EOModel

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	EOAccess/EOModel.h

Class Description

An EOModel represents a mapping between a database model and a set of classes based on the entity-relationship model. The model contains a number of EOEntity objects representing the entities (tables) of the database model. Each EOEntity object has a number of EOAttribute and EORelationship objects representing the properties (columns or fields) of the entity in the database model. See EOAttribute.h and EORelationship.h for more information on attributes and relationships.

An EOModel maintains a mapping between each of its EOEntity objects and a corresponding enterprise object class for use with the database level of the Enterprise Objects Framework. You can determine the EOEntity for a particular enterprise object with the **entityForObject:** method.

An EOModel is specific to a particular database server, and stores information needed to connect to that server. This includes the name of an adaptor framework to load so that the Enterprise Objects Framework can communicate with the database. Models are stored in the file system in a manner similar to adaptor framework; see “Loading a Model File” below for more information.

Models can have relationships that reference other models in the same model group. The other models may map to different databases and types of servers.

Models are organized into model groups; see the EOModelGroup class specification for more information.

Loading a Model File

EOModels are usually loaded from model files built with the EOModeler application rather than built programmatically. EOModel files are typically stored in a project or a framework.

You use **initWithContentsOfFile:** to load an EOModel. However, loading an EOModel doesn’t have the effect of loading all of its entities. EOModel files can be quite large, so to reduce start-up time, entity definitions are only loaded as needed. This incremental model loading is possible because an EOModel actually consists of one global file, with a separate file for each entity. Models have an **.eomodeld** file wrapper (which is actually a directory), and the individual entity files within the model are in ASCII format. The global file has the name **index.eomodeld**, and it contains the connection dictionary, the adaptor name, and a list of all of the entities in the model. It is this file that gets loaded when you use

initWithContentsOfFile:. Thereafter, when an entity is loaded, EOModel posts an EOEntityLoadedNotification.

Some of the EOModel methods contain the string “TableOfContents”. An EOModel’s “table of contents” corresponds to its global **index.eomodeld** file, which is used to access the model’s entities. **index.eomodeld** is just the ASCII representation of a model’s table of contents.

Creating an EOModel Programmatically

The EOAdaptorChannel class declares methods for reading basic schema information from a relational database. You can use this information to build up an EOModel programmatically, and then enhance that model by defining extra relationships, flattening attributes, and so on. See the class description in the EOAdaptorChannel class specification for information on reading basic schema information, and see the other modeling classes’ specifications for information on creating additional attributes and relationships.

Method Types

Initializing an EOModel instance	– initWithContentsOfFile: – initWithTableOfContentsPropertyList:path:
Saving an EOModel	– encodeTableOfContentsIntoPropertyList: – writeToFile:
Loading a model’s objects	– loadAllModelObjects
Getting a model’s entities	– addEntity: – entities – entityNamed: – entityNames
Removing entities	– removeEntity: – removeEntityAndReferences:
Getting the name	– beautifyNames – name – path – setName:
Checking references	– referencesToProperty: – externalModelsReferenced
Getting an object’s entity	– entityForObject:
Setting the adaptor bundle	– adaptorName – setAdaptorName:
Setting the connection dictionary	– connectionDictionary – setConnectionDictionary:

Setting the user dictionary	– userInfo – setUserInfo:
Working with stored procedures	– addStoredProcedure: – removeStoredProcedure: – storedProcedureNames – storedProcedureNamed: – storedProcedures
Getting the model's group	– modelGroup – setModelGroup:

Instance Methods

adaptorName

– (NSString *)**adaptorName**

Returns the name of the adaptor for the receiver. This name can be used with EOAdaptor's **adaptorWithName:** class method to create an adaptor.

addEntity:

– (void)**addEntity:**(EOEntity *)*anEntity*

Adds *anEntity* to the receiver. Raises an NSInvalidArgumentException if an error occurs (for example, if *anEntity* doesn't exist, if the entity belongs to another model, or if an entity of the same name is already in the receiver).

See also: – **entities**, – **removeEntityNamed:**

addStoredProcedure:

– (void)**addStoredProcedure:**(EOStoredProcedure *)*storedProcedure*

Adds *storedProcedure* to the receiver. Raises an NSInvalidArgumentException if an error occurs (for example, if a stored procedure of the same name is already in the receiver).

See also: – **removeStoredProcedures**, – **storedProcedures**

beautifyNames

– (void)**beautifyNames**

Makes all of the receiver's named components conform to a standard convention. Names that conform to this style are all lower-case except for the initial letter of each embedded word other than the first, which is upper case. Thus, "NAME" becomes "name", and "FIRST_NAME" becomes "firstName".

See also: – **name**

connectionDictionary

– (NSDictionary *)**connectionDictionary**

Returns a dictionary containing information used to connect to the database server. The connection dictionary is the place to specify default login information for applications using the model. See the EOAdaptor class specification for more information.

encodeTableOfContentsIntoPropertyList:

– (void)**encodeTableOfContentsIntoPropertyList:**(NSMutableDictionary *)*propertyList*

Encodes the receiver into a *propertyList* representation. This method is used to get an ASCII representation of an EOModel in property list format.

See also: – **initWithTableOfContentsPropertyList:path:**

entities

– (NSArray *)**entities**

Returns an array containing the receiver's entities. Note that this method loads every entity, and thus defeats the benefits of incremental model loading.

See also: – **entityNames**

entityForObject:

– (EOEntity *)**entityForObject:**(id)*anEO*

Returns the entity associated with *anEO*, whether *anEO* is an instance of an enterprise object class, an instance of EOGenericRecord, or a fault object (see the EOFault class specification for information on faults). Returns **nil** if *anEO* has no associated entity.

entityNamed:

– (EOEntity *)**entityNamed:**(NSString *)*name*

Returns the entity named *name*, or **nil** if no such entity exists. Posts an EOEntityLoadedNotification when the entity is loaded.

See also: – **entityNames**

entityNames

– (NSArray *)**entityNames**

Returns an array containing the names of the EOModel's entities.

See also: – **entities**

externalModelsReferenced

– (NSArray *)**externalModelsReferenced**

Returns an array containing those models that are referenced by this model.

See also: – **referencesToProperty:**

initWithContentsOfFile:

– **initWithContentsOfFile:**(NSString *)*path*

Initializes a newly-allocated EOModel by reading the contents of the file named *path* as a model archive. The file specified by path can either be an old-style (**.eomodel**) or new-style (**.eomodeld**) model file. Sets the EOModel's name and path. **initWithContentsOfFile:** raises an NSInvalidArgumentException if for any reason it cannot initialize the model from the file specified by *path*.

See also: – **name**, – **path**

initWithTableOfContentsPropertyList:path:

– **initWithTableOfContentsPropertyList:**(NSDictionary *)*tableOfContents* **path:**(NSString *)*path*

Uses *tableOfContents* (which is the property list representation of an EOModel) with the file name *path* to initialize the receiver.

See also: – **encodeTableOfContentsIntoPropertyList:**

loadAllModelObjects

– (void)**loadAllModelObjects**

Loads any of the receiver's entities, stored procedures, attributes, and relationships that have not yet been loaded.

See also: – **attributes** (EOEntity), – **entities**, – **relationships** (EOEntity), – **storedProcedures**

modelGroup

– (EOModelGroup *)**modelGroup**

Returns the model group of which the receiver is a part.

See also: – **setModelGroup:**

name

– (NSString *)**name**

Returns the receiver's name.

See also: – **path**

path

– (NSString *)**path**

Returns the name of the EOModel file used to create the receiver, or **nil** if the model wasn't initialized from a file.

See also: – **name**

referencesToProperty:

– (NSArray *)**referencesToProperty:**(id)*aProperty*

Returns an array of all properties in the receiver that reference *aProperty*, whether derived attributes, relationships that reference *aProperty*, and so on. Returns **nil** if *aProperty* isn't referenced by any of the properties in the model.

See also: – **externalModelsReferenced**

removeEntity:

– (void)**removeEntity:**(EOEntity *)*name*

Removes the entity with the given *name* without performing any referential integrity checking.

See also: – **addEntity:, – removeEntityAndReferences:**

removeEntityAndReferences:

– (void)**removeEntityAndReferences:**(EOEntity *)*entity*

Removes *entity* and any attributes or relationships in other entities that reference *entity.*

See also: – **removeEntity:**, – **addEntity:**

removeStoredProcedure:

– (void)**removeStoredProcedure:**(EOStoredProcedure *)*storedProcedure*

Removes the specified stored procedure without checking to see if an entity uses it.

See also: – **addStoredProcedure:**, **storedProcedures**

setAdaptorName:

– (void)**setAdaptorName:**(NSString *)*adaptorName*

Sets the name of the receiver's adaptor to *adaptorName*.

setConnectionDictionary:

– (void)**setConnectionDictionary:**(NSDictionary *)*connectionDictionary*

Sets the dictionary containing information used to connect to the database to *connectionDictionary.* See the EOAdaptor class specification for more information on working with **setConnectionDictionary:**.

setModelGroup:

– (void)**setModelGroup:**(EOModelGroup *)*group*

Sets the model group of which the receiver should be a part.

Note: You shouldn't change an EOModel's model group after it has been bound to other models in its group.

See also: – **modelGroup**

setName:

– (void)**setName:**(NSString *)*name*

Sets the name of the receiver to *name*.

setUserInfo:

– (void)**setUserInfo:**(NSDictionary *)*dictionary*

Sets the *dictionary* of auxiliary data, which your application can use for whatever it needs. *dictionary* can only contain property list data types (that is, NSDictionaries, NSStrings, NSArrays, and NSDatas).

storedProcedureNamed:

– (EOStoredProcedure *)**storedProcedureNamed:**(NSString *)*name*

Returns the stored procedure named *name*, or **nil** if the model doesn't contain a stored procedure with the given name.

See also: – **storedProcedureNames**, – **storedProcedures**

storedProcedureNames

– (NSArray *)**storedProcedureNames**

Returns an array containing the names of all of the model's stored procedures.

See also: – **storedProcedureNamed:**, – **storedProcedures**

storedProcedures

– (NSArray *)**storedProcedures**

Returns an array containing all of the model's stored procedures. Note that this method loads each of the model's stored procedures, thus defeating the benefits of incremental model loading.

See also: – **storedProcedureNames**, – **storedProcedureNamed:**

userInfo

– (NSDictionary *)**userInfo**

Returns a dictionary of user data. You can use this to store any auxiliary information it needs.

writeToFile:

– (void)**writeToFile:**(NSString *)*path*

Saves the receiver in the directory specified by *path*. If the file specified by path already exists, a backup copy is first created (using path with a "~" character appended). As a side-effect, this method resets the current path.

writeToFile: raises an NSInvalidArgumentException on any error which prevents the file from being written.

See also: – **path**

Notifications

EOModel declares and posts the following notification.

EOEntityLoadedNotification

Posted after an EOEntity is loaded into memory.

The notification contains:

Notification Object	The entity that was loaded.