EOClassDescription

Inherits From: NSObject

Declared In: EOControl/EOClassDescription.h

Class Description

The EOClassDescription class provides a mechanism for extending classes by giving them access to metadata not available in the Objective-C run-time system. This is achieved as follows:

- EOClassDescription provides a bridge between enterprise objects and the metadata contained in an external source of information, such as an EOModel. It defines a standard interface for accessing the information in an external source. It also manages the registration of EOClassDescription objects in your application.
- Enterprise Objects Framework extends NSObject by adding several EOClassDescription-related methods to it. An enterprise object class can either provide its own implementation of these methods or it can accept the default implementations. These methods enable an enterprise object to take advantage of behaviors defined by the Framework, such as undo and validation. This is discussed in more detail in the section “Using EOClassDescription.”

Enterprise Objects Framework implements a default subclass of EOClassDescription, EOEntityClassDescription. EOEntityClassDescription extends the behavior of enterprise objects by deriving information about them (such as NULL constraints and referential integrity rules) from an associated model file.

How Does It Work?

As noted above, Enterprise Objects Framework implements a default subclass of EOClassDescription, EOEntityClassDescription. In the typical scenario in which an enterprise object has a corresponding model file, a particular operation (such as validating a value) results in the broadcast of an EOClassDescriptionNeeded notification. When an EOModel object receives this notification it registers the metadata (class description) for the EOEntity on which the enterprise object is based.

An enterprise object takes advantage of the metadata registered for it by using the EOClassDescription-related methods that the Framework adds to NSObject. Primary among these methods is **classDescription**, which returns the class description associated with the enterprise object. Through this class description the enterprise object has access to all of the information relating to its entity in an EOModel file.

In addition to methods that return information based on an enterprise object’s class description, the EOClassDescription-related methods that the Framework adds to NSObject include methods that are

automatically invoked when a particular operation occurs. These include validation methods and methods that are invoked whenever an enterprise object is inserted or fetched.

All of this comes together in your running application. When a user tries to perform a particular operation on an enterprise object (such as attempting to delete it), the EOEditingContext sends these validation messages to your enterprise object, which in turn (by default) forwards them to its EOClassDescription. Based on the result, the operation is either accepted or refused. For example, referential integrity constraints in your model might state that you can't delete an department object that has employees. If a user attempts to delete a department that has employees, an exception is returned and the deletion is refused.

Using EOClassDescription

For the most part, you don't need to programmatically interact with EOClassDescription. It extends the behavior of your enterprise objects transparently. However, there are two cases in which you do need to programmatically interact with it:

- When you override EOClassDescription-related NSObject methods in an enterprise object class

  These methods are used to perform validation and to intervene when enterprise objects based on EOModels are created and fetched. For objects that don't have EOModels, you can override a different set of NSObject methods; this is described in more detail in the section "Working with Objects That Don't Have EOModels."

- When you create a subclass of EOClassDescription

Overriding Methods in an Enterprise Object

As described above, Enterprise Objects Framework adds several EOClassDescription-related methods to NSObject. It's common for enterprise object classes to override the following methods to either perform validation, to assign default values (**awakeFromInsertionInEditingContext:**), or to provide additional initialization to newly fetched objects (**awakeFromFetchInEditingContext:**):

validateValue:forKey:
validateForSave
validateForDelete
validateForInsert
validateForUpdate
awakeFromInsertionInEditingContext:
awakeFromFetchInEditingContext:

For example, an enterprise object class can implement a **validateForSave** method that checks the values of **salary** and **jobLevel** properties before allowing the values to be saved to the database:

```
- (NSException *)validateForSave
{
    if (salary > 1500 && jobLevel < 2)
```

```
        return [NSException validationExceptionWithFormat:
            @"The salary is too high for that position!"];
    // pass the check on to the EOClassDescription
    return [super validateForSave];
}
```

For more discussion of this subject, see the chapter "Designing Enterprise Objects" in the *Enterprise Objects Framework Developer's Guide*, and the class specification "Extensions to NSObject."

Working with Objects That Don't Have EOModels

Although an EOModel is the most common source of an EOClassDescription for a class, it isn't the only one. Objects that don't have an EOModel can implement EOClassDescription methods directly as instance methods, and the rest of the Framework will treat them just as it does enterprise objects that have this information provided by an external EOModel.

There are a few reasons you might want to do this. First of all, if your object implements the methods **entityName**, **attributeKeys**, **toOneRelationshipKeys**, and **toManyRelationshipKeys**, EOEditingContexts can snapshot the object and thereby provide undo for it.

For example, the following code excerpt shows an implementation of **attributeKeys** for a Circle class:

```
- (NSArray *)attributeKeys {
    static NSArray *array = nil;
    if (!array)
        array = [[NSArray alloc] initWithObjects:@"radius", @"x",
          @"y", @"color", nil];
    return array;
}
```

Secondly, you might want to implement EOClassDescription's validation or referential integrity methods to add these features to your classes.

Implementing EOClassDescription methods on a per-class basis in this way is a good alternative to creating a subclass of EOClassDescription.

Creating a Subclass of EOClassDescription

You create a subclass of EOClassDescription when you want to use an external source of information other than an EOModel to extend your objects. Another possible scenario is if you've added information to an EOModel (such as in its user dictionary) and you want that information to become part of your class description—in that case, you'd probably want to create a subclass of EOEntityClassDescription.

When you create a subclass of EOClassDescription, you only need to implement the methods that have significance for your subclass.

If you're using an external source of information other than an EOModel, you need to decide how to register class descriptions, which you do by invoking the method **registerClassDescription:forClass:**. You can

either invoke **registerClassDescription:forClass:** in response to an EOClassDescriptionNeeded notification, or you can invoke it at the time you initialize your application (in other words, you can register all potential class descriptions ahead of time). The default implementation in Enterprise Objects Framework is based on responding to an EOClassDescriptionNeeded notification. When EOModel objects receive this notification, they supply a class description for the specified class by invoking **registerClassDescription:forClass:**.

EOEntityClassDescription

There are only three methods in EOClassDescription have meaningful implementations (that is, that don't either return **nil** or simply return): **invalidateClassDescriptionCache**, **registerClassDescription:forClass:**, and **propagateDeleteForObject:editingContext:**. The default behavior of the rest of the methods in Enterprise Objects Framework comes from the implementation in the EOClassDescription subclass EOEntityClassDescription. For more information, see the EOEntityClassDescription class specification.

Method Types

Managing EOClassDescriptions	+ invalidateClassDescriptionCache + registerClassDescription:forClass:
Getting EOClassDescriptions	+ classDescriptionForClass: + classDescriptionForEntityName:
Allocating new object instances	– createInstanceWithEditingContext:globalID:zone:
Propagating delete	– propagateDeleteForObject:editingContext:
Returning information from the EOClassDescription	– entityName – attributeKeys – classDescriptionForDestinationKey: – toManyRelationshipKeys – toOneRelationshipKeys – inverseForRelationshipKey: – ownsDestinationObjectsForRelationshipKey: – deleteRuleForRelationshipKey:
Performing validation	– validateObjectForDelete: – validateObjectForSave: – validateValue:forKey:

Providing default characteristics for key display
– defaultFormatterForKey:
– displayNameForKey:

Handling newly inserted and newly fetched objects
– awakeObject:fromFetchInEditingContext:
– awakeObject:fromInsertionInEditingContext:

Class Methods

aggregateExceptionWithExceptions:

+ (NSException *)**aggregateExceptionWithExceptions:**(NSArray *)*subexceptions*

Returns an NSException with the same name, reason, and userInfo dictionary of the first exception in the *subexceptions* array, but with the userInfo dictionary augmented with the list of subexceptions under the key EOAdditionalExceptionsKey.

See also: – **exceptionAddingEntriesToUserInfo:**

classDescriptionForClass:

+ (EOClassDescription *)**classDescriptionForClass:**(Class)*aClass*

Invoked by the default implementation of the NSObject method **classDescription** to return the EOClassDescription for *aClass*. It's generally not safe to use this method directly—for example, individual EOGenericRecord instances can have different class descriptions.

classDescriptionForEntityName:

+ (EOClassDescription *)**classDescriptionForEntityName:**(NSString *)*entityName*

Returns the EOClassDescription registered under *entityName*.

invalidateClassDescriptionCache

+ (void)**invalidateClassDescriptionCache**

Flushes the class description cache. Because the EOModel objects in an application supply and register EOClassDescriptions on demand, the cache continues to be repopulated as needed after you invalidate it.

You'd use this method when a provider of EOClassDescriptions (such as an EOModel) has newly become available, or is about to go away. However, you should rarely need to directly invoke this method unless you're using an external source of information other than an EOModel.

registerClassDescription:forClass:

+ (void)**registerClassDescription:**(EOClassDescription *)*description* **forClass:**(Class)*class*

Registers an EOClassDescription object for *class* in the EOClassDescription cache. You should rarely need to directly invoke this method unless you're using an external source of information other than an EOModel.

Instance Methods

attributeKeys

– (NSArray *)**attributeKeys**

Overridden by subclasses to return an array of keys for attributes of the object. Attributes contain data (such as NSNumbers and NSStrings), as opposed to pointers to other enterprise objects. EOClassDescription's implementation of this method returns **nil**.

See also: – **entityName**, – **toOneRelationshipKeys**, – **toManyRelationshipKeys**

awakeObject:fromFetchInEditingContext:

– (void)**awakeObject:**(id)*object*
fromFetchInEditingContext:(EOEditingContext *)*anEditingContext*

Overridden by subclasses to perform standard post-fetch initialization for *object* in *anEditingContext.* EOClassDescription's implementation of this method does nothing.

awakeObject:fromInsertionInEditingContext:

– (void)**awakeObject:**(id)*object*
fromInsertionInEditingContext:(EOEditingContext *)*anEditingContext*

Assigns empty arrays to to-many relationship properties of newly inserted enterprise objects. Can be overridden by subclasses to propagate inserts for the newly inserted *object* in *anEditingContext*. More specifically, if *object* has a relationship (or relationships) that propagates the object's primary key and if no object yet exists at the destination of that relationship, subclasses should create the new object at the destination of the relationship.

classDescriptionForDestinationKey:

– (EOClassDescription *)**classDescriptionForDestinationKey:**(NSString *)*detailKey*

Overridden by subclasses to return the class description for objects at the destination of the relationship identified by *detailKey*. For example, the statement:

```
[project classDescriptionForDestinationKey:@"leader"]
```

might return the class description for the Employee class. EOClassDescription's implementation of this method returns **nil**.

createInstanceWithEditingContext:globalID:zone:

– (id)**createInstanceWithEditingContext:**(EOEditingContext *)*anEditingContext*
globalID:(EOGlobalID *)*globalID* **zone:**(NSZone *)*zone*

Overridden by subclasses to allocate an object of the appropriate class in *anEditingContext*, with *globalID*, in *zone*. If the object responds to **initWithEditingContext:classDescription:globalID** subclasses should invoke that method, otherwise they should invoke **init**. Implementations of this method should return an autoreleased object. Enterprise Objects Framework uses this method to allocate new instances of objects when fetching existing enterprise objects or inserting new ones in an EODisplayGroup. EOClassDescription's implementation of this method returns **nil**.

defaultFormatterForKey:

– (NSFormatter *)**defaultFormatterForKey:**(NSString *)*key*

Returns the default NSFormatter to use when parsing values for assignment to *key*. EOClassDescription's implementation returns **nil**. EOEntityClassDescription's implementation returns an NSFormatter based on the Objective-C data type specified for *key* in the associated model file.

deleteRuleForRelationshipKey:

– (EODeleteRule)**deleteRuleForRelationshipKey:**(NSString *)*relationshipKey*

Overridden by subclasses to return a delete rule indicating how to treat the destination of the given relationship when the receiving object is deleted. For example, the class description for an Invoice object might return EODeleteRuleCascade for the relationship **lineItems**, because when an Invoice is removed from an external store, its line items should be removed also. EOClassDescription's implementation of this method returns the delete rule EODeleteRuleNullify. In the common case, the delete rule for an enterprise object is defined in its EOModel.

displayNameForKey:

– (NSString *)**displayNameForKey:**(NSString *)*key*

Returns the default string to use in the user interface when displaying *key*. By convention, lowercase words are capitalized (for example, “revenue” becomes “Revenue”), and spaces are inserted into words with mixed case (for example, “firstName” becomes “First Name”).

entityName

– (NSString *)**entityName**

Overridden by subclasses to return a unique type name for objects of this class. EOEntityClassDescription returns its EOEntity’s name. EOClassDescription’s implementation of this method returns **nil**.

See also: – **attributeKeys**, – **toOneRelationshipKeys**, – **toManyRelationshipKeys**

exceptionAddingEntriesToUserInfo:

– (NSException *)**exceptionAddingEntriesToUserInfo:**(NSDictionary *)*additions*

Returns an NSException whose userInfo dictionary has been augmented with the object and property information contained in *additions*. When exceptions are raised by certain validation methods such as **validateValue:forKey:**, this method is sent to the exception to add object and property information to the exception’s userInfo dictionary. This information is stored in the userInfo dictionary under the keys EOValidatedObjectUserInfoKey and EOValidatedPropertyUserInfoKey, respectively. The exception that’s returned by this method has the same class with the same name and reason as the original exception; the only difference is the augmented userInfo dictionary.

inverseForRelationshipKey:

– (NSString *)**inverseForRelationshipKey:**(NSString *)*relationshipKey*

Overridden by subclasses to return the name of the relationship pointing back at the receiver from the destination of the relationship specified by *relationshipKey*. For example, suppose an Employee object has a relationship called **department** to a Department object, and Department has a relationship called **employees** back to Employee. The statement:

```
[employee inverseForRelationshipKey:@"department"]
```

returns the string “employees”.

EOClassDescription's implementation of this method returns **nil**.

ownsDestinationObjectsForRelationshipKey:

– (BOOL)**ownsDestinationObjectsForRelationshipKey:**(NSString *)*relationshipKey*

Overridden by subclasses to return YES or NO to indicate whether the objects at the destination of the relationship specified by *relationshipKey* should be deleted if they are removed from the relationship (and not transferred to the corresponding relationship of another object). For example, an Invoice object owns its line items. If a LineItem object is removed from an Invoice it should be deleted since it can't exist outside of an Invoice. EOClassDescription's implementation of this method returns NO. In the common case, this behavior for an enterprise object is defined in its EOModel.

propagateDeleteForObject:editingContext:

– (void)**propagateDeleteForObject:**(id)*object*
editingContext:(EOEditingContext *)*anEditingContext*

Propagates a delete operation for *object* in *anEditingContext*, according to the delete rules specified in the object's EOModel. This method is invoked whenever a delete operation needs to be propagated, as indicated by the delete rule specified for the EOEntity's relationship key.

See also: – **deleteRuleForRelationshipKey:**

toManyRelationshipKeys

– (NSArray *)**toManyRelationshipKeys**

Overridden by subclasses to return the keys for the to-many relationship properties of the receiver. To-many relationship properties contain arrays of pointers to other enterprise objects. EOClassDescription's implementation of this method returns **nil**.

See also: – **entityName**, – **toOneRelationshipKeys**, – **attributeKeys**

toOneRelationshipKeys

– (NSArray *)**toOneRelationshipKeys**

Overridden by subclasses to return the keys for the to-one relationship properties of the receiver. To-one relationship properties are pointers to other enterprise objects. EOClassDescription's implementation of this method returns **nil**.

See also: – **entityName**, – **toManyRelationshipKeys**, – **attributeKeys**

validateObjectForDelete:

– (NSException *)**validateObjectForDelete:**(id)*object*

Overridden by subclasses to determine whether it’s permissible to delete the object. Subclasses should return **nil** if the delete operation should proceed, or an unevaluated exception containing a user-presentable (localized) error message if not. EOClassDescription's implementation of this method returns **nil**.

validateObjectForSave:

– (NSException *)**validateObjectForSave:**(id)*object*

Overridden by subclasses to determine whether the values being saved for the object are acceptable. Subclasses should return **nil** if the values are acceptable and the save operation should therefore proceed, or an unevaluated exception containing a user-presentable (localized) error message if not. EOClassDescription's implementation of this method returns **nil**.

validateValue:forKey:

– (NSException *)**validateValue:**(id *)*valueP* **forKey:**(NSString *)*key*

Overridden by subclasses to validate the value pointed to by *valueP*. Subclasses should return **nil** if the value is acceptable, or an unevaluated exception containing a user-presentable (localized) error message if not. Implementations can replace **valueP* with a converted value (for example, an EOAttribute might convert an NSString to an NSNumber). EOClassDescription's implementation of this method returns **nil**.

Notifications

The following notification is declared by EOClassDescription and posted by enterprise objects in your application.

EOClassDescriptionNeededForClassNotification

Notification Object Enterprise object class

userInfo Dictionary None

One of the EOClassDescription-related methods that Enterprise Objects Framework adds to NSObject to extend the behavior of enterprise objects is **classDescription**. The first time an enterprise object receives a **classDescription** message (for example, when changes to the object are being saved to the database), it posts EOClassDescriptionNeededForClassNotification to notify observers (by default, the associated EOModel object) that a class description is needed. The observer then locates the appropriate class description and registers it in the application.

EOClassDescriptionNeededForEntityNameNotification

Notification Object Entity name

userInfo Dictionary None

When **classDescriptionForEntityName:** is invoked for a previously unregistered entity name, this notification is broadcast with the requested entity name as the object of the notification.