EOAdaptorContext

Inherits From: NSObject

Declared In: EOAccess/EOAdaptorContext.h

Class at a Glance

Purpose

EOAdaptorContext is an abstract class that defines transaction handling in Enterprise Objects Framework applications. You typically don't interact with EOAdaptorContext API directly; rather, a concrete adaptor context subclass inherits from EOAdaptorContext and overrides many of its methods, which are invoked automatically by the Enterprise Objects Framework. If you're not creating a concrete adaptor context subclass, there aren't very many methods you need to use, and you'll rarely invoke them directly.

Principle Attributes

- Array of adaptor channels
- Adaptor
- Delegate

Creation

Other framework classes	Adaptor context instances are generally created automatically.
– createAdaptorContext (EOAdaptor)	Creates an adaptor context and assigns its adaptor.

Commonly Used Methods

– beginTransaction	Begins a transaction in the database server.
– commitTransaction	Commits the last transaction begun.
– rollbackTransaction	Rolls back the last transaction begun.
– setDebugEnabled:	Enables debugging in all the adaptor context's channels.

Class Description

EOAdaptorContext is an abstract class that provides concrete subclasses with a structure for handling database transactions. A concrete subclass of EOAdaptorContext provides database-specific method implementations and represents a single transaction scope (logical user) on the database server to which its EOAdaptor object is connected. You never interact with instances of the EOAdaptorContext class, rather your Enterprise Objects Framework applications use instances of concrete subclasses that are written to work with a specific database or other persistent storage system. To create an instance of a concrete EOAdaptorContext subclass, you send a **createAdaptorContext** message to an instance of the corresponding EOAdaptor subclass. You rarely create adaptor contexts yourself. They are generally created automatically by other framework objects.

If a database server supports multiple concurrent transaction sessions, an adaptor context's EOAdaptor may have several contexts. An EOAdaptorContext may in turn have several EOAdaptorChannels, which handle actual access to the data on the server. An EOAdaptorContext by default has no EOAdaptorChannels; to create a new channel send your EOAdaptorContext a **createAdaptorChannel** message.

Note: When you use multiple EOAdaptorContexts, you can have several database server transactions in progress simultaneously. You should be aware of the issues involved in concurrent access if you do this.

Controlling Transactions

EOAdaptorContext defines a simple set of methods for explicitly controlling transactions: **beginTransaction**, **commitTransaction**, and **rollbackTransaction**. Each of these messages confirms the requested action with the adaptor context's delegate, then performs the action if possible.

There's also a set of methods for notifying an adaptor context that a transaction has been started, committed, or rolled back without using the **begin-**, **commit-**, or **rollbackTransaction** methods. For example, if you invoke a stored procedure in the server that begins a transaction, you need to notify the adaptor context that a transaction has been started. Use the following methods to keep an adaptor context synchronized with the state of the database server: **transactionDidBegin**, **transactionDidCommit**, and **transactionDidRollback**. These methods post notifications.

The Adaptor Context's Delegate and Notifications

You can assign a delegate to an adaptor context. The delegate responds to certain messages on behalf of the context. An EOAdaptorContext sends these messages directly to its delegate. The transaction-controlling methods—**beginTransaction**, **commitTransaction**, and **rollbackTransaction**—notify the adaptor context's delegate before and after a transaction operation is performed. Some delegate methods, such as **adaptorContextShouldBegin:**, let the delegate determine whether the context should perform an operation. Others, such as **adaptorContextDidBegin:**, are simply notifications that an operation has occurred. The delegate has an opportunity to respond by implementing the delegate methods. If the delegate wants to intervene, it implements **adaptorContextShouldBegin:**. If it simply wants notification when a transaction has begun, it implements **adaptorContextDidBegin:**.

EOAdaptorContext also posts notifications to the application's default notification center. Any object may register to receive one or more of the notifications posted by an adaptor context by sending the message **addObserver:selector:name:object:** to the default notification center (an instance of the NSNotificationCenter class). For more information on notifications, see the NSNotificationCenter class specification in the *Foundation Framework Reference*.

Creating an EOAdaptorContext Subclass

EOAdaptorContext provides many default method implementations that are sufficient for concrete subclasses. In fact, the following methods establish structure and conventions that other Enterprise Objects Framework classes depend on and should never be overridden:

+ setDebugEnabledDefault:
– transactionDidBegin
– transactionDidCommit
– transactionDidRollback
– transactionNestingLevel

Other methods require database-specific implementations that can be provided only by a concrete adaptor context subclass. A subclass must override the following methods in terms of the persistent storage system to which it interfaces:

– beginTransaction
– canNestTransactions
– commitTransaction
– createAdaptorChannel
– rollbackTransaction

Method Types

Creating an EOAdaptorContext	– initWithAdaptor:
Getting the adaptor	– adaptor
Creating adaptor channels	– createAdaptorChannel – channels
Checking connection status	– hasOpenChannels – hasBusyChannels

Controlling transactions	– beginTransaction – commitTransaction – rollbackTransaction – transactionDidBegin – transactionDidCommit – transactionDidRollback – canNestTransactions – transactionNestingLevel
Debugging	+ setDebugEnabledDefault: + debugEnabledDefault – setDebugEnabled: – isDebugEnabled
Setting the delegate	– delegate – setDelegate:

Class Methods

debugEnabledDefault

+ (BOOL)**debugEnabledDefault**

Returns YES if new adaptor context instances have debugging enabled by default, NO otherwise. By default, adaptor contexts have debugging enabled if the user default EOAdaptorDebugEnabled is YES. (For more information on user defaults, see the NSUserDefaults class specification in the *Foundation Framework Reference*.) You can override the user default using the class method **setDebugEnabledDefault:**, or you can set debugging behavior for a specific instance with the instance method **setDebugEnabled:**.

setDebugEnabledDefault:

+ (void)**setDebugEnabledDefault:**(BOOL)*flag*

Sets default debugging behavior for new instances of EOAdaptorContext. If *flag* is YES, debugging is enabled for new instances. If *flag* is NO, debugging is disabled. Use the instance method **setDebugEnabled:** to enable debugging for a specific adaptor context.

See also: + **debugEnabledDefault**, – **isDebugEnabled**

Instance Methods

adaptor

– (EOAdaptor *)**adaptor**

Returns the receiver's EOAdaptor.

See also: – **initWithAdaptor:**

beginTransaction

– (void)**beginTransaction**

Implemented by subclasses to attempt to begin a new transaction, nested within the current one if nested transactions are supported. Invokes the delegate method **adaptorContextShouldBegin:** before beginning the transaction. If the transaction is begun successfully, sends **self** a **transactionDidBegin** message and invokes the delegate method **adaptorContextDidBegin:**. Raises if the attempt is unsuccessful. Some possible reasons for failure are:

- A connection to the database hasn't been established.
- Nested transactions aren't supported, and a transaction is already in progress.
- A fetch is in progress.
- The delegate refuses**.**
- The database server fails to begin a transaction.

An adaptor context subclass should override this method without invoking EOAdaptorContext's implementation.

See also: – **commitTransaction**, – **rollbackTransaction**, – **canNestTransactions**, – **transactionNestingLevel**

canNestTransactions

– (BOOL)**canNestTransactions**

Implemented by subclasses to return YES if the database server and the adaptor context can nest transactions, NO otherwise. An adaptor context subclass should override this method without invoking EOAdaptorContext's implementation.

See also: – **transactionNestingLevel**

channels

– (NSArray *)**channels**

Returns an array of channels created by this context. Specific adaptors have different limits on the maximum number of channels a context can have.

See also: – **createAdaptorChannel**

commitTransaction

– (void)**commitTransaction**

Implemented by subclasses to attempt to commit the last transaction begun. Invokes the delegate method **adaptorContextShouldCommit:** before committing the transaction. If the transaction is committed successfully, sends **self** a **transactionDidCommit** message and invokes the delegate method **adaptorContextDidCommit:**. Raises if the attempt is unsuccessful. Some possible reasons for failure are:

- A transaction is not in progress.
- Fetches are in progress.
- The delegate refuses.
- The database server fails to commit (and may rollback).

In the first three cases, the transaction is not rolled back. In the fourth, it depends on the concrete subclass. Some concrete adaptor contexts roll back after the database server fails to commit a transaction and others don't; see your adaptor context's documentation for more information.

An adaptor context subclass should override this method without invoking EOAdaptorContext's implementation.

See also: – **beginTransaction**, – **transactionDidCommit**, – **hasBusyChannels**

createAdaptorChannel

– (EOAdaptorChannel *)**createAdaptorChannel**

Implemented by subclasses to create and return a new EOAdaptorChannel, or **nil** if a new channel cannot be created. Initializes the new channel by sending it **initWithAdaptorContext:self**. The newly created channel retains its context. A newly created adaptor context has no channels. Specific adaptors have different limits on the maximum number of channels a context can have, and **createAdaptorChannel** may fail if a newly created channel will exceed the limits.

An adaptor context subclass should override this method without invoking EOAdaptorContext's implementation.

See also: – **channels**

delegate

– **delegate**

Returns the adaptor context's delegate.

See also: – **setDelegate:**

hasBusyChannels

– (BOOL)**hasBusyChannels**

Returns YES if any of the receiver's channels have outstanding operations (that is, have a fetch in progress), NO otherwise.

See also: – **isFetchInProgress** (EOAdaptorChannel)

hasOpenChannels

– (BOOL)**hasOpenChannels**

Returns YES if any of the receiver's channels are open, NO otherwise.

See also: – **openChannel** (EOAdaptorChannel), – **isOpen** (EOAdaptorChannel)

initWithAdaptor:

– **initWithAdaptor:**(EOAdaptor *)*adaptor*

The designated initializer for the EOAdaptorContext class, this method is overridden by subclasses to initialize a newly allocated EOAdaptorContext subclass and retain *adaptor*. Returns **self**.

You never invoke this method directly. You must use the EOAdaptor method **createAdaptorContext** to create a new adaptor context.

See also: – **adaptor**

isDebugEnabled

– (BOOL)**isDebugEnabled**

Returns YES if debugging is enabled in the receiver, NO otherwise.

See also: – **setDebugEnabled:**, + **debugEnabledDefault**, + **setDebugEnabledDefault:**

rollbackTransaction

– (void)**rollbackTransaction**

Implemented by subclasses to attempt to roll back the last transaction begun. Invokes the delegate method **adaptorContextShouldRollback:** before rolling back the transaction. If the transaction is begun successfully, sends **self** a **transactionDidRollback** message and invokes the delegate method **adaptorContextDidRollback:**. Raises if the attempt is unsuccessful. Some possible reasons for failure are:

- A transaction is not in progress.
- Fetches are in progress.
- The delegate refuses.
- The database server fails to rollback.

An adaptor context subclass should override this method without invoking EOAdaptorContext's implementation.

See also: – **beginTransaction**, – **commitTransaction**

setDebugEnabled:

– (void)**setDebugEnabled:**(BOOL)*debugEnabled*

Enables debugging in the receiver and all its channels. If *debugEnabled* is YES, enables debugging; otherwise, disables debugging.

See also: – **setDebugEnabled:** (EOAdaptorChannel), – **isDebugEnabled**, + **setDebugEnabledDefault:**, – **channels**

setDelegate:

– (void)**setDelegate:***delegate*

Sets the receiver's delegate and all the receiver's channels to *delegate*. The receiver does not retain *delegate*.

See also: – **delegate**, – **channels**

transactionDidBegin

– (void)**transactionDidBegin**

Informs the adaptor context that a transaction has begun in the database server, so the receiver can update its state to reflect this fact and send an EOAdaptorContextBeginTransactionNotification. This method is invoked from **beginTransaction** after a transaction has successfully been started. Your application should invoke this method whenever it begins a transaction other than by sending a **beginTransaction** message (for example, by using EOAdaptorChannel's **evaluateExpression:**).

transactionDidCommit

– (void)**transactionDidCommit**

Informs the adaptor context that a transaction has committed in the database server, so the receiver can update its state to reflect this fact and send an EOAdaptorContextCommitTransactionNotification. This method is invoked from **commitTransaction** after a transaction has successfully committed. Your application should invoke this method whenever it commits a transaction other than by sending a **commitTransaction** message (for example, by using EOAdaptorChannel's **evaluateExpression:**).

transactionDidRollback

– (void)**transactionDidRollback**

Informs the adaptor context that a transaction has rolled back in the database server, so the receiver can update its state to reflect this fact and send an EOAdaptorContextRollbackTransactionNotification. This method is invoked from **rollbackTransaction** after a transaction has successfully been rolled back. Your application should invoke this method whenever it rolls back a transaction other than by sending a **rollbackTransaction** message (for example, by using EOAdaptorChannel's **evaluateExpression:**).

transactionNestingLevel

– (unsigned)**transactionNestingLevel**

Returns the number of transactions in progress. If the database server and the adaptor support nested transactions, this number may be greater than 1.

See also: – **canNestTransactions**

Notifications

EOAdaptorContextBeginTransactionNotification

Sent from **transactionDidBegin** to tell observers that a transaction has begun. The notification contains:

Notification Object	The notifying EOAdaptorContext object
Userinfo	None

EOAdaptorContextCommitTransactionNotification

Sent from **transactionDidCommit** to tell observers that a transaction has been committed. The notification contains:

Notification Object	The notifying EOAdaptorContext object
Userinfo	None

EOAdaptorContextRollbackTransactionNotification

Sent from **transactionDidRollback** to tell observers that a transaction has been rolled back. The notification contains:

Notification Object	The notifying EOAdaptorContext object
Userinfo	None

Methods Implemented By the Delegate

adaptorContextDidBegin:

– (void)**adaptorContextDidBegin:***context*

Invoked from **beginTransaction** to tell the delegate that a transaction has begun.

adaptorContextDidCommit:

– (void)**adaptorContextDidCommit:***context*

Invoked from **commitTransaction** to tell the delegate that a transaction has been committed.

adaptorContextDidRollback:

– (void)**adaptorContextDidRollback:***context*

Invoked from **rollbackTransaction** to tell the delegate that a transaction has been rolled back.

adaptorContextShouldBegin:

– (BOOL)**adaptorContextShouldBegin:***context*

Invoked from **beginTransaction** to tell the delegate that *context* is beginning a transaction. If this method returns NO, the adaptor context does not begin a transaction. Return YES to allow the adaptor context to begin a transaction.

adaptorContextShouldCommit:

– (BOOL)**adaptorContextShouldCommit:***context*

Invoked from **commitTransaction** to tell the delegate that *context* is committing a transaction. If this method returns NO, the adaptor context does not commit the transaction. Return YES to allow the adaptor context to commit.

adaptorContextShouldRollback:

– (BOOL)**adaptorContextShouldRollback:***context*

Invoked from **rollbackTransaction** to tell the delegate that *context* is rolling back a transaction. If this method returns NO, the adaptor context does not roll back the transaction. Return YES to allow the adaptor context to roll back.