initWithName:

Setting the name	setName:
	name
	+ isValidName:
Using joins	addJoin:
	removeJoin:
	joins
Getting attributes joined on	sourceAttributes
	destinationAttributes
Getting the definition	componentRelationships
	setDefinition:
	definition
Getting the entities joined	setEntity:
	entity
	destinationEntity
Checking type	isCompound
	isFlattened
Setting to-many	setToMany:
	isToMany
Checking references	referencesProperty:
Setting the user dictionary	setUserDictionary:
	userDictionary

```
(BOOL)addJoin:(EOJoin *)aJoin
```

(NSString *)definition

Returns the data path of a flattened relationship for example ªtoDepartment. toFacilityº. If the relat
returns nil.

(NSArray *)destinationAttributes

Returns the destination EOAttributes of a simple (nonflattened) relationship. These correspond one
attributes returned by sourceAttributes. Returns nil if the relationship is flattened.

joins

(EOEntity *)destinationEntity

Returns the relationship's destination entity, which is determined by the destination entity of its joi
relationship, and by whatever ends the data path for a flattened relationship. For example, if a flatt
definition is ªtoDepartment. toFacilityº the destination entity is the

entity

(EOEntity *)entity

Returns the relationship's source entity.

destinationEntity, addRelationship: (EOEntity)

initWithName:(NSString *)name

Initializes a newly allocated EORelationship with name as its name. The EORelationship needs to
a definition before it's usable. This is the designated initializer for the EORelationship class. Retur

(BOOL)isCompound

Returns YES if the relationship has more than one join (that is, if it joins more than one pair of attr
only one join or is a flattened relationship. See ªCreating a Simple Relationshipº in the class descr
on compound relationships.

(BOOL)isFlattened

Returns YES if the relationship traverses more than two entities, NO otherwise. See ªCreating a Fl
in the class description for an example of a flattened relationship.

(NSArray *)joins

Returns all joins used by the relationship, or nil if the relationship is flattened.

sourceAttributes, destinationAttributes

(NSString *)name

Returns the relationship's name.

(BOOL)referencesProperty:aProperty

Returns YES if aProperty is an EORelationship in the relationship's data path or is an EOAttribute
relationship's joins, NO otherwise. See the class description for information of how relationships r

referencesProperty: (EOEntity)

(void)removeJoin:(EOJoin *)aJoin

Deletes aJoin from the relationship. Does nothing if the relationship is flattened.

(void)setDefinition:(NSString *)definition

Changes the relationship to a flattened relationship by releasing all of its joins and setting definitio
example ªtoDepartment. toFacilityº. If the relationship doesn't have an entity, this method does not
Flattened Relationshipº in the class description for more information on flattened relationships.

setEntity:

(void)setEntity:(EOEntity *)anEntity

Sets the entity of the relationship to anEntity. You only need to use this method when creating a fla
EOEntity's addRelationship: to associate an existing relationship with an entity.

setDefinition:

(BOOL)setName:(NSString *)name

Sets the relationship's name to name. Returns YES if successful, NO if name is already in use by a
relationship of the same entity.

Sets the dictionary of auxiliary data, which your application can use for whatever it needs. aDictio
list (that is, it must contain only NSString, NSData, NSArray, and NSDictionary objects).

(NSArray *)sourceAttributes

Returns the source EOAttributes of a simple (nonflattened) relationship. These correspond one-to-
returned by destinationAttributes. Returns nil if the relationship is flattened.

joins

(NSDictionary *)userDictionary

Returns a dictionary of user data. Your application can use this data for whatever it needs. This dic
list (that is, it contains only NSString, NSData, NSArray, and NSDictionary objects).