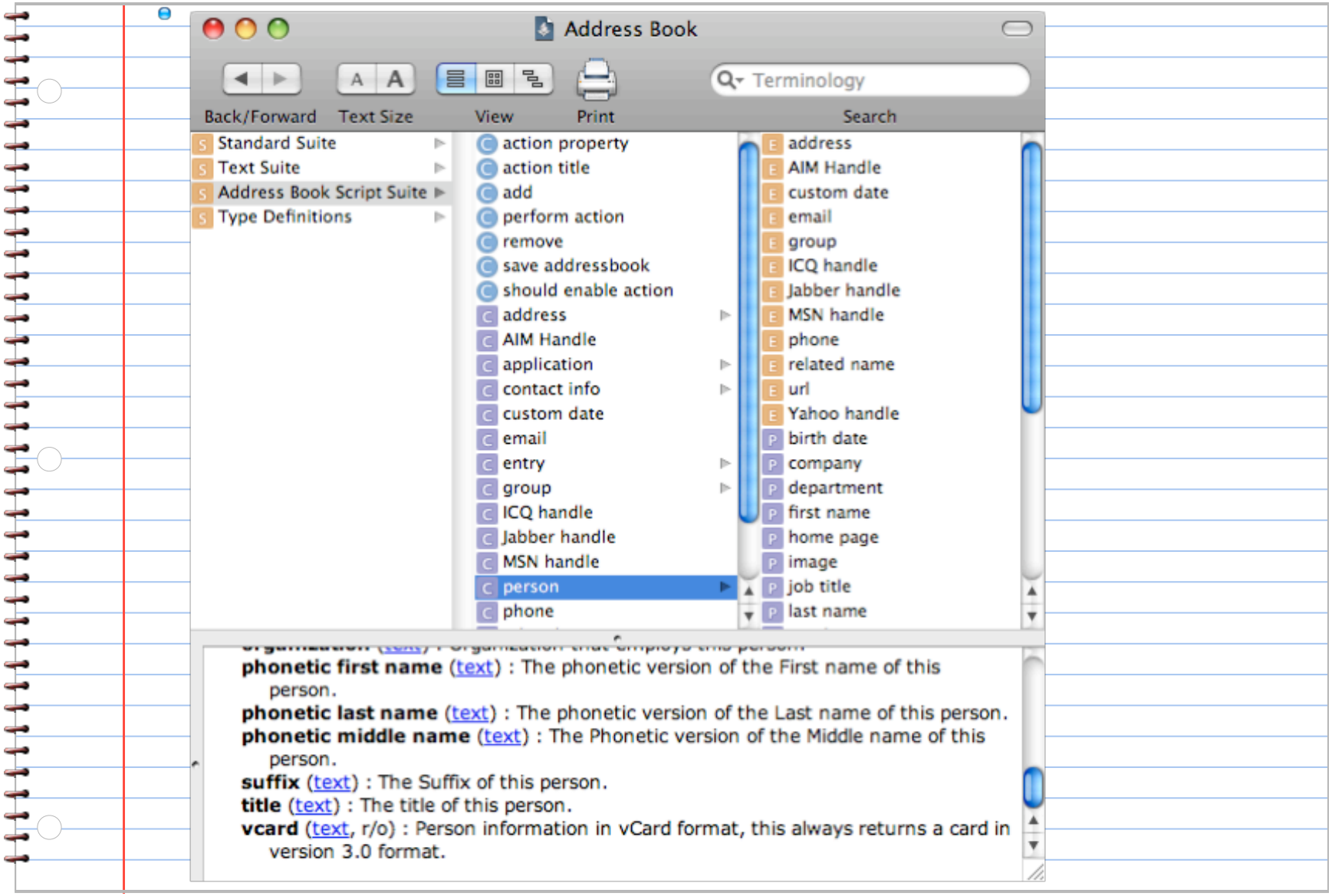
- The SCD Framework provides an opt-in way for a container to manage and even preserve the order of its "containees", per relationship.
- Of course, for relationships that expect very large numbers of members, persistent order may not be appropriate. The adopting developer must evaluate the tradeoffs in performance and footprint.

Core Data and AppleScript—What's the Same and What's Different?

- The SCD Framework shows that Core Data and AppleScript can work together readily. Adopting developers should be aware of the leading technical issues in the design decisions, and how the SCD Framework addresses them.
- What's the Same?
  - A Core Data Document-based application depends on its data model. Its data model resides in a file (or files) with the extension *.xcoredata, and provides a detailed, regular description of the application's "Model" component of the "Model-View-Controller" design pattern. These data models use the "Entity-Relationship" formalism.
  - An AppleScriptable Cocoa application depends on its AppleScript Dictionary. The dictionary can reside in a few forms, but this discussion focusses on the sdef form exclusively. An sdef AppleScript Dictionary provides a different but comparable view of the application's "Model" component—the Classes, Properties, and Elements of an AppleScript suite.
  - AppleScript Classes correspond to Core Data entities.
  - AppleScript properties correspond to Core Data to-one relationships and attributes.
  - AppleScript elements correspond incompletely to Core Data to-many relationships.
  - Motto: "Data Model + codes + terms = sdef"
    - These two expressions of an application's "Model" are sufficiently similar that a developer can compose a skeletal AppleScript suite from a Core Data Model "by inspection." The process requires some additional information, though. The developer must choose four-character codes and terms (names) for the AppleScript suite's Classes and for the scalar Properties. A reverse operation is also straight-forward. A developer can inspect an AppleScript dictionary and, with restrictions, compose a data model from it.
- What's Different?
  - Core Data's to-many relationships use NSSets. AppleScript expects something more like NSArrays. Core Data provides no intrinsic means of ordering members of to-many relationships, but AppleScript relies on indexed accessors for to-many relationships.
  - AppleScript relies on a "containment hierarchy." A scriptable Core Data Document-based application must explicitly

declare the document's containment hierarchy. The SCD Framework provides an easy and natural way to attend to this detail, but it amounts to an extra step.

- AppleScript Dictionary Suites also provide Types, Commands, and Events. The Core Data Model has no natural place for these constructs. Developers who must provide these AppleScript features can override SCD's dynamic sdef.
- A given AppleScript class can have no more than one collection per contained class. Consider the Address Book AppleScript dictionary:


Address Book
Back/Forward
Text Size
View
Print
Terminology
Search
Standard Suite
Text Suite
Address Book Script Suite
Type Definitions
action property
action title
add
perform action
remove
save addressbook
should enable action
address
AIM Handle
application
contact info
custom date
email
entry
group
ICQ handle
Jabber handle
MSN handle
person
phone
address
AIM Handle
custom date
email
group
ICQ handle
Jabber handle
MSN handle
phone
related name
url
Yahoo handle
birth date
company
department
first name
home page
image
job title
last name
phonetic first name (text) : The phonetic version of the First name of this person.
phonetic last name (text) : The phonetic version of the Last name of this person.
phonetic middle name (text) : The Phonetic version of the Middle name of this person.
suffix (text) : The Suffix of this person.
title (text) : The title of this person.
vcard (text, r/o) : Person information in vCard format, this always returns a card in version 3.0 format.

- Suppose we wanted to add to the "person" class a few collections of different kinds of other persons, say, relatives, friends, neighbors, and coworkers. Sorry: if you add an element comprising instances of the person class, it must take the unique name "people", which is the declared plural of person.
- One workaround for this restriction has been to declare an empty subclass for each additional collection. Core Data has no such restriction. An entity can have an arbitrary number of to-many relationships to a given target entity. See the Sketch+SCD example for a proposed workaround with SCD (SCD Commands to expose NSScriptKeyValueCoding).
- In Core Data, the typical to-many relationship has an an inverse relationship. By default, SCD exposes the inverse relationship to AppleScript.

How the SCD Framework matches Core Data to AppleScript

- Main Classes of the SCD Framework
  - The SCD Framework provides scriptable subclasses for two important Core Data classes.
    - SCDPersistentDocument:NSPersistentDocument
      - The adopting developer subclasses SCDPersistentDocument instead of NSPersistentDocument.
    - SCDManagedObject:NSManagedObject
      - Scriptable entities in the Data Model subclass from SCDManagedObject instead of NSManagedObject.
      - Non-scriptable entities remain as before.
      - The SCDManagedObject class uses a helper class cluster, SCDToManyArray, to manage the order of its scriptable to-many relationships, on an "opt-in" basis.
        - Instances of SCDManagedObject instantiate one instance of SCDToManyArray or a subclass, per to-many relationship, lazily.
  - The SCD Framework provides a bridge between the document and its Data Model's object graph. SCDContainmentRoot is a helper class for SCDPersistentDocument. This abstract parent class manages the containment hierarchy for a persistent document. The adopting developer creates a custom subclass of SCDContainmentRoot in the Data Model to match the custom subclass of SCDPersistentDocument.
- NSManagedObjectID and 'ID '
  - AppleScript can make use of an attribute of type NSString or NSNumber with the four-character code 'ID ' for accessing elements "by ID". The SCDManagedObject class uses the Core Data's NSManagedObjectID's URI representation in string form.
    - The dynamic sdef declares "unique ID" as a read-only AppleScript Property of the "managed object" class (SCDManagedObject).
    - The `uniqueID` accessor calculates its return value from the receiver's NSManagedObjectID.
    - The SCD Data Model does not show a matching attribute.

- AppleScript requires an object specifier (`NSScriptObjectSpecifier`) for each scriptable managed object. SCD returns an instance of `NSUniqueIDSpecifier`, using the object's `uniqueID`.

- Dynamic sdef Generation
  - Xcode ships with a little-used feature: a developer can annotate every entity, attribute, and relationship in a Data Model by adding keys and values to its "user info" dictionary. There's already a natural place in the Data Model for the codes and terms we need for the sdef. The SCD Framework defines some keys, and the adopting developer provides the matching values.

- The Sketch+SCD example provides provisional AppleScript API, in the form of a set of commands, to expose Core Data's richer notion of containment, and SCDManagedObject's NSScriptKeyValueCoding protocol. The implementation uses categories on classes (entities) of the SCD framework. These categories could readily move to the SCD framework.

The Model-Document-Suite Design Pattern

- Core Data uses a "Model-Document" design pattern. The pattern is established by the Xcode template for a new project to support a “Core Data Document-based Application.” The template yields:
  - a custom class named MyDocument, subclassed from NSPersistentDocument; and
  - MyDocument.xcdatamodel.
- The SCD Framework elaborates this design pattern to "Model-Document-Suite." If we retain "MyDocument" as a shared name, the result is
  - a custom class named MyDocument, subclassed from SCDPersistentDocument;
  - a custom class named MyContainmentRoot, a "helper class" for MyDocument; and
  - MyDocument.xcdatamodel.
  - The default dynamically generated sdef declares an AppleScript Suite for the document, based on its containment root.
- Each instance of MyDocument owns a singleton instance of MyContainmentRoot. MyContainmentRoot inherits from SCDContainmentRoot. MyDocument instantiates its MyContainmentRoot instance lazily, or retrieves it from its store. The adopting developer implements a method in the custom document class:
  - In MyDocument.m

```
+(NSString*)containmentRootEntityName
{
    return @"MyContainmentRoot";
}
```

  - Compare this technique to the familiar
  -

```
- (NSString *)windowNibName
{
    return @"MyDocument";
```

```
}
```

- MyContainmentRoot serves as:
  - the bridge between an AppleScript enabled document and its graph of scriptable Core Data objects;
  - the root of the document's AppleScript containment hierarchy;
  - a place for "user info" annotations that specify the AppleScript Suite associated with the Data Model; and
  - a place for any persistent attributes the document itself may need—the document's "print info", for example.

SCD Keys and Values in the Data Model "User Info"

Xcode's Data Model Editor's User Info for Entities, Relationships, and Attributes

- Apple's documentation for the Core Data Framework for the classes NSEntityDescription and NSPropertyDescription describe the "userInfo" and "setUserInfo:" methods. Xcode's Data Model Editor gives the developer "table view" access to these methods, and the SCD Framework exploits it. Some screen shots follow, taken from one of the example application projects, Bricks+SCD.

MyDocument.xcdatamodel

MyDocument.xcdatamodel | Brick

Entity	Abs	Cla
Brick	☐	Brick
MyContainmentRoot	☐	MyC
SCDContainmentRoot	☐	SCD
SCDManagedObject	☑	SCD
SCDPersistentToMany	☐	SCDP
SCDSortedToManyArr	☐	SCDS
SCDToManyArray	☐	SCD

Property	Kind	Type or Destination
bricksRoot	Relationship	MyContainmentRoot
name	Attribute	String

Entity

User Info:

Key	Value
sdef code	Brik
sdef description	This entity is dumb
sdef plural	bricks
sdef term	brick

Of "General, User Info, Configuration, and Synchronization," choose "User

MyDocument.xcdatamodel

MyDocument.xcdatamodel | Brick

Entity	Abs	C
Brick	☐	E
MyContainmentRoot	☐	M
SCDContainmentRoot	☐	S
SCDManagedObject	☑	S
SCDPersistentToManyArray	☐	S
SCDSortedToManyArray	☐	S
SCDToManyArray	☐	S

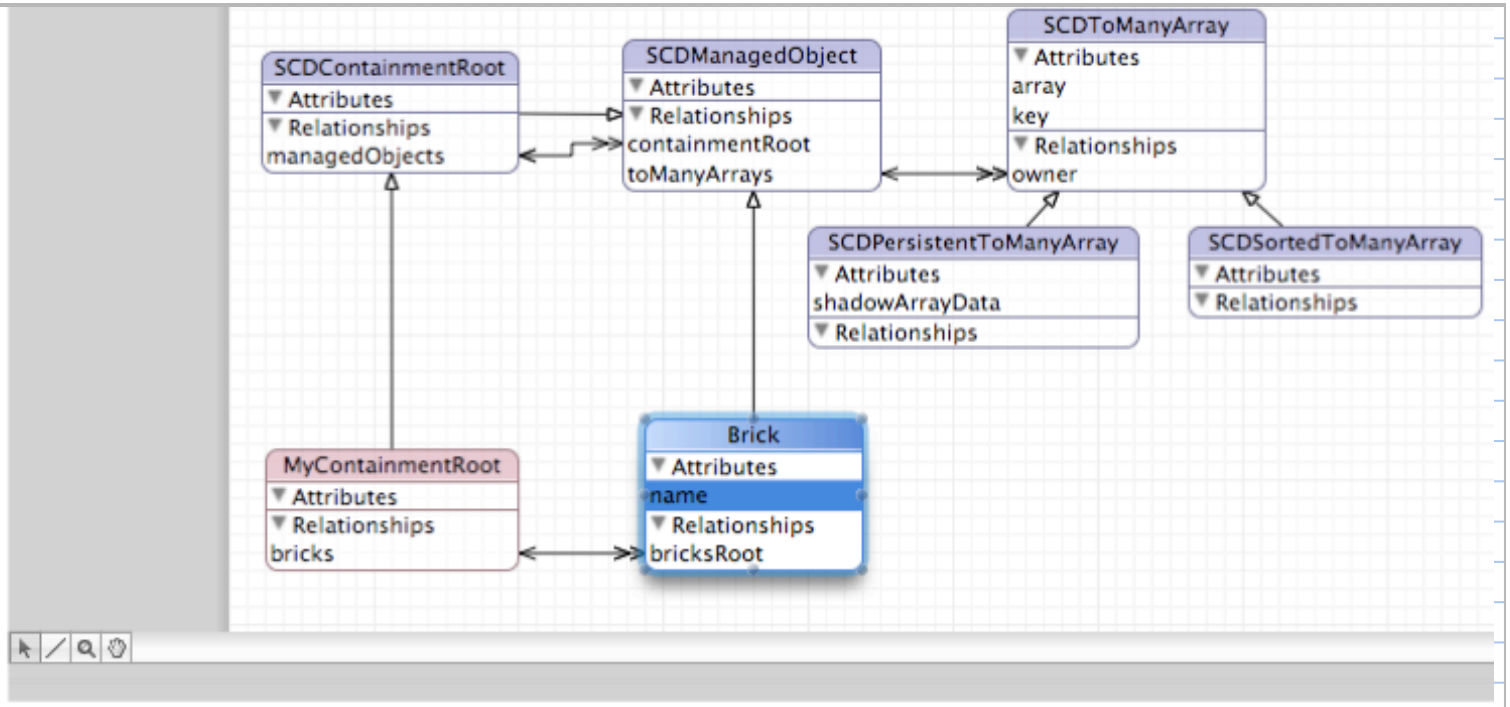
Property	Kind	Type or Destination
bricksRoot	Relationship	MyContainmentRoot
name	Attribute	String

Attribute

User Info:

Key	Value
sdef term	name
sdef description	hyper-normal,
sdef code	pnam
sdef cocoa key	name

Ver. Hash Modifier:



Xcode's Data Model Editor's User Interface for "User Info"

And here's how those user info entries appear in the resulting sdef:

Bricks+SCD

Styles | Spacing | Lists

```
            </element>
        </class>
        <class name="managed object" code="SCDo" description="scriptable managed object, Scriptable Core Data (SCD) framework
            <cocoa class="SCDManagedObject"></cocoa>
            <property name="unique id" code="ID  " description="Core Data objectID's absolute URI, in string form" type="text
                <cocoa key="uniqueID"></cocoa>
            </property>
        </class>
        <class name="brick" code="Brik" description="This entity is dumb as a brick." plural="bricks" inherits="managed objec
            <cocoa class="Brick"></cocoa>
            <property name="name" code="pnam" description="hyper-normal, plain-vanilla name attribute of type string" type="t
                <cocoa key="name"></cocoa>
            </property>
        </class>
    </suite>
</dictionary>
```

How SCD Exploits User Info in the Data Model

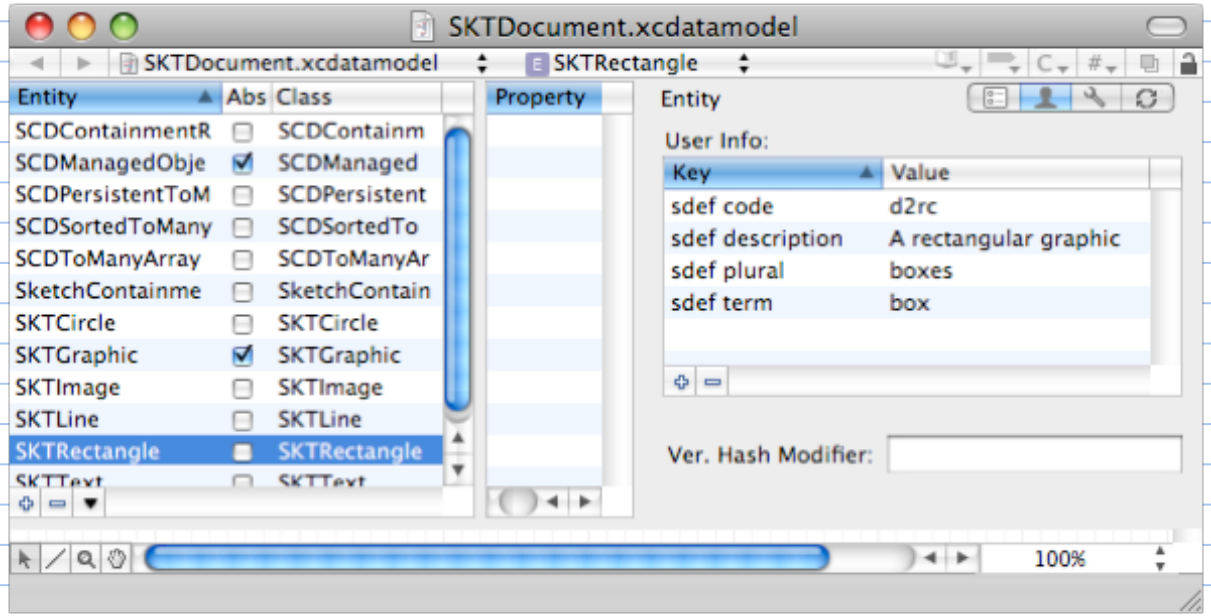
Dynamic sdef Generation: Data Model + codes + terms = sdef

Scriptable Entities, Attributes, and Relationships: sdef keys and values

- Entities
  - A scriptable entity must inherit from SCDManagedObject
  - Its user info
    - must have a four-character value for the key "sdef code"
    - must have a value for the key "sdef term"
    - may have a value for the key "sdef description"
    - may have a value for the key "sdef plural"
  -

Key	Description
sdef code	a four-character code
sdef description	description
sdef term	the sdef “name”
sdef plural	the sdef “plural”

User Info Keys and Values for a Scriptable Entity




```
            <property name="unique id" code="ID  " description="Core Data objectID's absolute URI, in string form"
type="text" access="r">
                <cocoa key="uniqueID"></cocoa>
            </property>
        </class>
        <class name="box" code="d2rc" description="A rectangular graphic" plural="boxes" inherits="graphic">
            <cocoa class="SKTRectangle"></cocoa>
            <property name="fill color" code="fclr" description="fill color" type="color">
                <cocoa key="scriptingFillColor"></cocoa>
            </property>
```

A scriptable entity may have scriptable attributes and relationships.

- Attributes
  - A scriptable attribute must have a Core Data type that maps to an AppleScript type.
    - String maps to text.
    - Bool maps to boolean.
    - {Int 16, Int 32, Int 64} map to integer.
    - {Decimal, Double, Float} map to real.
    - {Bool, Int 16, Int 32, Int 64, Decimal, Double, Float} map to number.
    - Other types may require special attention.
  - Its user info
    - must have a four-character value for the key "sdef code"
    - must have a value for the key "sdef term"
    - may have a value for the key "sdef description"
    - may have a value for the key "sdef KVC accessor"
  -

Key	Description
sdef code	a four-character code
sdef description	description
sdef term	the sdef “name”
sdef KVC accessor	the sdef “cocoa key”

User Info Keys and Values for a Scriptable Attribute

Entity	Abs	Class
SCDContainmentR	☐	SCDContai
SCDManagedObje	☑	SCDManag
SCDPersistentToM	☐	SCDPersist
SCDSortedToMany	☐	SCDSorted
SCDToManyArray	☐	SCDToMan
SketchContainme	☐	SketchCont
SKTCircle	☐	SKTCircle
SKTGraphic	☑	SKTGraphic
SKTImage	☐	SKTImage
SKTLine	☐	SKTLine
SKTRectangle	☐	SKTRectan
SKTText	☐	SKTText

Property	Kind	Type or
boundsAsNSValue	Attribute	Undefined
boundsShadowData	Attribute	Binary
docRoot	Relationship	SketchContai...
fillColor	Attribute	Undefined
fillColorData	Attribute	Binary
isDrawingFill	Attribute	Bool
isDrawingStroke	Attribute	Bool
strokeColor	Attribute	Undefined
strokeColorData	Attribute	Binary
strokeWidth	Attribute	Float

Attribute

User Info:

Key	Value
sdef KVC accessor	scriptingStrok
sdef code	slwd
sdef description	the width of t
sdef term	stroke thickn

Ver. Hash Modifier:

```
<class name="graphic" code="grph" description="An object in a Sketch+SCD document. There are subclasses for each kind of graphic." plural="graphics" inherits="managed object">
    <cocoa class="SKTGraphic"></cocoa>
    <property name="fill color" code="fclr" description="fill color" type="color">
        <cocoa key="scriptingFillColor"></cocoa>
    </property>
    <property name="stroke color" code="sclr" description="stroke color" type="color">
        <cocoa key="scriptingStrokeColor"></cocoa>
    </property>
    <property name="stroke thickness" code="slwd" description="the width of the stroke" type="real">
        <cocoa key="scriptingStrokeWidth"></cocoa>
    </property>
</class>
```

Relationships

- A relationship is scriptable if its owner and target entities are scriptable.
- Its user info may have a value for the key "sdef description".

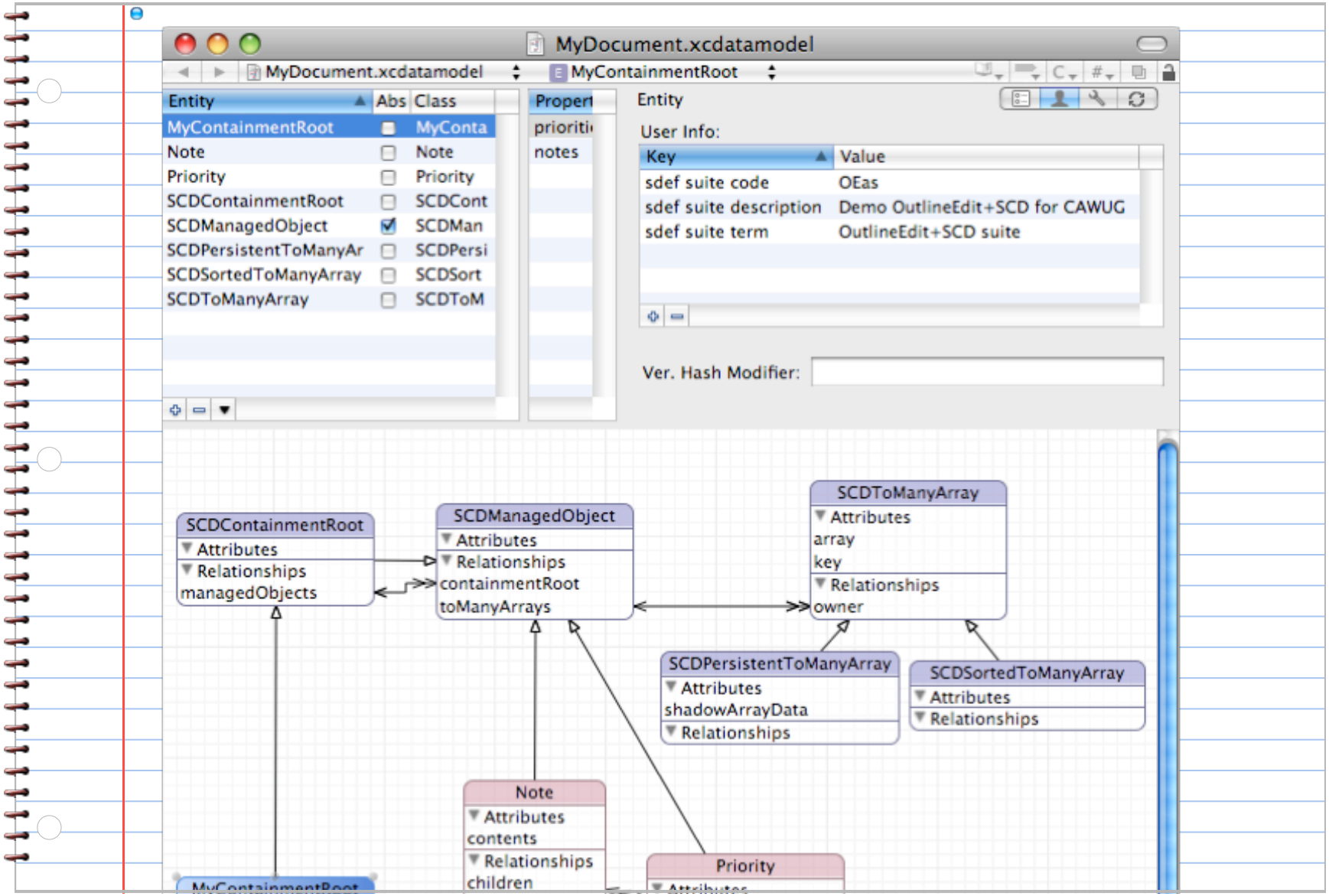
The Containment Root

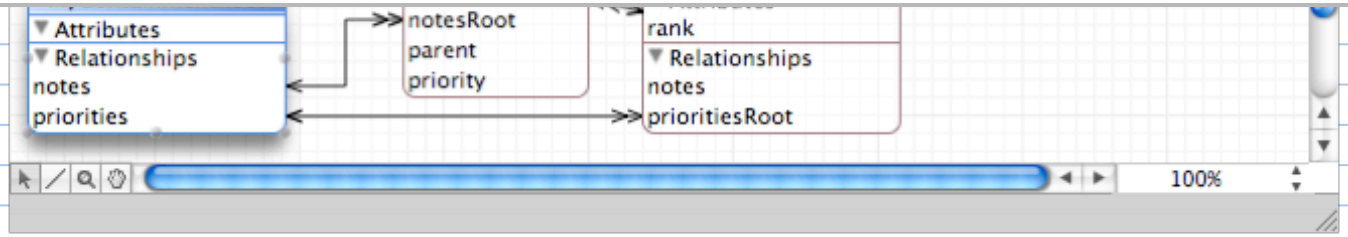
Specifying the Suite's sdef keys and values

- The SCD Framework uses the Document's custom containment root entity to hold the keys and values that specify the suite.
-

Key	Description
sdef suite code	a four-character code for the suite
sdef suite description	a description for the suite
sdef suite term	the sdef “name” for the suite

Custom Containment Root User Info Keys


MyDocument.xcdatamodel
MyDocument.xcdatamodel
MyContainmentRoot
Entity
Abs
Class
MyContainmentRoot
Note
Priority
SCDContainmentRoot
SCDManagedObject
SCDPersistentToManyAr
SCDSortedToManyArray
SCDToManyArray
Entity
User Info:
Key
Value
sdef suite code
OEas
sdef suite description
Demo OutlineEdit+SCD for CAWUG
sdef suite term
OutlineEdit+SCD suite
Ver. Hash Modifier:
SCDContainmentRoot
Attributes
Relationships
managedObjects
SCDManagedObject
Attributes
Relationships
containmentRoot
toManyArrays
SCDToManyArray
Attributes
array
key
Relationships
owner
SCDPersistentToManyArray
Attributes
shadowArrayData
Relationships
SCDSortedToManyArray
Attributes
Relationships
Note
Attributes
contents
Relationships
children
Priority

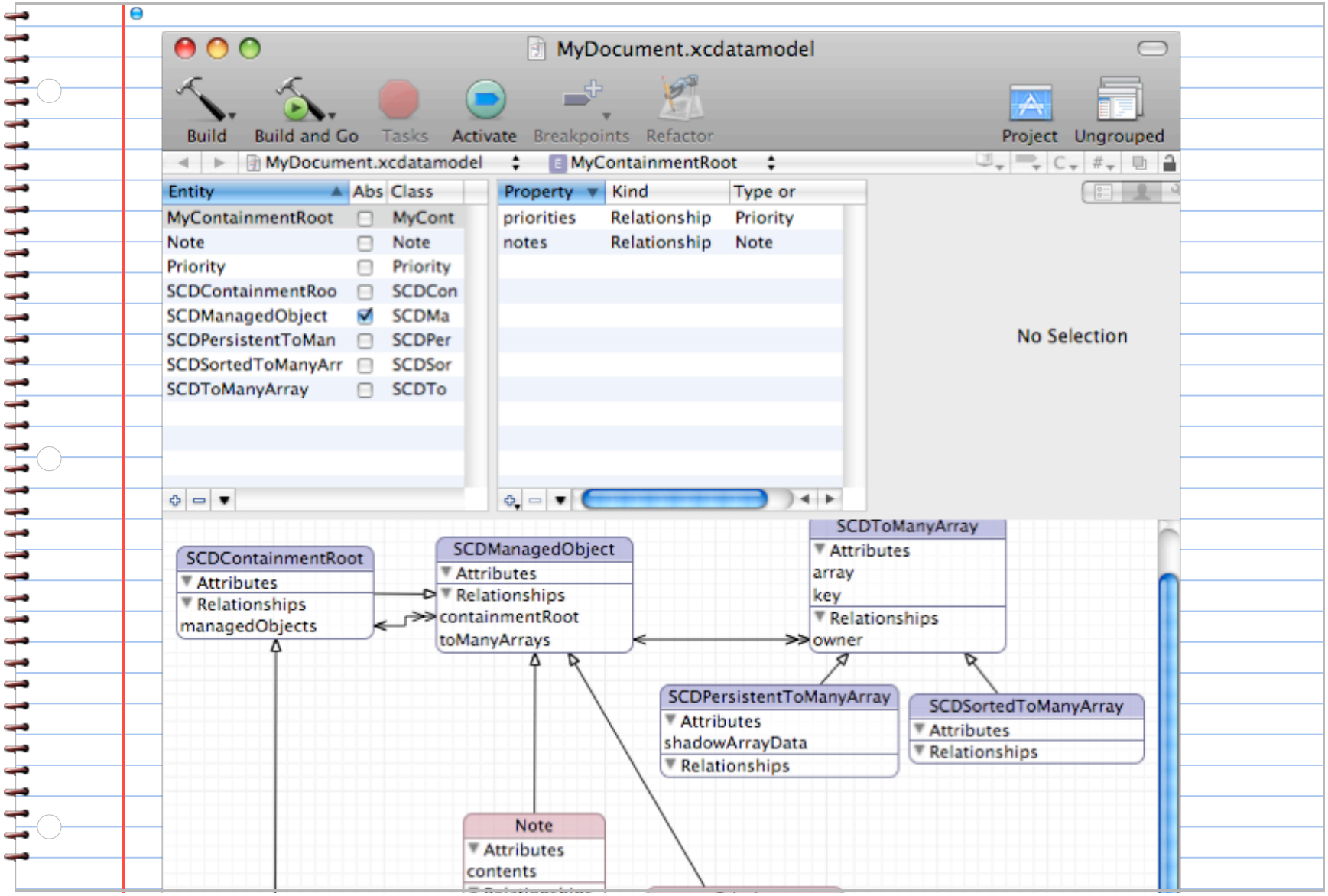


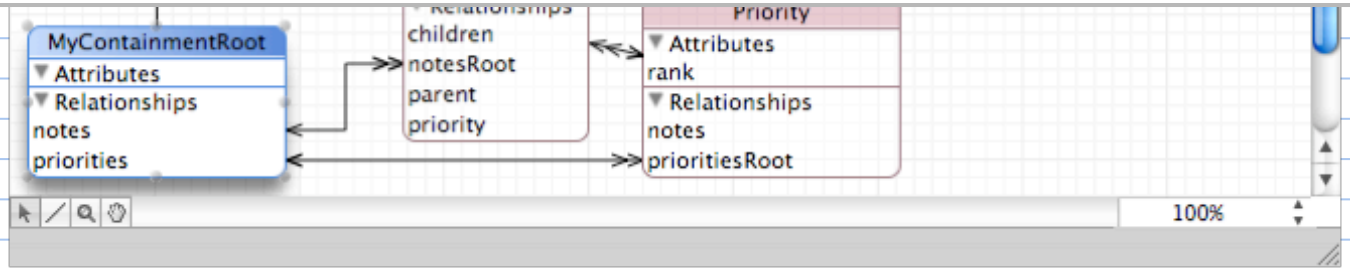
The corresponding section of the generated sdef follows:

```
            </responds-to>
        </class>
    </suite>
    <suite name="OutlineEdit+SCD suite" code="OEas" description="Demo OutlineEdit+SCD for CAWUG">
        <!--Rough draft sdef generated from a data model by the Scriptable Core Data (SCD) Framework.-->
        <class name="document" code="docu" inherits="document" description="Demo OutlineEdit+SCD for CAWUG">
            <cocoa class="MyDocument"></cocoa>
            <element type="managed object" access="rw" description="Description forthcoming">
                <cocoa key="managedObjects"></cocoa>
            </element>
```

Specifying the Document's Elements and Properties: sdef keys and values

- The SCD Framework uses the Document's custom containment root entity to extend the AppleScript document class ('docu') with elements (to-many relationships) and properties (attributes).
- The generated sdef exposes any scriptable relationships of the custom containment root as elements of the document.
- The relationship's user info may have a key "sdef description" with a value.


MyDocument.xcdatamodel
Build
Build and Go
Tasks
Activate
Breakpoints
Refactor
Project
Ungrouped
MyDocument.xcdatamodel
MyContainmentRoot
Entity
Abs
Class
MyContainmentRoot
MyCont
Note
Note
Priority
Priority
SCDContainmentRoo
SCDCon
SCDManagedObject
SCDMa
SCDPersistentToMan
SCDPer
SCDSortedToManyArr
SCDSor
SCDToManyArray
SCDTo
Property
Kind
Type or
priorities
Relationship
Priority
notes
Relationship
Note
No Selection
SCDContainmentRoot
Attributes
Relationships
managedObjects
SCDManagedObject
Attributes
Relationships
containmentRoot
toManyArrays
SCDToManyArray
Attributes
array
key
Relationships
owner
SCDPersistentToManyArray
Attributes
shadowArrayData
Relationships
SCDSortedToManyArray
Attributes
Relationships
Note
Attributes
contents




```
<class name="document" code="docu" inherits="document" description="Demo OutlineEdit+SCD for CAWUG">
    <cocoa class="MyDocument"></cocoa>
    <element type="managed object" access="rw" description="Description forthcoming">
        <cocoa key="managedObjects"></cocoa>
    </element>
    <element type="note" access="rw" description="Description forthcoming">
        <cocoa key="notes"></cocoa>
    </element>
    <element type="priority" access="rw" description="Description forthcoming">
        <cocoa key="priorities"></cocoa>
    </element>
</class>
```

- The generated sdef may expose any scriptable attributes of the custom containment root as properties of the document. (No examples provided.)

Specifying SCD Core Data Run Time Behavior

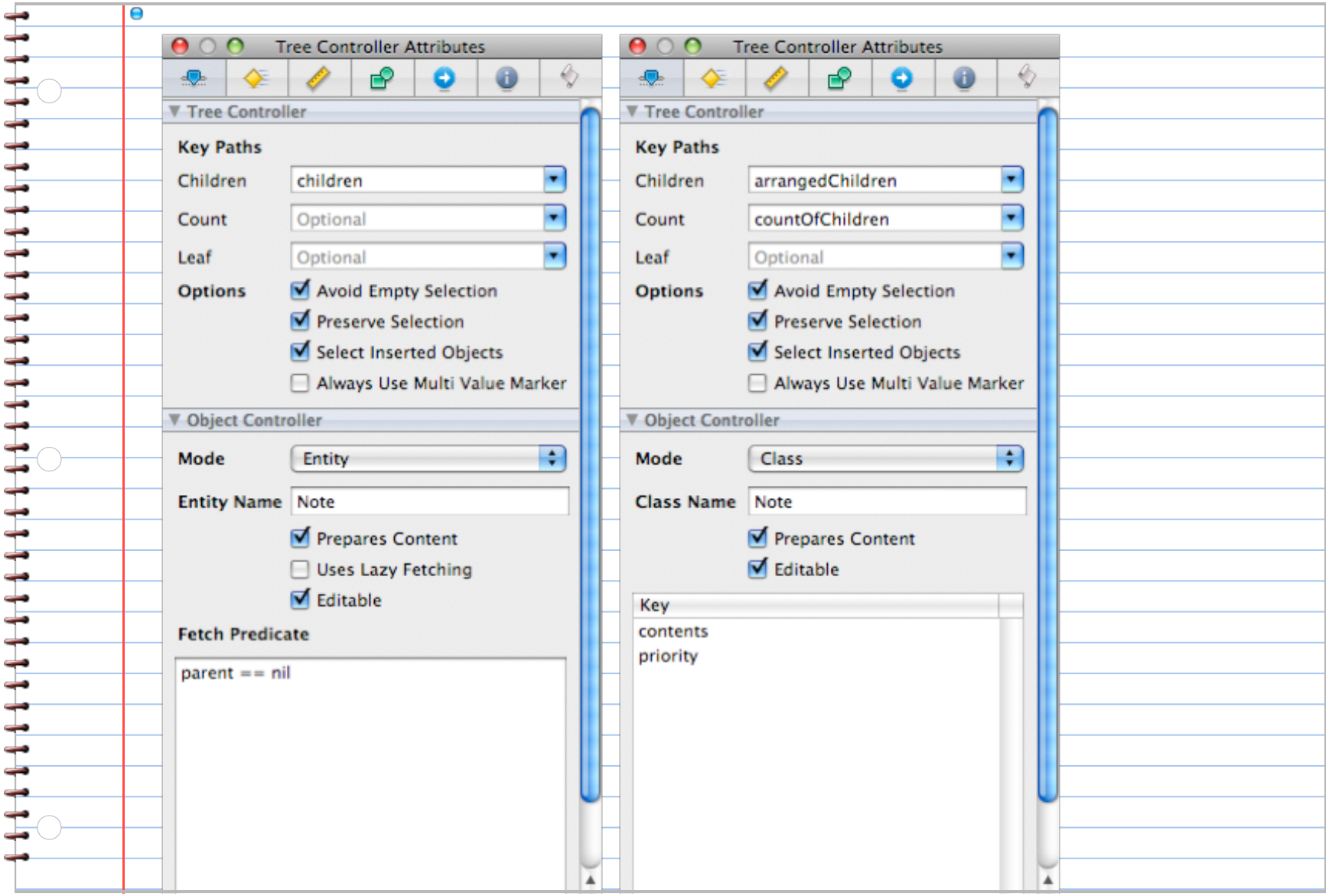
Specifying the Ordering Behavior of To-Many Relationships

- A few keys specify the ordering behavior of a to-many relationship.
-

Key	Description
transient order	Default. Specifies that the relationship does not save its order. The value is ignored.
persistent order	Specifies that the relationship save its order as part of the saved document. The value is ignored.
sort name:	Specifies that the relationship sorts its members, and does not save its order. The value specifies the string sent as the argument to the method `-(NSArray*)sortDescriptorsForName:(NSString*)name` to the document or the containing entity instance. The expected result is an array of sort descriptors. Adopting developers must implement the method in the containing entity, or in the custom document subclass (aka MyDocument).

User Info Keys for Relationships to Specify Ordering

- Note that the SCD Framework manages the order of to-many relationships in the "Model" component of the "Model-View-Controller" design pattern. Apple's OutlineEdit example manages order in the "View" component. Compare tree controllers of OutlineEdit (left) and OutlineEdit+SCD (right), below:


Tree Controller Attributes
Tree Controller
Key Paths
Children
children
Count
Optional
Leaf
Optional
Options
Avoid Empty Selection
Preserve Selection
Select Inserted Objects
Always Use Multi Value Marker
Object Controller
Mode
Entity
Entity Name
Note
Prepares Content
Uses Lazy Fetching
Editable
Fetch Predicate
parent == nil
Tree Controller Attributes
Tree Controller
Key Paths
Children
arrangedChildren
Count
countOfChildren
Leaf
Optional
Options
Avoid Empty Selection
Preserve Selection
Select Inserted Objects
Always Use Multi Value Marker
Object Controller
Mode
Class
Class Name
Note
Prepares Content
Editable
Key
contents
priority

- Typically, array and tree controllers in "Views" of SCD-based "Models" can use the class mode instead of the entity mode, and use key paths to methods that return ordered arrays.

Specifying Other Behavior in Relationships: "new/delete"

- A special user info key specifies that a relationship can "make new" and "delete" member objects. (Some relationships should only "add" and "remove".)
-

Key	Description
new/delete	Optional. Specifies that the relationship can instantiate new objects and delete existing objects. The value is ignored.

A Special User Info Key for Relationships

Specifying Class Name for Attributes of "Undefined" Core Data Type

Key	Description
setAttributeValueClassName:	Optional, for attributes with "Undefined" data type. The value is a string that specifies the argument for the method -[NSAttributeDescription setAttributeValueClassName:]. The framework invokes the method when the document first accesses it `managedObjectModel`. See: `-[SCDPersistentDocument managedObjectModel]`

- As a convenience for developers, the SCD Framework supports the key "`setAttributeValueClassName:`". See the Apple documentation for the instance method `-[NSAttributeDescription setAttributeValueClassName:]`.
- Attributes of "Undefined" Core Data type can resort to a recommended design pattern:
  - See: Core Data Programming Guide: Non-Standard Persistent Attributes.
  - Consider an attribute named "foo" of "Undefined" Core Data type.
  - Specify its value class with the user info key "setAttributeValueClassName:" and an appropriate value (e.g., NSArray, NSColor).
  - Add a new, corresponding attribute to hold "shadow data", e.g., "fooShadowData", of Core Data type "binary".
  - In the get accessor for "foo", if the underlying Core Data value is nil, lazily instantiate a value from fooShadowData.
  - In the -willSave method, calculate and set the current value for fooShadowData from the current value of foo.
  - See the Sketch+SCD example project, and refer to the SKTGraphic class. Refer to the data model entries and accessors for
    - boundsAsNSValue uses "NSValue", and pairs with boundsShadowData
    - fillColor uses "NSColor", and pairs with fillColorData

- strokeColor uses "NSColor", and pairs with strokeColorData
- In the SCD framework, the SCDToManyArray entity's "array" attribute uses "NSArray", and in its subclass SCDPersistentToManyArray, pairs with "shadowArrayData".

The SCD Distribution

Introduction

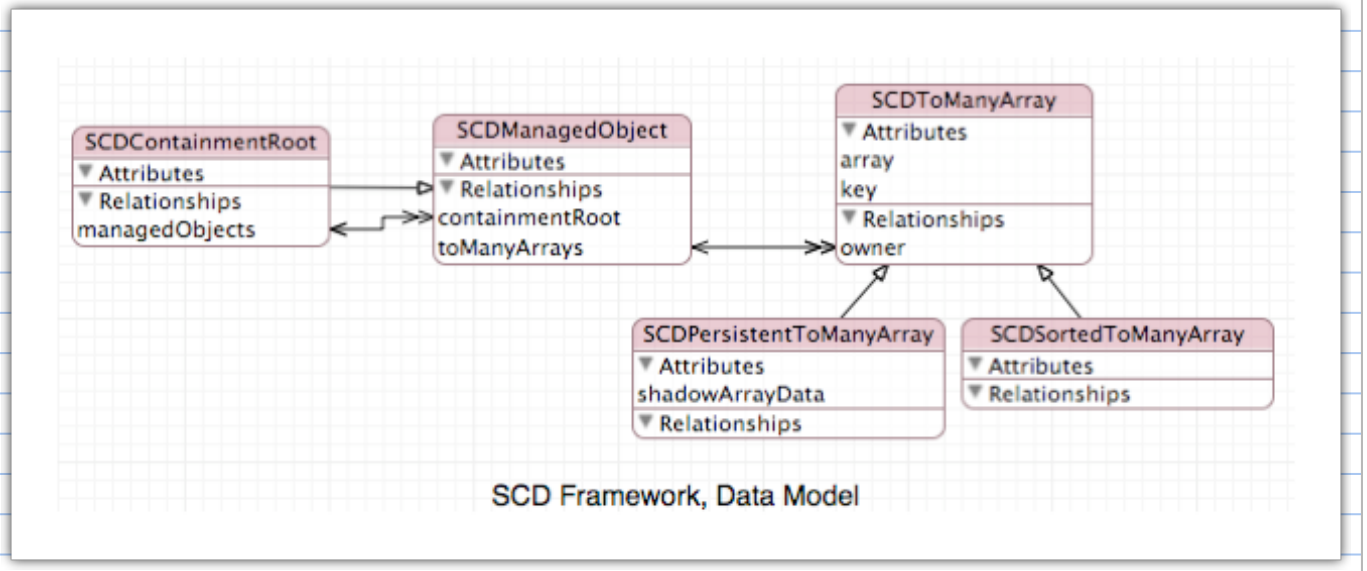
- The SCD Distribution comprises the SCD Framework itself and three example applications.
  - It provides the SCD framework as a separate stand-alone project.
  - Each example application arrives as an Xcode source code project. In each project, the application target depends on the SCD Framework target, and "embeds" the SCD framework in its Resources.
  - Adopting developers should consult
    http://developer.apple.com/documentation/MacOSX/Conceptual/BPFrameworks/Tasks/CreatingFrameworks.html
    and its section
    "Embedding a Private Framework in Your Application Bundle".
    The subsection
    "Using Separate Xcode Projects For Each Target"

The SCD Framework

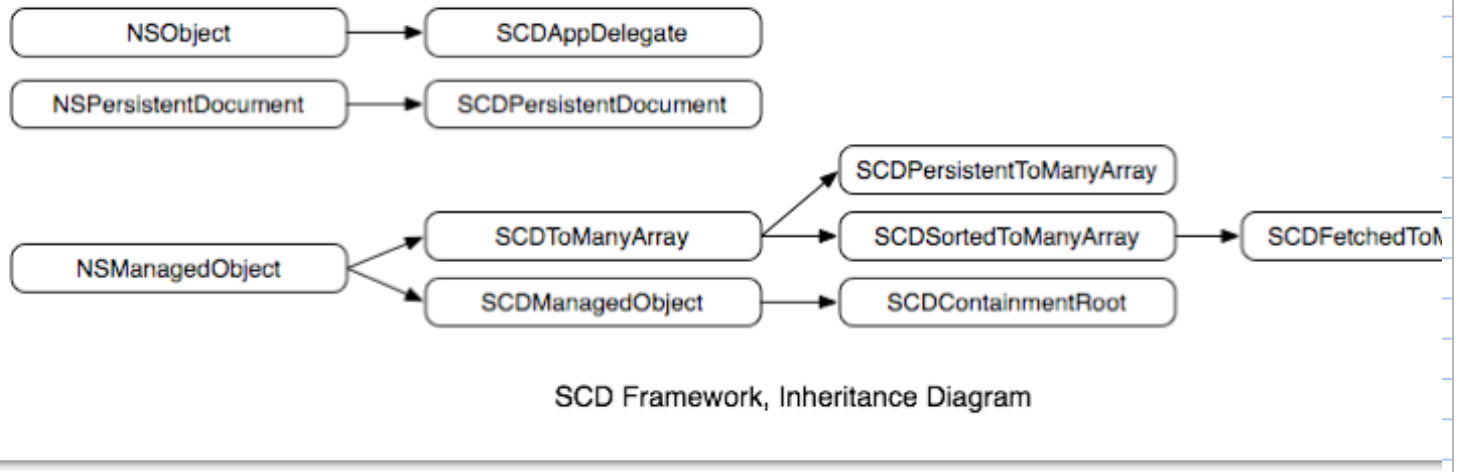
Overview of the SCD Framework

- The SCD Framework simplifies the task of AppleScript enabling Core Data Document-based applications. The adopting developer embeds the SCD Framework in the target application, and adds a few entities from the framework's data model to the project's data model.

The SCD Data Model and Its Entity Classes

- The SCD Framework's Data Model:
- 

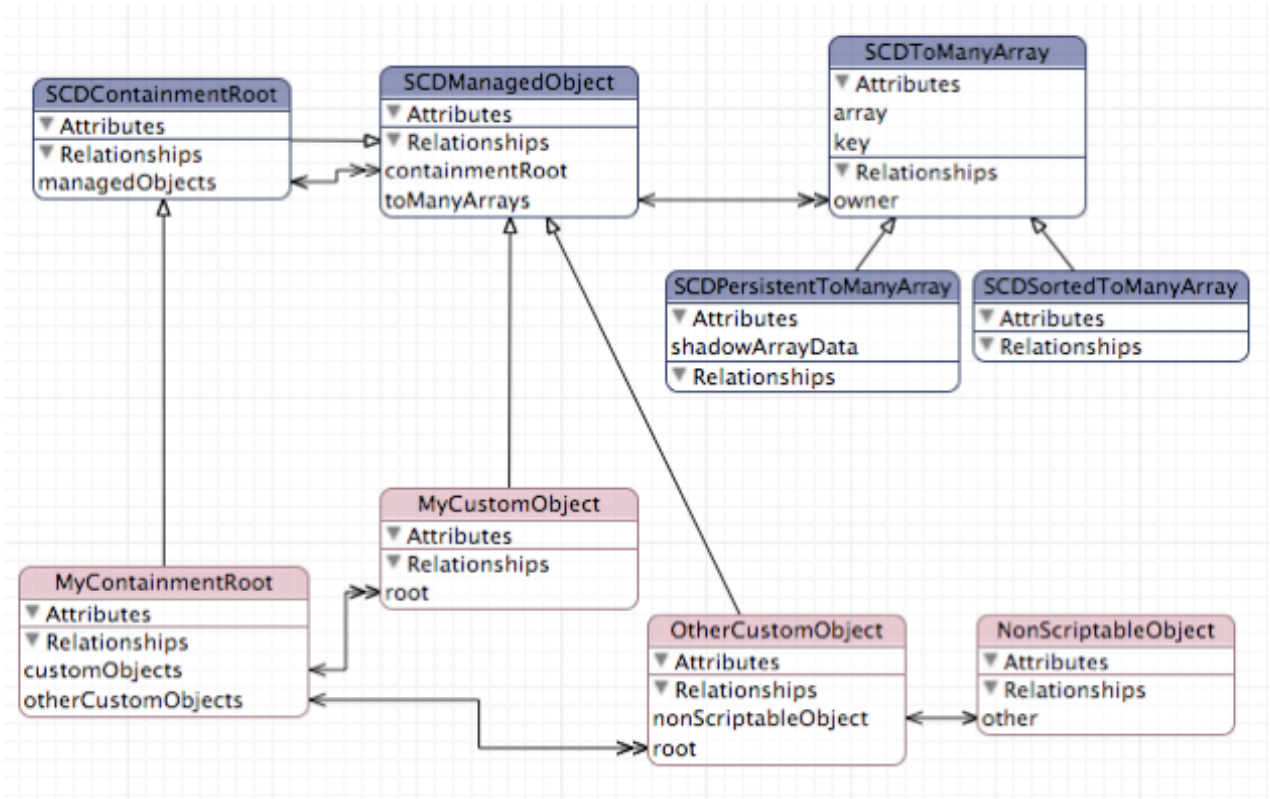
- The inheritance diagram of the primary classes of the SCD Framework:



SCD Framework, Inheritance Diagram

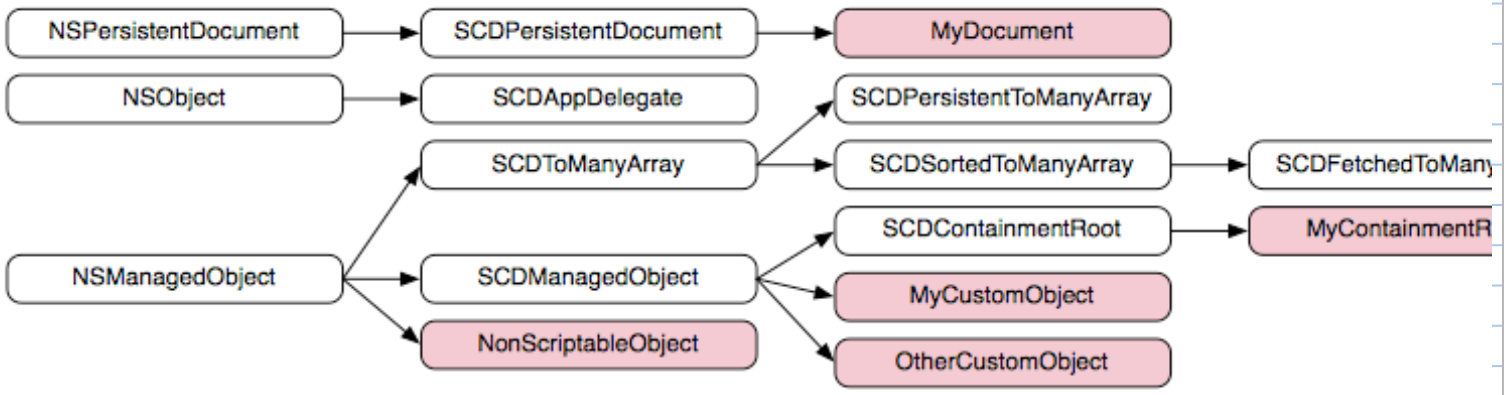
A Prototype Custom Data Model with Custom Subclasses

- A Data Model Prototype Using the SCD Framework:



A Data Model Prototype Using the SCD Framework

The corresponding inheritance diagram of the main classes for this prototype Data Model:



The Inheritance Diagram for the Prototype Core Data Document-based Application

The SCDToMany Class Cluster

- The SCD Framework uses the SCDToManyArray Class Cluster to manage order among the members of its to-many relationships. An instance of SCDManagedObject maps each of its to-many relationships to an instance of SCDToManyArray (or a subclass). The choice of ordering key in the relationship's "user info" determines at run time which class of the cluster to instantiate. The SCDToManyArray class cluster comprises:
  - SCDToManyArray, the parent entity of the cluster. The parent class has:
    - a to-one relationship to its "owner," an instance of SCDManagedObject.
    - two attributes: its key, which is a string, and its array. The array attribute's data type is "Undefined" as far as Core Data's persistence is concerned, but its "attribute value class name" is "NSArray." The key identifies a to-many relationship in the owner. The array manages the order of the members of the relationship.
  - SCDPersistentToManyArray adds an attribute called "shadowArrayData" of type "binary." In its -willSave method, the receiver builds an array of (permanent) URIs that reflects the order of the members of its array, and then archives the array as NSData. This action saves the order of the members of the owner's to-many relationship with the receiver's key, as part of the action of saving the document's graph of objects. When the document is re-opened and rebuilds its object graph, the receiver lazily unpacks its stored, ordered URIs, and recovers the order of the members of its relationship.
  - SCDSortedToManyArray sorts the members of its relationship using an array of sort descriptors.

Dynamic sdef Generation

- The SCD Framework provides the `SDefGenerating` category on `NSManagedObjectModel`. It is a rather rich category, but it couples only loosely to the rest of the framework.
- The -[SCDAppDelegate sdefData] method calls a method in the category with its managed object model as the receiver.

The SCD Example Application Projects

Introduction

- The SCD Framework Distribution includes three sample application source code projects.
  - Bricks+SCD, the bootstrap demo
  - OutlineEdit+SCD, derived from Apple's OutlineEdit example
  - Sketch+SCD, derived from Apple's Sketch example

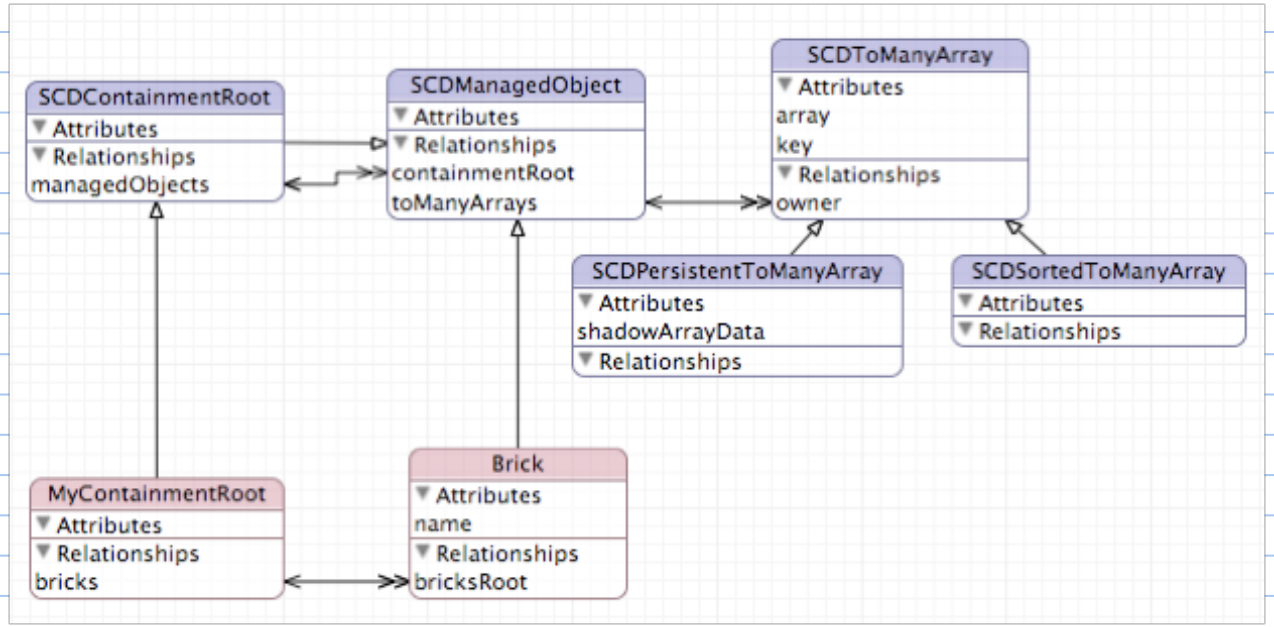
Bricks+SCD

Overview

- Bricks+SCD was the first application written under the SCD Framework. It was intended as a very simple demonstration of the framework. In fact, the framework and the Brick+SCD application shared the same development cycle. Its Data Model is "dumb as a brick."

Data Model

- The Data Model for Bricks+SCD has four entities from the Data Model in the SCD Framework, simply copied and pasted. It also has the required custom subclass of SCDContainmentRoot. Beyond those required entities, it has just one custom entity, "Brick." The Brick entity has just one attribute, "name." MyContainmentRoot has a to-many to Brick, and the inverse relationship is a to-one to the containment root.
-



- The document's containment root has a to-many to "bricks," and its members have persistent order.

Entity	Abs	Class
Brick	☐	Brick
MyContainmentRoot	☐	MyContainme
SCDContainmentRoot	☐	SCDContainm
SCDManagedObject	☑	SCDManaged
SCDPersistentToMany	☐	SCDPersistent
SCDSortedToManyArr	☐	SCDSortedTo
SCDToManyArray	☐	SCDToManyA

Prope	Kind	Type or
bricks	Relationship	Brick

Relationship

User Info:

Key	Value
new/delete	
persistent order	
sdef description	The document's

Try setting the order to "sort name:" with the value "by name", re-compile and run again.

Entity	Abs	Class
Brick	☐	Brick
MyContainmentRoot	☐	MyContainme
SCDContainmentRoot	☐	SCDContainm
SCDManagedObject	☑	SCDManaged
SCDPersistentToMany	☐	SCDPersistent
SCDSortedToManyArr	☐	SCDSortedTo
SCDToManyArray	☐	SCDToManyA

Prope	Kind	Type or
bricks	Relationship	Brick

Relationship

User Info:

Key	Value
new/delete	
sdef description	The document's
sort name:	by name

Test Scripts

- makeMonths.applescript makes a new Bricks+SCD document and populates it with twelve bricks named for the months of the year. The order is important. Save and re-open the resulting document. The order is preserved.
- makePotatoes.applescript and makeSoflage.applescript are similar.
- helloBricks+SCD.applescript uses the existing front document and makes a new brick that shows the current data and time.
- sortSafePeekAndPokeAtNamesAndUniqueIDsAndPokeAtNames.applescript renames the bricks of the existing front document in a sequence with array indexes.

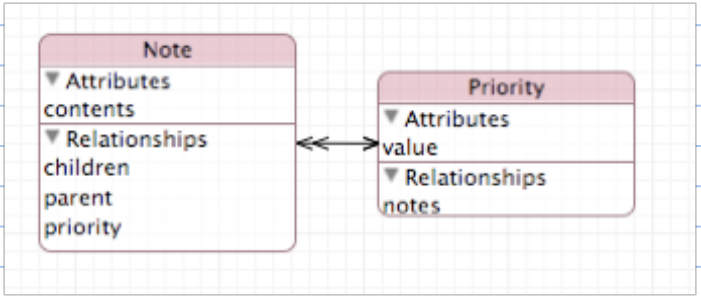
OutlineEdit+SCD

Overview

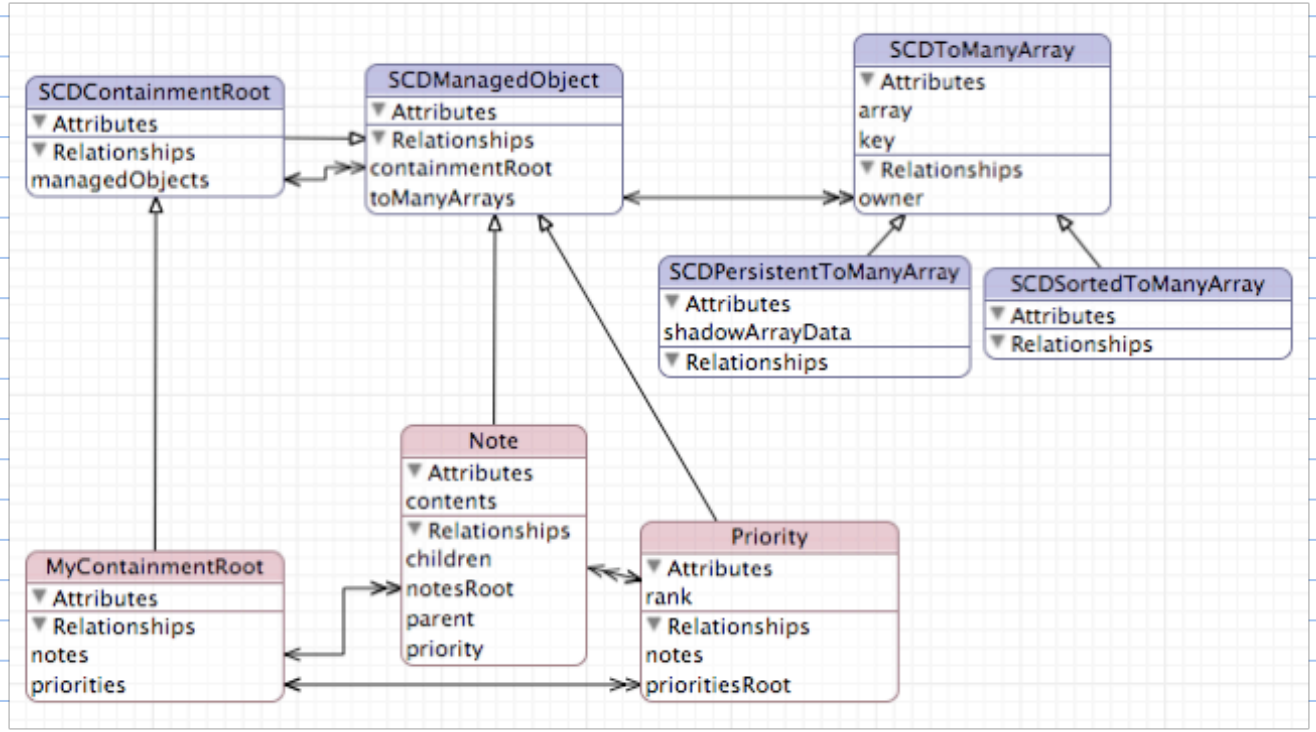
- OutlineEdit+SCD derives from the Core Data example project, OutlineEdit. It demonstrates how to add the SCD Framework to an existing Core Data Document-based application. Apple's OutlineEdit example does not preserve the order of its "Notes" and therefore fails in its nominal purpose, to edit outlines. OutlineEdit+SCD includes the SCD Framework, but makes only minor changes to the example source code. The main changes are are to the Data Model, where the SCD base classes are added, and the Note and Priority entities inherit from SCDManagedObject instead of NSManagedObject. Annotations in the Data Model specify persistent order for the document's notes and priorities, and for each Note's children.

Data Model

- Here's the Data Model of Apple's OutlineEdit example (the "Before" shot):
-



- Here's the Data Model of the OutlineEdit+SCD example (the "After" shot). The blue entities come from the SCD Framework. The Note and Priority entities both have new to-one relationships to MyContainmentRoot, and the inverse relationships are "to-many."



Of course, we want to preserve the order of the "Note" instances in the document, so we specify the order of MyContainmentRoot's "notes" relationship:

Entity	Abs	Cla
MyContainmentRoot	☐	MyCo
Note	☐	Note
Priority	☐	Priori
SCDContainmentRoot	☐	SCDC
SCDManagedObject	☑	SCD
SCDPersistentToMany	☐	SCDP
SCDSortedToManyArr	☐	SCDS
SCDToManyArray	☐	SCDT

Property	Kind	Type o
notes	Relationship	Note
priorities	Relationship	Priority

Relationship

User Info:

Key	Value
new/delete	
persistent order	

Test Scripts

- peekAtPriorities.applescript acts on an empty document to count and "get" the priorities.
- writeLimerick.applescript makes a new document and fills it with a limerick. It assigns a priority to each of the five notes, reflecting the poetic structure.
- probePriorities.applescript acts on the document that results from the script above, and sets new "rank" values for those priorities.

Sketch+SCD

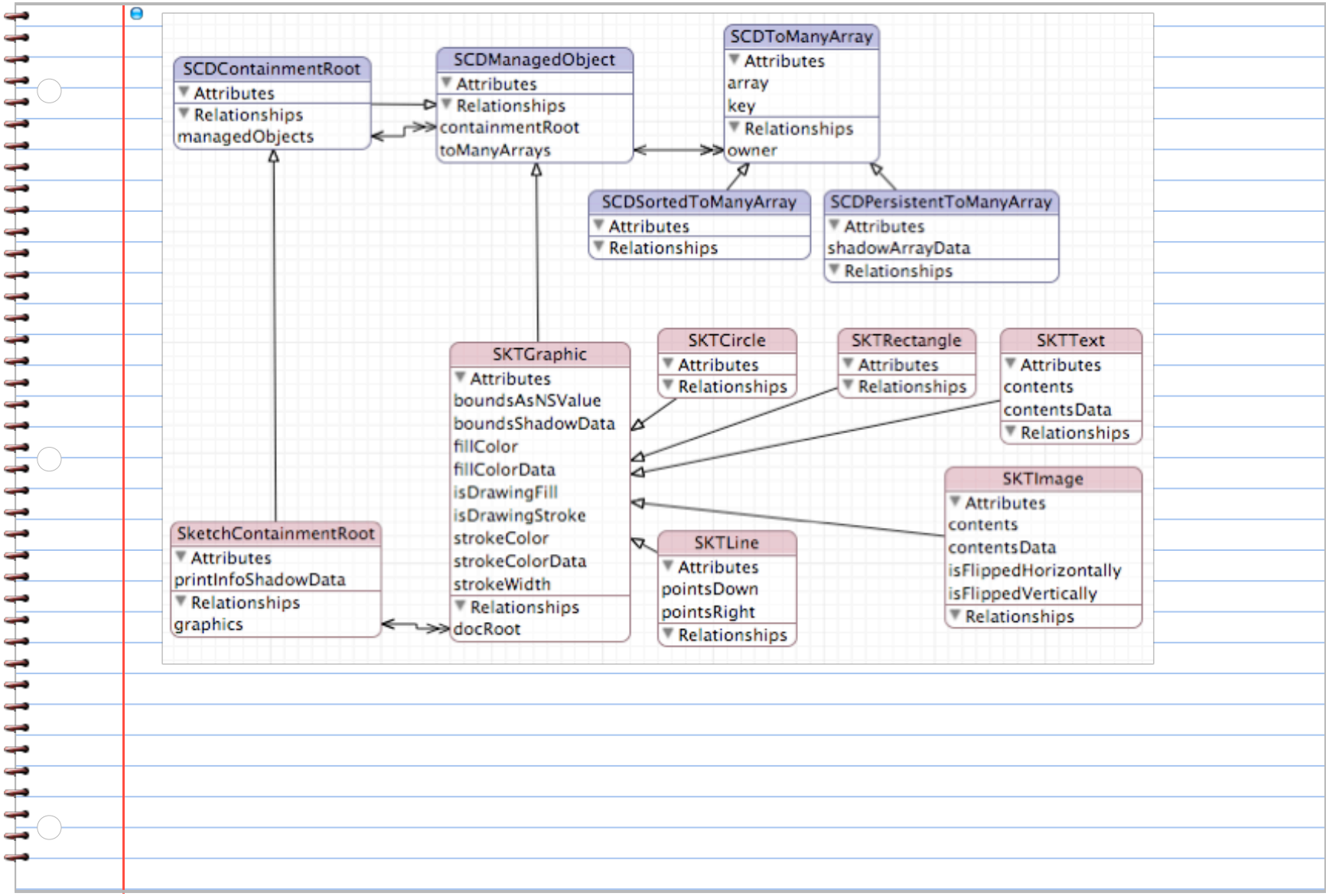
Overview

- The Sketch+SCD example derives from the venerable Sketch example (née Draw and Draw2, from the Age of NeXTSTEP). Sketch+SCD demonstrates how an existing Cocoa application can be revised for Core Data persistence, yet retain its AppleScript Dictionary.
- In Sketch, the primary class of the "Model" is SKTGraphic. Its core instance variable, "bounds", is of type NSRect. The Sketch+SCD version uses an NSValue to wrap the NSRect. The corresponding Core Data attributes are "boundsAsNSValue" (transient) and "boundsAsShadowData" (binary). The AppleScript Dictionary for Sketch suggests separate numeric attributes in the Data Model for the x and y positions, the width, and the height. But, for this demonstration, it seemed good to adapt the Core Data Model to the Cocoa implementation, and to keep it as familiar as possible.
- In the original Sketch project, the SKTGraphicView class adds and removes itself as an observer to instances of SKTGraphic and subclasses, as the instances are added to or removed from the graphics array. Initially, that led to trouble with Core Data's notions of object life cycle. For instance, the "Revert to Saved" menu action croaked. And in my misery, I added behavior and API to the SCD Framework to manage KVO per relationship, respecting the Core Data object life cycle. See the section "Make the Pain Go Away!"
- I've added the alpha channel to the colors. But, it seems AppleScript neglects the alpha channel. That's a bit sad.
- I fixed a glitch in SKTImage. In Apple's Sketch example, SKTImage can't save some kinds of images.
  - Build and run Sketch from the Leopard distribution.
  - Make a new Sketch document.
  - Drag in the "Rectangle.tiff" file from the project's source code directory (used in the Sketch application's Resources, for its tool window).
  - Save and re-open the document.
  - The document appears blank. Oops!
  - Select all.
  - The image appears in the right place and with the right size, but is empty.

- See the `-[SKTAppDelegate sdefData]` method. It overrides `SCDAppDelegate's` method to return a different sdef at run time. Developers can readily modify the code and recompile to achieve three distinct results:
  - For the value of `NSString *target` variable, choose `@"Sketch-112_After"`, and the method will return the sdef from the Sketch-112 example, minimally modified to accommodate the SCD Framework, and with a few bug fixes.
  - For the value of `NSString *target` variable, choose `@"Sketch-112_AfterPlusCommands"`, and the method will return the sdef above, with a few added commands. The additional commands might prove generally useful for SCD-based data models and extended sdefs.
  - Turn the whole method into a comment, and the `SCDAppDelegate` method will return the sdef generated by the SCD framework. It permits access to the object graph, but the only AppleScript enabled attribute is the stroke width.

Data Model

- Here's the Data Model for Sketch+SCD. The major features:
  - The familiar group of entities from the SCD Framework Data Model, copied and pasted.
  - A custom subclass of SCDContainmentRoot, SketchContainmentRoot
    - with a to-many to SKTGraphic
  - The central custom entity, SKTGraphic
    - with a to-one to SketchContainmentRoot, and the obvious inverse
    - with attributes
      - boundsAsNSValue, transient and undefined, with a User Info entry, "setAttributeValueClassName:" = "NSValue"
      - boundsShadowData, for saving the bounds information as binary data
      - fillColorData and strokeColorData, for saving the colors as binary data
      - otherwise, it's a faithful Core Data representation of the original SKTGraphic class in Sketch
  - The subclasses
    - SKTText and SKTImage have contentsData for saving contents as binary data


SCDContainmentRoot
Attributes
Relationships
managedObjects
SCDManagedObject
Attributes
Relationships
containmentRoot
toManyArrays
SCDToManyArray
Attributes
array
key
Relationships
owner
SCDSortedToManyArray
Attributes
Relationships
SCDPersistentToManyArray
Attributes
shadowArrayData
Relationships
SKTGraphic
Attributes
boundsAsNSValue
boundsShadowData
fillColor
fillColorData
isDrawingFill
isDrawingStroke
strokeColor
strokeColorData
strokeWidth
Relationships
docRoot
SKTCircle
Attributes
Relationships
SKTRectangle
Attributes
Relationships
SKTText
Attributes
contents
contentsData
Relationships
SKTImage
Attributes
contents
contentsData
isFlippedHorizontally
isFlippedVertically
Relationships
SKTLine
Attributes
pointsDown
pointsRight
Relationships
SketchContainmentRoot
Attributes
printInfoShadowData
Relationships
graphics

Supporting the sdef from the Sketch-112 example

- The class SKTAppDelegate overrides -sdefData from SCDAppDelegate.
  - Use the line: `NSString *target = @"Sketch-112_After";` to set the path for the bundle resource.
- The method then returns the sdef from the Sketch-112 example (stored in the application's Resources), but with a few modifications.
  - Added "`<type type="text"/>`" to a few AppleScript Classes. See the comments in the scripts for credit to the author of that fix, Steve Evangelou.
  - Added the "managed object" class (from the typical generated sdefs).
  - Hmmm. I found it necessary to change the AppleScript name (term) for the SKTRectangle class to "box" (was "rectangle"). There's a "rectangle" type in the Standard Suite (the window's bounds property).
- The sdef from Sketch-112 declares not just graphics, but circles, lines, boxes, images, and text areas as elements of the document. The SketchDocument and SketchContainmentRoot classes add some methods to accommodate these as filtered arrays.

Test Scripts for the Sketch-112 sdef

- make circle, box, line, text area.applescript does as its name suggests: makes one of each of these graphics. In makes a new front document first.
- evangelou++.applescript is named after the author cited in the comments. This script "peeks" at the text contents of the first text area of a document. Run this script on the document evangelouTest.xml.
- swapColorsInCircles.applescript swaps the fill and stroke colors of every graphic in the existing front document. AppleScript neglects the alpha components, though. Run this script on the document swapColorsInCircles.xml.
- stepInstancesOfClasses.applescript changes the x and y positions of each graphic in the document, depending on the class of the graphic.

Extending the sdef from the Sketch-112 example with SCD Commands

- The class SKTAppDelegate overrides -sdefData from SCDAppDelegate.
  - Use the line: `NSString *target = @"Sketch-112_AfterPlusCommands";` to set the path for the bundle resource.
- The method then returns the sdef from the Sketch-112 example as above, but with a few added commands.
- Basic Core Suite Commands:
-

AppleScript term	initial SCD support
close	√
count	√
delete	√
duplicate	√
exists	√
get	√
make	√
move	Nope. SCD goes "object-first"
open	√
quit	√
save	√
set	√

- The proposed additional SCD commands need some review. They expose the `NSScriptKeyValueCoding` protocol to AppleScript.

Proposed AppleScript command terminology term	SCDManagedObject command handler
index of item <managed object> in elements with key <text>	`-(id)handleSCDIndexOf:(NSScriptCommand*)command;`
item at index <integer> in elements with key <text>	`-(id)handleSCDItemAtIndex:(NSScriptCommand*)command;`
insert item <managed object> at index <integer> in elements with key <text>	`-(void)handleSCDInsert:(NSScriptCommand*)command;`
relocate item at index <integer> to <integer> in elements with key <text>	`-(void)handleSCDRelocate:(NSScriptCommand*)command;`
remove item at index <integer> from elements with key <text>	`-(id)handleSCDRemove:(NSScriptCommand*)command;`
element keys	`-(id)handleSCDElementKeys:(NSScriptCommand*)command;`

Test Scripts for the Sketch-112 sdef Extended with SCD Commands

- test SCD commands.applescript performs a cross-check on the index of item x in elements with key "foo" command. In the Sketch+SCD application, every object in the custom graphics collection is also a member of the managed objects collection of the containment root. Run this script on a document with a large number of graphics, e.g., 544 graphics.xml. Note that the managed objects collection has only transient order. For a given document, the result of the command **set** x **to** graphic 42 remains the same across save/open operations, unless one edits the collection. The same is not true of the managed objects collection.
- test Sketch+SCD remove and insert commands.applescript changes the order of the graphics collection in a document. Run this script on "5 graphics.xml".
- test Sketch+SCD relocate command.applescript changes the reorders the graphics collection without removing and inserting objects. Run this script on "5 graphics.xml".
- test text areas again.applescript changes the text contents of each text area in a document. Run it on any document with text areas.

A Test Script using the Dynamically Generated sdef

- The primary goal of the Sketch+SCD example source code is to AppleScript enable the application with a known sdef, conventionally composed, and stored in the application's Resources. However, in the course of the development of any AppleScript enabled Core Data application using the SCD Framework, it may very well prove useful to verify that the dynamically generated sdef performs as expected. In the case of Sketch+SCD, the only scriptable attribute is the stroke width of SKTGraphic. There's not much you can do with that, except to check that it works.
- In SKTAppDelegate, select the method -[SKTAppDelegate sdefData] and turn it into a comment.
- Recompile and run.
- Build or choose a document with at least one graphic that has strokeWidth.
- Run the script `peekPokeStrokeThickness 1.applescript`.

Guidelines for the Adopting Developer

Adopting an Existing Core Data Document-based Application to use the SCD Fram

- Start with an initial working version of your Core Data Document-based application.
  - If your application manages the order of members of any to-many relationships,
    - prepare to let SCD manage it, or
    - prepare appropriate custom subclasses of SCDToManyArray
- Add the SCD Framework.
- Copy the entities and relationships from the Data Model of the SCD Framework (SCD_Model.xcdatamodel)
- Paste them into your custom Data Model
- Fix the custom document inheritance:
  - MyDocument:NSPersistentDocument becomes MyDocument:SCDPersistentDocument
- Add an application delegate that inherits from SCDAppDelegate. This adaptation impacts your nib files.
- Subclass SCDContainmentRoot to, say, MyContainmentRoot
  - its relationships should reflect your custom document's AppleScript hierarchy
  - its attributes can handle persistent attributes of your document
- In the Data Model Editor,
  - scriptable entities must inherit from SCDManagedObject
    - and for each scriptable entity, be sure to specify your custom class in the data model
  - add "sdef code" and "sdef term" to the user info of the entity
  - add "sdef code" and "sdef term" to the user info of each scriptable attribute
  - add optional user info keys and values to relationships and attributes as required
    - for to-many relationships, specify transient, persistent, or sorted order
    - for attributes of "Undefined" Core Data type, specify the attribute value class: setAttributeValueClassName:
- Run mogenerator early and often:
  - Jon Rentzsch has provided the mogenerator tool.
    - See: rentzsch.com: mogenerator: Core Data codegen

- mogenerator separates custom logic from "boilerplate" logic
  - The entity MyCustomEntity in your Data Model yields the files:
    - MyCustomEntity.h,m — a class for custom methods
    - _MyCustomEntity.h,m — a class for "boilerplate" accessors
      - and some "boilerplate" accessors
- There's a set of methods that presently don't appear in the mogenerator template. We need a special template to treat them. Jon Rentzsch has promised such a template, but it isn't here yet. See http://devworld.apple.com/documentation/Cocoa/Conceptual/ModelObjects/ModelObjects.pdf
  - "Collection Accessors

    Collection accessors follow patterns, different for sets and arrays. The patterns are described in Key-Value Coding Programming Guide, but here is a summary. Given a relationship named `<key>` :

    ■ For an array, you implement `countOf<Key>` and `objectIn<Key>AtIndex:`. You may also implement `get<Key>:range:`. If you want to support mutations, you also implement `insertObject:in<Key>AtIndex:` and `removeObjectFrom<Key>AtIndex:`. Again to improve performance, you may also implement `replaceObjectIn<Key>AtIndex:withObject:`.

    ■ For a set, you implement an `add<Key>Object:` and `remove<Key>Object:` pair, an `add<Key>:` and `remove<Key>:` pair, or both pairs. For greater efficiency, you can also implement `intersect<Key>:`."

- Compile and test your application before you enable AppleScript.
  - check for the original behavior
  - check persistent order and sorted order behavior, where applicable
- Got NSAppleScriptEnabled? Modify the Info.plist for "dynamic" sdef.
  -

```
<key>NSAppleScriptEnabled</key>
<string>YES</string>
<key>OSAScriptingDefinition</key>
<string>dynamic</string>
```

- Or in Xcode 3.1,
-

Bundle version	22
Scriptable	☑
Main nib file base name	SketchMenu
Principal class	NSApplication
OSAScriptingDefinition	dynamic

- Test the dynamically generated sdef
  - The generated sdef simply exposes your data model as a dictionary.
  - In the case of Sketch+SCD, the generated sdef is useful only as a demonstration and cross-check.
  - "Your mileage may vary."
- As necessary, override the -[SCDAppDelegate sdefData] method (see Sketch+SCD)

Observations from the Development Cycle

Core Data Models and Hierarchy

- One of the early tasks in adapting the SCD Framework comprises:
  - copy the entities and relationships from the SCD Framework's data model
  - paste them into your custom data model
  - for your scriptable custom entities, change the parent entity to SCDManagedObject.
- My experience in the SCD Framework development cycle suggests that Apple should provide an "include" mechanism in Core Data Models:
  - A developer should be able to "include" a data model from a component framework or bundle in a custom data model.
  - Declare once, deploy as required.

Scripting Definition Files—We Need a Way to Check an sdef at Design Time

- My experience in the SCD Framework development cycle suggests that Apple should provide MUCH better consistency checking for sdef files. A stand-alone application would help a great deal.

A Retrospective

- In retrospect, it might have been better to name SCDContainmentRoot differently:
  - to emphasize the delegate nature
    - SCDDocumentDelegate
    - CoreDataDelegate
    - CoreDataDocumentDelegate
  - or to emphasize the proxy nature