EODataSource

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	EOControl/EODataSource.h

Class Description

EODataSource is an abstract class that defines a basic interface for providing enterprise objects. It exists primarily as a simple means for an EODisplayGroup or other higher-level class to access a store of objects. EODataSource defines functional implementations of very few methods; concrete subclasses, such as EODatabaseDataSource and EODetailDataSource, define working data sources by implementing the others. EODatabaseDataSource, for example, provides objects held by an EOEditingContext, while EODetailDataSource provides objects from a relationship property of a master object.

An EODataSource provides its objects all at once, with its **fetchObjects** method. **insertObject:** and **deleteObject:** add and remove individual objects, and **createObject** instantiates a new object. A few other methods provide more general information about the objects, as described below.

Creating a Subclass

The job of an EODataSource is to provide objects that share a set of properties, so that they can be managed uniformly by its client, such as an EODisplayGroup. Typically these objects are all of the same class, or share a superclass that defines the common properties managed by the client. All that's needed, however, is that every object have the properties expected by the client. For example, if an EODataSource provides Member and Guest objects, they can be implemented as subclasses of a more general Customer class, or they can be independent classes defining the same properties (**lastName**, **firstName**, and **address**, for example). You typically specify the kind of objects an EODataSource provides when you initialize it. Subclasses usually define a special **init...** method whose arguments describe the objects. EODatabaseDataSource, for example, defines **initWithEditingContext:entityName:**, which uses an EOEntity to describe the set of objects. Another subclass might use an EOClassDescription, a class or superclass for the objects, or even a collection of existing instances.

A subclass can provide two other pieces of information about its objects, using methods declared by EODataSource. First, if your subclass keeps its objects in an EOEditingContext, it should override the **editingContext** method to return that EOEditingContext. It doesn't have to use an EOEditingContext, though, in which case it can just use the default implementation of **editingContext**, which returns **nil**. Keep in mind, however, the amount of work EOEditingContexts do for you, especially when using EODisplayGroups. For example, EODisplayGroups depend on change notifications from EOEditingContexts to update changes in the objects displayed. If your subclass or its clients depend on

change notification, you should use an EOEditingContext for object storage and change notification. If you don't use one, you'll have to implement that functionality yourself. For more information, see these class specifications:

EOObjectStore
EOEditingContext
EODisplayGroup
EODelayedObserverQueue
EODelayedObserver

The other piece of information—also optional—is an EOClassDescription for the objects. Interface Builder uses an EOClassDescription to get the keys it displays in its Connections Inspector, and EODataSource uses it by default when creating new objects. Your subclass should override **classDescriptionForObjects** to return the class description if it uses one and if it's providing objects of a single superclass. Your subclass can either record an EOClassDescription itself, or get it from some other object, such as an EOEntity or from the objects it provides (through the added NSObject method **classDescription**). If it doesn't use an EOClassDescription at all it can use the default implementation, which returns **nil**.

Manipulating Objects

A concrete subclass of EODataSource must at least provide objects by implementing **fetchObjects**. If it supports insertion of new objects, it should implement **insertObject:**, and if it supports deletion it should also implement **deleteObject:**. An EODataSource that implements its own store must define these methods from scratch. An EODataSource that uses another object as a store can forward these messages to that store. For example, an EODatabaseDataSource turns these three requests into **objectsWithFetchSpecification:editingContext:**, **insertObject:**, and **deleteObject:** messages to its EOEditingContext.

Implementing Master-Detail Data Sources

An EODataSource subclass can also implement a pair methods that allow it to be used in master-detail configurations. The first method, **dataSourceQualifiedByKey:**, should create and return a new data source, set up to provide objects of the destination class for a relationship in a master-detail setup. In a master-detail setup, changes to the detail apply to the objects in the master; for example, adding an object to the detail also adds it to the relationship of the master object. The standard EODetailDataSource class works well for this purpose, so you can simply implement **dataSourceQualifiedByKey:** to create and return one of these. Once you have a detail EODataSource, you can set the master object by sending the detail a **qualifyWithRelationshipKey:ofObject:** message. The detail then uses the master object in evaluating the relationship, and applies inserts and deletes to that master object.

Another kind of paired EODataSource setup, called master-peer, is exemplified by the EODatabaseDataSource class. In a master-peer setup, the two EODataSources are independent, so that changes to one don't affect the other. Inserting into the "detail," for example, has no effect on the master object. See that class description for more information.

Method Types

Getting the objects	– fetchObjects
Inserting and deleting objects	– createObject – insertObject: – deleteObject:
Creating detail EODataSources	– dataSourceQualifiedByKey: – qualifyWithRelationshipKey:ofObject:
Getting the editing context	– editingContext
Getting the class description	– classDescriptionForObjects

Instance Methods

classDescriptionForObjects

– (EOClassDescription *)**classDescriptionForObjects**

Implemented by subclasses to return an EOClassDescription that provides information about the objects provided by the receiver. EODataSource's implementation returns **nil**.

createObject

– (id)**createObject**

Creates a new object, inserts it in the receiver's collection of objects if appropriate, and returns the object. Returns **nil** if the receiver can't create the object or can't insert it. You should invoke **insertObject:** after this method to actually add the new object to the receiver.

As a convenience, EODataSource's implementation sends the receiver's EOClassDescription a **createInstanceWithEditingContext:globalID:zone:** to create the object. If this succeeds and the receiver has an EOEditingContext, it sends the EOEditingContext an **insertObject:** message to register the new object with the EOEditingContext (note well that this does *not* insert the object into the EODataSource). Subclasses that don't use EOClassDescriptions or EOEditingContexts should override this method *without* invoking **super**'s implementation.

See also: – **classDescriptionForObjects**, – **editingContext**

dataSourceQualifiedByKey:

– (EODataSource *)**dataSourceQualifiedByKey:**(NSString *)*relationshipKey*

Implemented by subclasses to return a detail EODataSource that provides the destination objects of the relationship named by *relationshipKey*. The detail EODataSource can be qualified using **qualifyWithRelationshipKey:ofObject:** to set a specific master object (or to change the relationship key). EODataSource's implementation merely raises an NSInvalidArgumentException; subclasses shouldn't invoke **super**'s implementation.

deleteObject:

– (void)**deleteObject:**(id)*anObject*

Implemented by subclasses to delete *anObject*. EODataSource's implementation merely raises an NSInvalidArgumentException; subclasses shouldn't invoke **super**'s implementation.

editingContext

– (EOEditingContext *)**editingContext**

Implemented by subclasses to return the receiver's EOEditingContext. EODataSource's implementation returns **nil**.

fetchObjects

– (NSArray *)**fetchObjects**

Implemented by subclasses to fetch and return the objects provided by the receiver. EODataSource's implementation returns **nil**.

insertObject:

– (void)**insertObject:**(id)*object*

Implemented by subclasses to insert *anObject*. EODataSource's implementation merely raises an NSInvalidArgumentException; subclasses shouldn't invoke **super**'s implementation.

qualifyWithRelationshipKey:ofObject:

– (void)**qualifyWithRelationshipKey:**(NSString *)*key* **ofObject:**(id)*sourceObject*

Implemented by subclasses to qualify the receiver, a detail EODataSource, to display destination objects for the relationship named *key* belonging to *sourceObject*. *key* should be the same as the key specified in the **dataSourceQualifiedByKey:** message that created the receiver. If *sourceObject* is **nil**, the receiver qualifies itself to provide no objects. EODataSource's implementation merely raises an NSInvalidArgumentException; subclasses shouldn't invoke **super**'s implementation.