The ODBCEOAdaptor Framework

Framework:	System/Library/Frameworks/ODBCEOAdaptor.framework
Header File Directories:	System/Library/Frameworks/ODBCEOAdaptor.framework/Headers

Introduction

The ODBCEOAdaptor framework is a set of classes that allow your programs to connect to an ODBC server. These classes provide ODBC-specific method implementations for the EOAccess framework's EOAdaptor, EOAdaptorChannel, EOAdaptorContext, and EOSQLExpression abstract classes.

ODBC (Open Data Base Connectivity) defines a standard interface that Windows applications can use to access any data source. Unlike the other Enterprise Objects Frameworks adaptors that support a single type of database, the ODBC adaptor supports any data source that has an ODBC driver. Consequently, in addition to having standard adaptor features, the ODBC adaptor also manages information relating to the driver and to the data types defined by the data source the driver supports.

The following table lists the classes in the ODBCEOAdaptor Framework and provides a brief description of each class.

Class	Description
ODBCAdaptor	Represents a single connection to a ODBC database server, and is responsible for keeping login and model information, performing ODBC-specific formatting of SQL expressions, and reporting errors.
ODBCChannel	Represents an independent communication channel to the database server its ODBCAdaptor is connected to.
ODBCContext	Represents a single transaction scope on the database server to which its adaptor object is connected.
ODBCSQLExpression	Defines how to build SQL statements for ODBCChannels.

The Connection Dictionary

The connection dictionary contains items needed to connect to an ODBC server, such as the data source (it's common to omit the user name and password from the connection dictionary, and prompt users to enter those values in a login panel). The keys of this dictionary identify the information the server expects, and the values of those keys are the values that the adaptor uses when trying to connect to the server. For ODBC the required keys are as follows:

dataSource
userName
password

The connection dictionary can also optionally have the keys connectionString, typeInfo, and driverInfo.

The connectionString contains the user name, password, and data source. If the connectionString key is present in the connection dictionary, the other login keys are ignored and this string is used to connect to the database.

The typeInfo key refers to the typeInfo dictionary, which is used to cache type information in the connection dictionary. This is done because different ODBC drivers work with different data types. Caching type information in the connection dictionary avoids costly connections to the driver and the database. The typeInfo dictionary contains the following information for every type in your database:

```
defaultODBCType = (<CHAR/TIMESTAMP/BIT/...>, ...)
precision = <precision>
minScale = <minScale>
maxScale = <maxScale>
isUnsigned = <YES/NO>
isNullable = <YES/NO>
isSearchable = <YES/NO>
createParams = <0/1/2>
```

Likewise, the driverInfo key refers to the driverInfo dictionary, which stores information about the driver, such as its name and version. This information is also cached in the connection dictionary.

Locking

All adaptors use the database server's native locking facilities to lock rows on the server. If you’re using the Microsoft SQL Server, the ODBC adaptor locks a row by using the HOLDLOCK keyword in SELECT statements. In all other cases it uses the SELECT... FOR UPDATE... statement. Locking occurs when:

- You send the adaptor channel a **selectAttributes:fetchSpecification:lock:entity:** message with YES specified as the value for the **lock:** keyword.
- You explicitly lock an object’s row with the EODatabaseContext’s **lockObjectWithGlobalID: editingContext:** message.
- You set pessimistic locking at the database level and fetch objects.

Data Type Mapping

Every adaptor provides a mapping between each server data type and the Objective-C type to which a database value will be coerced when it's fetched from the database. ODBC adds an intermediate layer: the generic ODBC type (identifier) to which each database data type maps.

For example, the following table shows the mapping from some of the Microsoft Access database data types to ODBC to Objective-C and Java:

Microsoft Access Database Type	Generic ODBC Type	Objective-C Type	Java Data Type
TEXT	SQL_VARCHAR	NSString	String
CURRENCY	SQL_NUMERIC	NSDecimalNumber	BigDecimal
BINARY	SQL_BINARY	NSData	NSData
DATETIME	SQL_TIMESTAMP	NSCalendarDate	NSGregorianDate

The following table lists the mapping between generic ODBC types and Objective-C types.

ODBC Data Type	Objective-C Data Type	Java Data Type
SQL_VARCHAR	NSString	String
SQL_CHAR	NSString	String
SQL_LONGVARCHAR	NSString	String
SQL_DECIMAL	NSDecimalNumber	BigDecimal
SQL_NUMERIC	NSDecimalNumber	BigDecimal
SQL_BIGINT	NSNumber	Number
SQL_SMALLINT	NSNumber	Number
SQL_INTEGER	NSNumber	Number
SQL_REAL	NSNumber	Number
SQL_FLOAT	NSNumber	Number
SQL_DOUBLE	NSNumber	Number
SQL_BIT	NSNumber	Number
SQL_TINYINT	NSNumber	Number
SQL_VARBINARY	NSData	NSData
SQL_BINARY	NSData	NSData
SQL_LONGVARBINARY	NSData	NSData
SQL_TIMESTAMP	NSCalendarDate	NSGregorianDate
SQL_DATE	NSCalendarDate	NSGregorianDate
SQL_TIME	NSCalendarDate	NSGregorianDate

Since ODBCAdaptor's type information is stored in a model's connection dictionary, the type mapping methods—**externalTypesWithModel:**, **internalTypeForExternalType:model:**, and **isValidQualifierType:model:**—use the model argument if it is provided. If the model argument isn't provided, these methods don't have data type information available to them.

Prototype Attributes

The ODBCEOAdaptor Framework provides the following set of prototype attributes:

Name	External Type	Value Class Name	Other Attributes
binaryID	BINARY	NSData	width = 12

city	CHAR	NSString	columnName = CITY
			width = 50

date	DATETIME	NSCalendarDate	columnName = ""
longText	LONGTEXT	NSString	
money	CURRENCY	NSDecimalNumber	columnName = ""

phoneNumber	CHAR	NSString	columnName = PHONE
			width = 20

rawImage	LONGBINARY	NSData	columnName = RAW_IMAGE

state	CHAR	NSString	columnName = STATE;
			width = 2

streetAddress	CHAR	NSString	columnName = STREET_ADDRESS
			width = 100

tiffImage	LONGBINARY	NSImage	adaptorValueConversionMethodName = TIFFRepresentation
			columnName = PHOTO
			valueFactoryMethodName = "imageWithData:";

uniqueID	LONG	NSNumber	columnName = ""
			valueType = i

zipCode	CHAR	NSString	columnName = ZIP
			width = 10

Generating Primary Keys

Each adaptor provides a database-specific implementation of the method **primaryKeyForNewRowWithEntity:** for generating primary keys. The ODBCChannel's implementation uses a table named EO_PK_TABLE to keep track of the next available primary key value for a given table. The table contains a row for each table for which the adaptor provides primary key values.

ODBCChannel's implementation of **primaryKeyForNewRowWithEntity:** attempts to select a value from the EO_PK_TABLE for the new row's table. If the attempt fails because the table doesn't exist, the adaptor creates the table using the following SQL statement:

```
CREATE TABLE EO_PK_TABLE (
    NAME TEXT_TYPE(40),
    PK NUMBER_TYPE
)
```

where *TEXT_TYPE* is the external (database) type for characters and *NUMBER_TYPE* is the external type for the table's primary key attribute. The ODBC adaptor sets the PK value for each row to the corresponding table's maximum primary key value plus one. After determining a primary key value for the new row, the ODBC adaptor updates the counter in the corresponding row in EO_PK_TABLE.

For more information on this topic, see *Enterprise Objects Framework Developer's Guide*.

Bind Variables

The ODBCAdaptor uses bind variables. A bind variable is a placeholder used in an SQL statement that is replaced with an actual value after the database server determines an execution plan. You use the following ODBCSQLExpression methods to operate on bind variables:

– bindVariableDictionaryForAttribute:value:
– mustUseBindVariableForAttribute:
– shouldUseBindVariableForAttribute:

ODBCAdaptor

Inherits From:	EOAdaptor : NSObject
Declared In:	ODBCEOAdaptor/ODBCAdaptor.h

Class Description

An ODBCAdaptor represents a single connection to an ODBC database server, and is responsible for keeping login and model information, performing ODBC-specific formatting of SQL expressions, and reporting errors.

ODBC (Open Data Base Connectivity) defines a standard interface that Windows applications can use to access any data source. Unlike the other Enterprise Objects Frameworks adaptors that support a single type of database, the ODBC adaptor supports any data source that has an ODBC driver. Consequently, in addition to having standard adaptor features, the ODBC adaptor also manages information relating to the driver and to the data types defined by the data source the driver supports.

The ODBCAdaptor class doesn’t support nested transactions.

Method Types

Mapping external types to internal types	+ assignExternalTypeForAttribute:
	+ externalTypeForOdbcType:model:
	+ externalTypesWithModel:
	+ getOdbcInfoWithConnectionDictionary:
	+ internalTypeForExternalType:model:
	+ odbcTypeForExternalType:model:
	+ odbcTypeForStringRepresentation:
	+ resetOdbcInfoWithConnectionDictionary:
	+ stringRepresentationForOdbcType:
Access information in the connection dictionary	+ driverInfoForModel:
	+ typeInfoForModel:
	– driverInfo
	– typeInfo

Testing the connection dictionary	– assertConnectionDictionaryIsValid – defaultExpressionClass – isValidQualifierType:model: – odbcConnectionString – odbcEnvironment

Class Methods

assignExternalTypeForAttribute:

+ (void)**assignExternalTypeForAttribute:**(EOAttribute *)*attribute*

Sets the external information for *attribute* based on the internal type, precision, and width.

driverInfoForModel:

+ (NSDictionary *)**driverInfoForModel:**(EOModel *)*model*

Returns an NSDictionary containing the driver information cached in the connection dictionary of *model*. If the information is not yet cached in *model*, connects to the database to get it.

See also: **typeInfoForModel:**, – **driverInfo**, – **typeInfo**

externalTypeForOdbcType:model:

+ (NSString *)**externalTypeForOdbcType:**(int)*type* **model:**(EOModel *)*model*

Returns the external type that represents the best match for an ODBC *type* in *model*.

externalTypesWithModel:

+ (NSArray *)**externalTypesWithModel:**(EOModel *)*model*

Overrides the EOAdaptor method **externalTypesWithModel:** to return the ODBC database types.

See also: **internalTypeForExternalType:model:**

getOdbcInfoWithConnectionDictionary:

+ (NSDictionary *)**getOdbcInfoWithConnectionDictionary:**(NSDictionary *)*connectionDictionary*

Sets up the typeInfo and driverInfo dictionaries in *connectionDictionary*, and returns an updated connection dictionary. Creates an ODBCAdaptor, ODBCContext, and ODBCChannel, and connects to the database to get the information for the typeInfo and driverInfo dictionaries.

internalTypeForExternalType:model:

+ (NSString *)**internalTypeForExternalType:**(NSString *)*externalType* **model:**(EOModel *)*model*

Overrides the EOAdaptor method **internalTypeForExternalType:model:** to return the name of the Objective-C class used to represent values stored in the database as *externalType*.

See also: **externalTypesWithModel:**

odbcTypeForExternalType:model:

+ (NSString *)**odbcTypeForExternalType:**(NSString *)*externalType* **model:**(EOModel *)*model*

Returns the ODBC type for *externalType*, as defined in the typeInfo dictionary in *model*’s connection dictionary.

odbcTypeForStringRepresentation:

+ (int)**odbcTypeForStringRepresentation:**(NSString *)*type*

Returns the ODBC type (such as SQL_CHAR) for *type* (such as @“CHAR”). The method **stringRepresentationForOdbcType:** performs the opposite function: returning a string for a specified ODBC type. These methods are used in conjunction to encode ODBC types in the typeInfo dictionary.

resetOdbcInfoWithConnectionDictionary:

+ (NSDictionary *)**resetOdbcInfoWithConnectionDictionary:**(NSDictionary *)*connectionDictionary*

Removes the typeInfo and driverInfo dictionaries from a copy of *connectionDictionary* and returns the modified connection dictionary.

stringRepresentationForOdbcType:

+ (NSString *)**stringRepresentationForOdbcType:**(int)*type*

Returns the string representation of *type*—for example, for the type SQL_CHAR this method would return the string @“CHAR”. The method **odbcTypeForStringRepresentation:** performs the opposite function: returning the ODBC type for a specified string. These methods are used in conjunction to encode ODBC types in the typeInfo dictionary.

typeInfoForModel:

+ (NSDictionary *)**typeInfoForModel:**(EOModel *)*model*

Returns an NSDictionary containing the type information cached in the connection dictionary of *model*. If the information is not yet cached in *model*, connects to the database to get it.

See also: **driverInfoForModel:**, – **driverInfo**, – **typeInfo**

Instance Methods

assertConnectionDictionaryIsValid

– (void)**assertConnectionDictionaryIsValid**

Determines whether the receiver’s connection dictionary is valid. The adaptor uses this method in conjunction with displaying a server login panel. Raises an exception if an error occurs.

Note that this method doesn’t open a connection to the database—that happens when the first adaptor channel is sent an **openChannel** message.

defaultExpressionClass

– (Class)**defaultExpressionClass**

Returns the ODBCSQLExpression class.

driverInfo

– (NSDictionary *)**driverInfo**

Returns an NSDictionary containing the driver information cached in the receiver’s model’s connection dictionary. If the information is not yet cached in the model, connects to the database to get it.

See also: – **typeInfo**

isValidQualifierType:model:

– (BOOL)**isValidQualifierType:**(NSString *)*typeName* **model:**(EOModel *)*model*

Returns YES if *model*’s attribute of the type *typeName* can be used in a qualifier, otherwise returns NO.

odbcConnectionString

– (NSString *)**odbcConnectionString**

Returns the user name, password, and data source as a string that’s used to connect to the database.

odbcEnvironment

– (void *)**odbcEnvironment**

Returns the ODBC Environment Handle HENV as a **void***; to work with it, you must cast it to HENV.

typeInfo

– (NSDictionary *)**typeInfo**

Returns an NSDictionary containing the type information cached in the receiver’s model’s connection dictionary. If the information is not yet cached in the model, connects to the database to get it.

See also: – **driverInfo**, **driverInfoForModel:**, **typeInfoForModel:**

ODBCChannel

Inherits From:	EOAdaptorChannel : NSObject
Declared In:	ODBCEOAdaptor/ODBCChannel.h

Class Description

An ODBCChannel represents an independent communication channel to the database server its ODBCAdaptor is connected to. All of an ODBCChannel's operations take place within the context of transactions controlled or tracked by its ODBCContext. An ODBCContext can manage multiple ODBCChannels, and a channel is associated with only one context.

The features ODBCChannel adds to EOAdaptorChannel are methods for returning the ODBC Statement Handle (HSTMT), and for returning a dictionary-formatted result from SQLTypeInfo().

Method Types

Getting the HSTMT data structure	
	– odbcStatement
Getting type information	
	– odbcTypeInfo
Opening and closing a channel	
	– openChannel
	– closeChannel
	– isOpen
Modifying rows	
	– deleteRowsDescribedByQualifier:entity:
	– insertRow:forEntity:
Getting schema information	
	– describeModelWithTableNames:
	– describeTableNames

Fetching rows	
	– selectAttributes:fetchSpecification:lock:entity:
	– fetchRowWithZone:
	– attributesToFetch
	– cancelFetch
	– describeResults
	– setAttributesToFetch:
	– isFetchInProgress
Sending SQL to the server	
	– evaluateExpression:
Assigning primary keys	
	– primaryKeyForNewRowWithEntity:

Instance Methods

attributesToFetch

– (NSArray *)**attributesToFetch**

Overrides the EOAdaptorChannel method **attributesToFetch** to return the set of attributes to retrieve with **fetchRowWithZone:**.

cancelFetch

– (void)**cancelFetch**

Overrides the EOAdaptorChannel method **cancelFetch** to clear all result sets established by the last **selectAttributes:fetchSpecification:lock:entity:** or **evaluateExpression:** message and terminate the current fetch, so that **isFetchInProgress** returns NO.

closeChannel

– (void)**closeChannel**

Overrides the EOAdaptorChannel method **closeChannel** to close the channel so that it can't perform operations with the server. Any fetch in progress is canceled. This method has the side effect of closing the receiver's adaptor context's connection with the database if the receiver is its adaptor context's last open channel.

deleteRowsDescribedByQualifier:entity:

– (unsigned)**deleteRowsDescribedByQualifier:**(EOQualifier *)*qualifier*
entity:(EOEntity *)*entity*

Overrides the EOAdaptorChannel method **deleteRowsDescribedByQualifier:entity:** to delete the rows described by *qualifier* and return the number of rows deleted. Raises an exception on failure. Some possible reasons for failure are:

- The adaptor channel isn't open
- The adaptor channel is in an invalid state (for example, it's fetching).
- An error occurs in the database server

describeModelWithTableNames:

– (EOModel *)**describeModelWithTableNames:**(NSArray *)*tableNames*

Overrides the EOAdaptorChannel method **describeModelWithTableNames:** to create and return a default model containing entities for the tables specified in *tableNames*. Assigns the adaptor name and connection dictionary to the new model. This method is typically used in conjunction with **describeTableNames**. Raises an exception if an error occurs.

describeResults

– (NSArray *)**describeResults**

Overrides the EOAdaptorChannel method **describeResults** to return an array of EOAttributes describing the properties available in the current result set, as determined by **selectAttributes:fetchSpecification: lock:entity:** or a statement evaluated by **evaluateExpression:**. Raises an exception if an error occurs.

describeTableNames

– (NSArray *)**describeTableNames**

Overrides the EOAdaptorChannel method **describeTableNames** to read and return an array of table names from the database. This method is used in conjunction with **describeModelWithTableNames:** to build a default model. Raises an exception if an error occurs.

evaluateExpression:

– (void)**evaluateExpression:**(EOSQLExpression *)*expression*

Overrides the EOAdaptorChannel method **evaluateExpression:** to send *expression* to the database server for evaluation, beginning a transaction first and committing it after evaluation if a transaction isn't already in progress. Raises an exception if an error occurs.

fetchRowWithZone:

– (NSMutableDictionary *)**fetchRowWithZone:**(NSZone *)*zone*

Overrides the EOAdaptorChannel method **fetchRowWithZone:** to fetch the next row from the result set of the last **selectAttributes:fetchSpecification:lock:entity:** or **evaluateExpression:** message sent to the receiver. Returns values for the receiver's **attributesToFetch**. When there are no more rows in the current result set, this method returns **nil**, and invokes the delegate method **adaptorChannelDidChangeResultSet:** if there are more results sets. When there are no more rows or result sets, this method returns **nil**, ends the fetch, and invokes **adaptorChannelDidFinishFetching:**. **isFetchInProgress** returns YES until the fetch is canceled or until this method exhausts all result sets and returns **nil**. Raises an exception if an error occurs.

insertRow:forEntity:

– (void)**insertRow:**(NSDictionary *)*row* **forEntity:**(EOEntity *)*entity*

Overrides the EOAdaptorChannel method **insertRow:forEntity:** to insert the values of *row* into the table in the database that corresponds to *entity. row* is an NSDictionary whose keys are attribute names and whose values are the values to insert. Raises an exception on failure. Some possible reasons for failure are:

- The user logged in to the database doesn't have permission to insert a new row.
- The adaptor channel is in an invalid state (for example, fetching).
- The row fails to satisfy a constraint defined in the database server.

isFetchInProgress

– (BOOL)**isFetchInProgress**

Overrides the EOAdaptorChannel method **isFetchInProgress** to return YES if the receiver is fetching, NO otherwise. An adaptor channel is fetching if:

- It's been sent a successful **selectAttributes:fetchSpecification:lock:entity:** message.
- An expression sent through **evaluateExpression:** resulted in a select operation being performed.

An adaptor channel stops fetching when there are no more records to fetch or when it's sent a **cancelFetch** message.

isOpen

– (BOOL)**isOpen**

Overrides the EOAdaptorChannel method **isOpen** to return YES if the channel has been opened with **openChannel**, NO if not.

odbcStatement

– (void *)**odbcStatement**

Returns the ODBC Statement Handle HSTMT as a **void***; you must cast the returned value to HSTMT to work with it.

odbcTypeInfo

– (NSDictionary *)**odbcTypeInfo**

Returns the result from SQLTypeInfo(), formatted in an NSDictionary ready to incorporate into a model file.

openChannel

– (void)**openChannel**

Overrides the EOAdaptorChannel method **openChannel** to put the channel and both its context and adaptor into a state where they are ready to perform database operations. Raises an exception if error occurs.

primaryKeyForNewRowWithEntity:

– (NSDictionary *)**primaryKeyForNewRowWithEntity:**(EOEntity *)*entity*

Overrides the EOAdaptorChannel method **primaryKeyForNewRowWithEntity:** to return a primary key for a new row in the database table that corresponds to *entity*. If unsuccessful, returns **nil**.

selectAttributes:fetchSpecification:lock:entity:

– (void)**selectAttributes:**(NSArray *)*attributes*
  fetchSpecification:(EOFetchSpecification *)*fetchSpecification*
  lock:(BOOL)*flag*
  entity:(EOEntity *)*entity*

Overrides the EOAdaptorChannel method **selectAttributes:fetchSpecification:lock:entity:** to select *attributes* in rows matching the qualifier in *fetchSpecification* and set the receiver's attributes to fetch. The selected rows compose one or more result sets, each row of which will be returned by subsequent **fetchRowWithZone:** messages according to *fetchSpecification*'s sort orderings. If *flag* is YES, the rows are locked if possible so that no other user can modify them (the lock specification in *fetchSpecification* is ignored). Raises an exception if an error occurs. Some possible reasons for failure are:

- The adaptor channel is in an invalid state (for example, fetching).
- The database failed to lock the specified rows.

setAttributesToFetch:

– (void)**setAttributesToFetch:**(NSArray *)*attributes*

Overrides the EOAdaptorChannel method **setAttributesToFetch:** to change the set of *attributes* used to describe the fetch data in the middle of a select. This method raises an exception if invoked when there is no fetch in progress.

updateValues:inRowsDescribedByQualifier:entity:

– (unsigned)**updateValues:**(NSDictionary *)*row*
  inRowsDescribedByQualifier:(EOQualifier *)*qualifier*
  entity:(EOEntity *)*entity*

Overrides the EOAdaptorChannel method **updateValues:inRowsDescribedByQualifier:entity:** to update the rows described by *qualifier* with the values in *values*. *values* is an NSDictionary whose keys are attribute names and whose values are the new values for those attributes (the dictionary need only contain entries for the attributes being changed). Returns the number of updated rows. Raises an exception if an error occurs. Some possible reasons for failure are:

- The user logged in to the database doesn't have permission to update.
- The adaptor channel is in an invalid state (for example, fetching).
- The new values fail to satisfy a constraint defined in the database server.

ODBCContext

Inherits From: EOAdaptorContext : NSObject

Declared In: ODBCEOAdaptor/ODBCContext.h

Class Description

An ODBCContext represents a single transaction scope on the database server to which its adaptor object is connected. If the server supports multiple concurrent transaction sessions, the adaptor may have several adaptor contexts. An ODBCContext may in turn have several ODBCChannels, which handle actual access to the data on the server.

The features the ODBCContext class adds to EOAdaptorContext are methods for managing ODBC connections and for getting information about the driver.

Instance Methods

odbcConnect

– (void)**odbcConnect**

Opens a connection to the database server. ODBCChannel sends this message to ODBCContext when it (ODBCChannel) is about to open a channel to the server. This method is called automatically by the framework.

odbcDatabaseConnection

– (void *)**odbcDatabaseConnection**

Returns the ODBC Database Connection Handle (HDBC) as a **void***; you must cast it to HDBC to work with it.

See also: – **setOdbcDatabaseConnection:**

odbcDisconnect

– (void)**odbcDisconnect**

Closes the connection to the database server. ODBCChannel sends this message to ODBCContext when it (ODBCChannel) has just closed a channel to the server.

setOdbcDatabaseConnection:

– (void)**setOdbcDatabaseConnection:**(void *)*odbcDatabaseConnection*

Sets to *odbcDatabaseConnection* the ODBC Database Connection Handle (HDBC). You can invoke this method from the delegate method **adaptorContextShouldConnect:** to set up a connection in an alternative way (by using SQLBrowseConnect(), for example).

See also: – **odbcDatabaseConnection**

ODBCSQLExpression

Inherits From: EOSQLExpression : NSObject

Declared In: ODBCEOAdaptor/ODBCSQLExpression.h

Class Description

ODBCSQLExpression defines how to build SQL statements for ODBCChannels.

Bind Variables

The ODBCAdaptor uses bind variables. A bind variable is a placeholder used in an SQL statement that is replaced with an actual value after the database server determines an execution plan. You use the following methods to operate on bind variables:

– bindVariableDictionaryForAttribute:value:
– mustUseBindVariableForAttribute:
– shouldUseBindVariableForAttribute:

Instance Methods

bindVariableDictionaryForAttribute:value:

– (NSMutableDictionary *)**bindVariableDictionaryForAttribute:**(EOAttribute *)*attribute* **value:***value*

Overrides the EOSQLExpression method **bindVariableDictionaryForAttribute:value:** to return the receiver’s bind variable dictionaries. For more information on bind variables, see the discussion in the class description.

See also: – **mustUseBindVariableForAttribute:**, – **shouldUseBindVariableForAttribute:**

lockClause

– (NSString *)**lockClause**

Overrides the EOSQLExpression method **lockClause** to return the SQL string used in a SELECT statement to lock selected rows. If you’re using the Microsoft SQL Server, this method returns @“HOLDLOCK”. Otherwise, it returns @“FOR UPDATE”.

mustUseBindVariableForAttribute:

– (BOOL)**mustUseBindVariableForAttribute:**(EOAttribute *)*attribute*

Overrides the EOSQLExpression method **mustUseBindVariableForAttribute:** to return YES since in the ODBC adaptor, the receiver must always use bind variables for *attribute*. A returned value of YES indicates that the underlying RDBMS requires that bind variables be used for attributes with *attribute*'s external type.

See also: – **shouldUseBindVariableForAttribute:**, – **bindVariableDictionaryForAttribute:value:**

prepareSelectExpressionWithAttributes:lock:fetchSpecification:

– (void)**prepareSelectExpressionWithAttributes:**(NSArray *)*attributes* **lock:**(BOOL)*lock* **fetchSpecification:**(EOFetchSpecification *)*fetchSpec*

Overrides the EOSQLExpression method **prepareSelectExpressionWithAttributes:lock: fetchSpecification:**to generate a SELECT statement. For a more complete description of what this entails, see the **prepareSelectExpressionWithAttributes:lock:fetchSpecification:** method description in the EOSQLExpression class specification.

shouldUseBindVariableForAttribute:

– (BOOL)**shouldUseBindVariableForAttribute:**(EOAttribute *)*attribute*

Overrides the EOSQLExpression method **shouldUseBindVariableForAttribute:** to return YES since in the ODBC adaptor, the receiver must always be able to provide a bind variable dictionary for *attribute*. A returned value of YES indicates that the receiver should use bind variables for attributes with *attribute*'s external type.

See also: – **mustUseBindVariableForAttribute:**, – **bindVariableDictionaryForAttribute:value:**