EOObjectStore

Inherits From: NSObject

Conforms To: NSObject (NSObject)

Declared In: EOControl/EOObjectStore.h

Class Description

EOObjectStore is an abstract class defining the interface for an “intelligent” repository of objects, whether one based on external data or one that manages objects entirely in memory. An EOObjectStore is responsible for constructing and registering objects, servicing object faults, and saving changes made to objects. EOEditingContext is the principal subclass used for managing objects in memory—in fact, the primary purpose of the EOObjectStore interface is to service EOEditingContexts, not to define a completely general interface. The access layer’s EODatabaseContext class, a subclass of EOObjectStore, provides objects from relational databases. EODatabaseContexts, and other EOObjectStores based on external data, are often shared by several EOEditingContexts.

An EOObjectStore identifies its objects in two ways: by **id** for identification within a specific EOEditingContext, and by EOGlobalID for universal identification of the same record among multiple stores. EOObjectStores perform uniquing of their objects based on EOGlobalIDs, and use the IDs to coordinate changes among separate EOObjectStores—both within and, potentially, across applications—and between nested stores (as illustrated in the EOEditingContext class specification).

For external repositories, an EOObjectStore may delay actually fetching an object’s data, instead creating an EOFault as a placeholder. When an EOFault is accessed (sent a message), it triggers its EOObjectStore to fetch its data and transform it into an instance of the appropriate object class. This preserves both the object’s **id** and its EOGlobalID, while saving the cost of fetching data that may not be used. EOFaults are typically created for the destinations of relationships for objects that are explicitly fetched. See the EOFault and EOFaultHandler class specifications for more information.

Subclasses of EOObjectStore

As noted above, EOEditingContext is the principal subclass of EOObjectStore, used for managing objects in memory. For stores based on external data, there are several subclasses. EOCooperatingObjectStore defines stores that work together to manage data from several distinct sources (such as different databases). EODatabaseContext is actually a subclass of this class. A group of EOCooperatingObjectStores is managed by another subclass of EOObjectStore, called EOObjectStoreCoordinator. If you’re defining a subclass of EOObjectStore, it’s probably one based on an external data repository, and it should therefore inherit from EOCooperatingObjectStore so as to work well with an EOObjectStoreCoordinator—though this isn’t required.

A subclass of EOObjectStore must implement all of its methods. The default method implementations raise exceptions.

Method Types

Initializing objects	– initializeObject:withGlobalID:
Getting objects	– objectsWithFetchSpecification:editingContext: – objectsForSourceGlobalID:relationshipName:editingContext:
Getting faults	– faultForGlobalID:editingContext: – arrayFaultWithSourceGlobalID:relationshipName:editingContext: – refaultObject:withGlobalID:editingContext:
Locking objects	– lockObjectWithGlobalID:editingContext: – isObjectLockedWithGlobalID:editingContext:
Saving changes to objects	– saveChangesInEditingContext:
Invalidating objects	– invalidateAllObjects – invalidateObjectsWithGlobalIDs:

Instance Methods

arrayFaultWithSourceGlobalID:relationshipName:editingContext:

– (NSArray *)**arrayFaultWithSourceGlobalID:**(EOGlobalID *)*globalID*
relationshipName:(NSString *)*relationshipName*
editingContext:(EOEditingContext *)*anEditingContext*

Implemented by subclasses to return the destination objects for a to-many relationship, whether as real instances or as an EOFault. *globalID* identifies the source object for the relationship (which doesn't necessarily exist in memory yet), and *relationshipName* is the name of the relationship. The object identified by *globalID* and the destination objects for the relationship all belong to *anEditingContext*.

If you implement this method to return an EOFault, you must define an EOFaultHandler subclass that stores *globalID* and *relationshipName*, using them to fetch the objects in a later **objectsForSourceGlobalID:relationshipName:editingContext:** message and that turns the EOFault into an NSArray containing those objects. See the EOFaultHandler and EOFault class specifications for more information on faults.

See the EOEditingContext and EODatabaseContext class specifications for more information on how this method works in concrete subclasses.

See also: – **faultForGlobalID:editingContext:**

faultForGlobalID:editingContext:

– (id)**faultForGlobalID:**(EOGlobalID *)*globalID*
editingContext:(EOEditingContext *)*anEditingContext*

If the receiver is *anEditingContext* and the object associated with *globalID* is already registered in *anEditingContext*, returns that object. Otherwise creates a to-one fault, registers it in *anEditingContext*, and returns the fault. This method is always directed first at *anEditingContext*, which forwards the message to its parent object store if needed to create a fault.

If you implement this method to return an EOFault, you must define an EOFaultHandler subclass that stores *globalID*, uses it to fetch the object and turn the EOFault into that object, and initialize the object with EOObjectStore's **initializeObject:withGlobalID:**. See the EOFaultHandler and EOFault class specifications for more information on faults.

See the EOEditingContext and EODatabaseContext class specifications for more information on how this method works in concrete subclasses.

See also: – **arrayFaultWithSourceGlobalID:relationshipName:editingContext:**,
– **recordObject:globalID:** (EOEditingContext)

initializeObject:withGlobalID:

– (void)**initializeObject:**(id)*anObject*
withGlobalID:(EOGlobalID *)*globalID*
editingContext:(EOEditingContext *)*anEditingContext*

Implemented by subclasses to set *anObject*'s properties, as obtained for *globalID*. This method is typically invoked after *anObject* has been creating using EOClassDescription's **createInstanceWithEditingContext:globalID:zone:** and NSObject's **initWithEditingContext:classDescription:globalID:**, and after a fault has been fired.

See also: – **initWithEditingContext:classDescription:globalID:** (NSObject Additions),
– **awakeFromInsertionInEditingContext:** (NSObject Additions),
– **awakeFromFetchInEditingContext:** (NSObject Additions)

invalidateAllObjects

– (void)**invalidateAllObjects**

Discards the values of all objects held by the receiver and turns them into EOFaults. This causes all locks to be dropped and any transaction to be rolled back. The next time any object is accessed, its data is fetched anew. Any child object stores are also notified that the objects are no longer valid. See the EOEditingContext class specification for more information on how this method works in concrete subclasses.

This method should also post an EOInvalidatedAllObjectsInStoreNotification.

See also: – **invalidateObjectsWithGlobalIDs:**, – **refaultObject:withGlobalID:editingContext:**

invalidateObjectsWithGlobalIDs:

– (void)**invalidateObjectsWithGlobalIDs:**(NSArray *)*globalIDs*

Signals that the objects identified by the EOGlobalIDs in *globalIDs* should no longer be considered valid and that they should be turned into EOFaults. This causes data for each object to be refetched the next time it's accessed. Any child object stores are also notified that the objects are no longer valid.

See also: – **invalidateAllObjects**, – **refaultObject:withGlobalID:editingContext:**

isObjectLockedWithGlobalID:editingContext:

– (BOOL)**isObjectLockedWithGlobalID:**(EOGlobalID *)*globalID*
editingContext:(EOEditingContext *)*anEditingContext*

Returns YES if the object identified by *globalID* is locked, NO if it isn't. See the EODatabaseContext class specification for more information on how this method works in concrete subclasses.

lockObjectWithGlobalID:editingContext:

– (void)**lockObjectWithGlobalID:**(EOGlobalID *)*globalID*
editingContext:(EOEditingContext *)*anEditingContext*

Locks the object identified by *globalID*. See the EODatabaseContext class specification for more information on how this method works in concrete subclasses.

objectsForSourceGlobalID:relationshipName:editingContext:

– (NSArray *)**objectsForSourceGlobalID:**(EOGlobalID *)*globalID*
relationshipName:(NSString *)*relationshipName*
editingContext:(EOEditingContext *)*anEditingContext*

Returns the destination objects for a to-many relationship. This method is used by an array fault previously constructed using **arrayFaultWithSourceGlobalID:relationshipName:editingContext:**. *globalID* identifies the source object for the relationship (which doesn't necessarily exist in memory yet), and *relationshipName* is the name of the relationship. The object identified by *globalID* and the destination objects for the relationship all belong to *anEditingContext*.

See the EOEditingContext and EODatabaseContext class specifications for more information on how this method works in concrete subclasses.

objectsWithFetchSpecification:editingContext:

– (NSArray *)**objectsWithFetchSpecification:**(EOFetchSpecification *)*aFetchSpecification*
editingContext:(EOEditingContext *)*anEditingContext*

Fetches objects from an external store according to the criteria specified by *fetchSpecification* and returns them in an NSArray for inclusion in *anEditingContext*. If one of these objects is already present in memory, this method doesn't overwrite its values with the new values from the database. Raises an exception if an error occurs.

See the EOEditingContext and EODatabaseContext class specifications for more information on how this method works in concrete subclasses.

refaultObject:withGlobalID:editingContext:

– (void)**refaultObject:**(id)*anObject*
withGlobalID:(EOGlobalID *)*globalID*
editingContext:(EOEditingContext *)*anEditingContext*

Turns *anObject* into an EOFault, identified by *globalID* in *anEditingContext*. This method should be used with caution since refaulting an object doesn't remove the object snapshot from the undo stack. Objects that have been inserted but not saved, or that have been deleted, shouldn't be refaulted.

saveChangesInEditingContext:

– (void)**saveChangesInEditingContext:**(EOEditingContext *)*anEditingContext*

Saves any changes in *anEditingContext* to the receiver's repository. Sends **insertedObjects**, **deletedObjects**, and **updatedObjects** messages to *anEditingContext* and applies the changes to the receiver's data repository as appropriate. For example, EODatabaseContext implements this method to send operations to an EOAdaptor for making the changes in a database.

Notifications

EOInvalidatedAllObjectsInStoreNotification

Posted whenever an EOObjectStore receives an **invalidateAllObjects** message. The notification contains:

Notification Object	The EOObjectStore that received the **invalidateAllObjects** message.
Userinfo	None

EOObjectsChangedInStoreNotification

Posted whenever an EOObjectStore observes changes to its objects. The notification contains:

Notification Object The EOObjectStore that observed the change.

Userinfo

Key	Value
updated	An NSArray of EOGlobalIDs for objects whose properties have changed. A receiving EOEditingContext typically responds by refaulting its corresponding objects.
inserted	An NSArray of EOGlobalIDs for objects that have been inserted into the EOObjectStore.
deleted	An NSArray of EOGlobalIDs for objects that have been deleted from the EOObjectStore.
invalidated	An NSArray of EOGlobalIDs for objects that have been turned into EOFaults.