initWithDatabase:
Getting the adaptor context adaptorContext
Getting the database object database
Finding open channels hasBusyChannels
Controlling transactions beginTransaction
commitTransaction
rollbackTransaction
Notifying of other transactions transactionDidBegin
transactionDidCommit
transactionDidRollback
Nesting transactions canNestTransactions
transactionNestingLevel
Setting the update strategy setUpdateStrategy:
updateStrategy
Uniquing/snapshotting keepsSnapshots
forgetObject:
objectForPrimaryKey:entity:
recordObject:primaryKey:snapshot:
snapshotForObject:

(BOOL)beginTransaction

Attempts to begin a new transaction, nested within the current one if nested transactions are suppo
successful and NO if not (specifically, if nested transactions aren't supported).

transactionDidBegin, canNestTransactions, transactionNestingLevel, commitTransaction, rollba

(BOOL)canNestTransactions

Returns YES if the database server can nest transactions, NO otherwise.

transactionNestingLevel

(BOOL)commitTransaction

Attempts to commit the last transaction begun, returning YES if successful and NO if not.

When an EODatabaseContext commits the top-level transaction (so that the nesting level becomes
snapshots of any updated objects back into its EODatabase.

transactionDidCommit, rollbackTransaction, beginTransaction, transactionNestingLevel

(EODatabase *)database

Returns the EODatabase that the context works with.

initWithDatabase:

(void)forgetObject:anEO

Removes anEO from the EODatabaseContext's uniquing tables and destroys its snapshot with resp
EODatabaseContext. Doesn't affect equivalent information kept by the EODatabase until the top-l
committed then, the EODatabaseContext sends the EODatabase a forgetObject: message for anEO
NSInvalidArgumentException if anEO is nil or if there's no transaction in progress, and raises
NSInternalInconsistencyException if anEO is a fault (see the EOFault class specification for inform

forgetObject: (EODatabase)

(BOOL)hasBusyChannels

Returns YES if the receiver has any EODatabaseChannels with fetches in progress, NO otherwise.

isFetchInProgress (EODatabaseChannel)

(BOOL)keepsSnapshots

Returns NO if the EODatabaseContext's locking strategy is EONoUpdate and the parent EODatab snapshots returns YES otherwise. EODatabaseContexts nearly always keep local snapshots.

keepsSnapshots (EODatabase)

objectForPrimaryKey:(NSDictionary *)aKey entity:(EOEntity *)anEntity

Returns the unique enterprise object for aKey and anEntity, or nil if one doesn't exist.

objectForPrimaryKey: (EODatabase)

(void)recordObject:anEO
primaryKey:(NSDictionary *)aKey
snapshot:(NSDictionary *)aSnapshot

Records anEO and aSnapshot under aKey with respect to the EODatabaseContext. Raises NSInval under any of the following conditions:

·No transaction is in progress.
·When uniquing and either anEO or aKey is nil.
·When uniquing and anEO is an EOFault.
·When context keeps snapshots and aSnapshot is nil.

There may already be a unique instance recorded for anEO's key, so you shouldn't expect that the this method can be shared. To get the unique enterprise object created for anEO, use objectForPrin

Changes to an EODatabaseContext's snapshots are folded back into its parent EODatabase when t is committed.

recordObject:primaryKey:snapshot: (EODatabase), commitTransaction

(BOOL)rollbackTransaction

Attempts to roll back the last nested transaction in progress, returning YES if successful and NO if

transactionDidRollback, commitTransaction, beginTransaction

(void)setUpdateStrategy:(EOUpdateStrategy)strategy

Set the update strategy used by the EODatabaseContext to strategy. See the class description for in strategies. Raises NSInvalidArgumentException if the context has any transactions in progress.

transactionNestingLevel

(NSDictionary *)snapshotForObject:anEO

Returns the snapshot associated with anEO, if there is one otherwise returns nil.

other than by sending the database context a beginTransaction message (for example, by using EO
evaluateExpression:).

(void)transactionDidCommit

Informs the EODatabaseContext and its EOAdaptorContext that the server has committed a transa
the result of a stored procedure). This method allows the EODatabaseContext to maintain a valid s
server. Your application should invoke this method whenever it commits a transaction in the datab
other than by sending the database context a commitTransaction message (for example, by using E
evaluateExpression:).

When an EODatabaseContext commits the top-level transaction (so that the nesting level becomes
the context's snapshots are folded back into the context's EODatabase.

(void)transactionDidRollback

Informs the EODatabaseContext and its EOAdaptorContext that the database has rolled back a tran
as the result of a stored procedure). This method allows the EODatabaseContext to maintain a vali
the server. Your application should invoke this method whenever it rolls back a transaction in the d
way other than by sending a rollbackTransaction message (for example, by using EOAdaptorChan
evaluateExpression:). It should also invoke this method when a transaction is rolled back by a trigg
or by the server itself due to an error condition such as a deadlock.

(unsigned int)transactionNestingLevel

Returns the number of transactions in progress. If the EODatabaseContext's adaptor supports neste
number may be greater than 1.

canNestTransactions

(EOUpdateStrategy)updateStrategy

Return the update strategy used by the EODatabaseContext. The default strategy is EOUpdateWith
See the class description for information on update strategies.