EOObjectStoreCoordinator

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

Class Description

EOObjectStoreCoordinator provides the abstraction of a single object store by directing one or more EOCooperatingObjectStores in managing objects from distinct data respositories.

EOObjectStore Methods

EOObjectStoreCoordinator overrides the following EOObjectStore methods:

objectsWithFetchSpecification:editingContext:
objectsForSourceGlobalID:relationshipName:editingContext:
propertiesForObjectWithGlobalID:editingContext:
faultForGlobalID:editingContext:
arrayFaultWithSourceGlobalID:relationshipName:editingContext:
refaultObject:withGlobalID:editingContext:
saveChangesInEditingContext:
invalidateAllObjects
invalidateObjectWithGlobalID:

With the exception of **saveChangesInEditingContext:**, EOObjectStoreCoordinator's implementation of these methods simply forwards the message to an EOCooperatingObjectStore or stores. The message **invalidateAllObjects** is forwarded to all EOCooperatingObjectStores. The rest of the messages are forwarded to the appropriate EOCooperatingObjectStore, based on which store responds YES to the messages **ownsGlobalID:**, **ownsObject:**, and **handlesFetchSpecification:** (which message is used to determine EOCooperatingObjectStore responsibility depends on the context). The EOObjectStore methods listed above aren't documented in this class specification (except for **saveChangesInEditingContext:**)—for descriptions of them, see the EOObjectStore and EODatabaseContext class specifications

For the method **saveChangesInEditingContext:**, the EOObjectStoreCoordinator guides its EOCooperatingObjectStores through a multi-pass save protocol in which each EOCooperatingObjectStore saves its own changes and forwards remaining changes to other EOCooperatingObjectStores. For example, if in its **recordChangesInEditingContext:** method one EOCooperatingObjectStore notices the removal of an object from an "owning" relationship but that object belongs to another EOCooperatingObjectStore, it

informs the other store by sending the EOObjectStoreCoordinator a **forwardUpdateForObject:changes:** message. For a more details, see the method description for **saveChangesInEditingContext:**.

Note: Although it manages objects from multiple repositories, EOObjectStoreCoordinator doesn't absolutely guarantee consistent updates when saving changes across object stores. If your application requires guaranteed distributed transactions, you can either provide your own solution by creating a subclass of EOObjectStoreCoordinator that integrates with a TP monitor, use a database server with built-in distributed transaction support, or design your application to write to only one object store per save operation (though it may read from multiple object stores). For more discussion of this subject, see the method description for **saveChangesInEditingContext:**.

Method Types

Initializing instances	– init
Setting the default coordinator	+ setDefaultCoordinator: + defaultCoordinator
Managing EOCooperatingObjectStores	– addCooperatingObjectStore: – removeCooperatingObjectStore: – cooperatingObjectStores
Saving changes	– saveChangesInEditingContext:
Communication between EOCooperatingObjectStores	– forwardUpdateForObject:changes: – valuesForKeys:object:
Returning EOCooperatingObjectStores	– objectStoreForGlobalID: – objectStoreForFetchSpecification: – objectStoreForObject:
Getting the userInfo dictionary	– userInfo – setUserInfo:

Class Methods

defaultCoordinator

+ (id)**defaultCoordinator**

Returns a shared instance of EOObjectStoreCoordinator.

setDefaultCoordinator:

+ (void)**setDefaultCoordinator:**(EOObjectStoreCoordinator *)*coordinator*

Sets a shared instance EOObjectStoreCoordinator.

Instance Methods

addCooperatingObjectStore:

– (void)**addCooperatingObjectStore:**(EOCooperatingObjectStore *)*store*

Adds *store* to the list of EOCooperatingObjectStores that need to be queried and notified about changes to enterprise objects. Posts the notification EOCooperatingStoreWasAdded.

See also: – **removeCooperatingObjectStore:**, – **cooperatingObjectStores**

cooperatingObjectStores

– (NSArray *)**cooperatingObjectStores**

Returns the receiver's EOCooperatingObjectStores.

See also: – **addCooperatingObjectStore:**, – **removeCooperatingObjectStore:**

forwardUpdateForObject:changes:

– (void)**forwardUpdateForObject:**(id)*object* **changes:**(NSDictionary *)*changes*

Tells the receiver to forward a message from an EOCooperatingObjectStore to another store informing it that *changes* need to be made to *object*. For example, inserting an object in a relationship property of one EOCooperatingObjectStore might require changing a foreign key property in an object owned by another EOCooperatingObjectStore.

This method first locates the EOCooperatingObjectStore that's responsible for applying the changes, and then it sends the store the message **recordUpdateForObject:changes:**.

See also: – **recordUpdateForObject:changes:** (EOCooperatingObjectStore)

init

– **init**

Initializes a newly allocated EOObjectStoreCoordinator and returns **self**. This is the designated initializer for the EOObjectStoreCoordinator class.

objectStoreForFetchSpecification:

– (EOCooperatingObjectStore *)**objectStoreForFetchSpecification:**
(EOFetchSpecification *)*fetchSpecification*

Returns the EOCooperatingObjectStore responsible for fetching objects with *fetchSpecification*. Returns **nil** if no EOCooperatingObjectStore can be found that responds YES to **handlesFetchSpecification:**.

See also: – **objectStoreForGlobalID:**, – **objectStoreForObject:**

objectStoreForGlobalID:

– (EOCooperatingObjectStore *)**objectStoreForGlobalID:**(EOGlobalID *)*globalID*

Returns the EOCooperatingObjectStore for the object identified by *globalID*. Returns **nil** if no EOCooperatingObjectStore can be found that responds YES to **ownsGlobalID:**.

See also: – **objectStoreForFetchSpecification:**, – **objectStoreForObject:**

objectStoreForObject:

– (EOCooperatingObjectStore *)**objectStoreForObject:**(id)*object*

Returns the EOCooperatingObjectStore that owns *object*. Returns **nil** if no EOCooperatingObjectStore can be found that responds YES to **ownsObject:**.

See also: – **objectStoreForFetchSpecification:**, – **objectStoreForGlobalID:**

removeCooperatingObjectStore:

– (void)**removeCooperatingObjectStore:**(EOCooperatingObjectStore *)*store*

Removes *store* from the list of EOCooperatingObjectStores that need to be queried and notified about changes to enterprise objects. Posts the notification EOCooperatingStoreWasRemoved.

See also: – **addCooperatingObjectStore:**, – **cooperatingObjectStores**

saveChangesInEditingContext:

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

Overrides the EOObjectStore method **saveChangesInEditingContext:** to save the changes made in *anEditingContext*. This message is sent by an EOEditingContext to an EOObjectStoreCoordinator to commit changes. When an EOObjectStoreCoordinator receives this message, it guides its EOCooperatingObjectStores through a multi-pass save protocol in which each EOCooperatingObjectStore

saves its own changes and forwards remaining changes to other EOCooperatingObjectStores. When this method is invoked, the following sequence of events occurs:

1. The receiver sends each of its EOCooperatingObjectStores the message **prepareForSaveWithCoordinator:editingContext:**, which informs them that a multi-pass save operation is beginning. When the EOCooperatingObjectStore is an EODatabaseContext, it takes this opportunity to generate primary keys for any new objects in the EOEditingContext.

2. The receiver sends each of its EOCooperatingObjectStores the message **recordChangesInEditingContext**, which prompts them to examine the changed objects in the EOEditingContext, record any operations that need to be performed, and notify the receiver of any changes that need to be forwarded to other EOCooperatingObjectStores. For example, if in its **recordChangesInEditingContext:** method one EOCooperatingObjectStore notices the removal of an object from an "owning" relationship but that object belongs to another EOCooperatingObjectStore, it informs the other store by sending the EOObjectStoreCoordinator a **forwardUpdateForObject:changes:** message.

3. The receiver sends each of its EOCooperatingObjectStores the message **performChanges**. This tells the stores to transmit their changes to their underlying databases. When the EOCooperatingObjectStore is an EODatabaseContext, it responds to this message by taking the EODatabaseOperations that were constructed in the previous step, constructing EOAdaptorOperations from them, and giving the EOAdaptorOperations to an available EOAdaptorChannel for execution.

4. If **performChanges** fails for any of the EOCooperatingObjectStores, all stores are sent the message **rollbackChanges**.

5. If **performChanges** succeeds for all EOCooperatingObjectStores, the receiver sends them the message **commitChanges**, which has the effect of telling the adaptor to commit the changes.

   If **commitChanges** fails for a particular EOCooperatingObjectStore, that EOCooperatingObjectStore and all subsequent ones are sent the message **rollbackChanges**. However, the EOCooperatingObjectStores that have already committed their changes do not roll back. In other words, the EOObjectStoreCoordinator doesn't perform the two-phase commit protocol necessary to guarantee consistent distributed update.

This method raises an exception if an error occurs.

setUserInfo:

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

Sets the *dictionary* of auxiliary data, which your application can use for whatever it needs.

See also: – **userInfo**

userInfo

– (NSDictionary *)**userInfo**

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

See also: – **setUserInfo:**

valuesForKeys:object:

– (NSDictionary *)**valuesForKeys:**(NSArray *)*keys* **object:**(id)*object*

Communicates with the appropriate EOCooperatingObjectStore to get the values identified by *keys* for *object*, so that it can then forward them on to another EOCooperatingObjectStore. EOCooperatingObjectStores can hold values for an object that augment the properties in the object. For instance, an EODatabaseContext stores foreign key information for the objects it owns. These foreign keys may well not be defined as properties of the object. Other EODatabaseContexts can find out the object's foreign keys by sending the EODatabaseContext that owns the object a **valuesForKeys:object:** message (through the EOObjectStoreCoordinator).

Notifications

The following notifications are declared and posted by EOObjectStoreCoordinator.

EOCooperatingObjectStoreWasAdded

Notification Object The EOObjectStoreCoordinator.

userInfo Dictionary None.

When an EOObjectStoreCoordinator receives an **addCooperatingObjectStore:** message and adds an EOCooperatingObjectStore to its list, it posts EOCooperatingStoreWasAdded to notify observers.

EOCooperatingObjectStoreWasRemoved

Notification Object The EOObjectStoreCoordinator.

userInfo Dictionary None.

When an EOObjectStoreCoordinator receives a **removeCooperatingObjectStore:** message and removes an EOCooperatingObjectStore from its list, it posts EOCooperatingStoreWasRemoved to notify observers.

EOCooperatingObjectStoreNeeded

Notification Object The EOObjectStoreCoordinator.

userInfo Dictionary	One of the following: globalID (globalID for operation) fetch specification (fetch specification for operation) object (object for operation)

Posted when an EOObjectStoreCoordinator receives a request that it can’t service with any of its currently registered EOCooperatingObjectStores. The observer can call back to the coordinator to register an appropriate EOCooperatingObjectStore based on the information in the userInfo dictionary.