The OracleEOAdaptor Framework

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

Introduction

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

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

Class	Description
OracleAdaptor	Represents a single connection to a Oracle database server, and is responsible for keeping login and model information, performing Oracle-specific formatting of SQL expressions, and reporting errors.
OracleChannel	Represents an independent communication channel to the database server its OracleAdaptor is connected to.
OracleContext	Represents a single transaction scope on the database server to which its adaptor object is connected.
OracleSQLExpression	Defines how to build SQL statements for OracleChannels.

The Connection Dictionary

The connection dictionary contains items needed to connect to an Oracle server, such as the server name and database (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. The logon keys for Oracle are as follows:

serverId
userName
password

The OracleAdaptor attempts to connect with a connection string of the form “userName/password@serverId”. If all the values except the one for serverId are absent, then OracleAdaptor attempts to connect with just the value for serverId. For more information on logon keys, see ““Using SQL*Net”.”

The connection dictionary can optionally include two additional keys: connectionString and NLS_LANG. The connectionString contains a string to be used to login to the database server. If the connectionString key is present in the connection dictionary, the other logon keys are ignored and this string is used to connect to the database.

NLS_LANG allows you to set the Oracle NLS_LANG environment variable. NLS_LANG declares to the Oracle server the character set being used by the client, as well as the language in which you want server error messages to appear. The format is as follows:

language_territory. characterSet

For example, supplying the value japanese_japan.jeuc for the NLS_LANG key tells the server that the language is Japanese, the territory is Japan, and the character set is jeuc. See your Oracle documentation for a complete list of types available for this field.

To add the NLS_LANG key and a value to your connection dictionary, you must manually edit your model file. For example:

```
connectionDictionary = {
    password = tiger;
    serverId = sjOracle;
    userName = scott;
    NLS_LANG = american_america.us7ascii;
};
```

Subsequently changing the connection dictionary in your model file using the Set Adaptor Info command in EOModeler has no effect on these keys and their values-they are preserved unless you edit the file to remove them.

The default character set for Japanese systems is jeuc. If you are using a non-Japanese system, the default is whatever Oracle provides. You only need to add the NLS_LANG key to your connection dictionary if you are using a character set other than your system's default.

Note: Enterprise Objects Framework uses Rhapsody encoding to represent string data, and it passes strings to the database without converting them to the database character set. If you require that the data passed to your server is in an encoding other than Rhapsody encoding, you need to subclass NSString.

Locking

All adaptors use the database server's native locking facilities to lock rows on the server. The Oracle adaptor locks a row by using the SELECT... FOR UPDATE... statement. This 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. The following table lists the mapping used by OracleAdaptor.

Oracle Data Type	Objective-C Data Type	Java Data Type
VARCHAR2	NSString	String
NUMBER	NSDecimalNumber	BigDecimal
LONG	NSString	String
DATE	NSCalendarDate	NSGregorianDate
RAW	NSData	NSData
LONG RAW	NSData	NSData
CHAR	NSString	String
MLSLABEL	NSString	String
REFCURSOR	OracleChannel	OracleChannel

The type mapping methods—**externalTypesWithModel:**, **internalTypeForExternalType:model:**, and **isValidQualifierType:model:**—allow for an adaptor to supplement its set of type mappings with additional mappings for user-defined database types. OracleAdaptor does not make use of the model argument if one is provided.

Prototype Attributes

The OracleAdaptor Framework provides the following set of prototype attributes:

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

city	VARCHAR2	NSString	columnName = CITY
			width = 50

date	DATE	NSCalendarDate	columnName = "
longText	LONG	NSString	
money	NUMBER	NSDecimalNumber	columnName = ""

phoneNumber	VARCHAR2	NSString	columnName = PHONE
			width = 20

rawImage	"LONG RAW"	NSData	columnName = RAW_IMAGE

state	VARCHAR2	NSString	columnName = STATE
			width = 2

streetAddress	VARCHAR2	NSString	columnName = STREET_ADDRESS
			width = 100

tiffImage	"LONG RAW"	NSImage	adaptorValueConversionMethodName = TIFFRepresentation
			columnName = PHOTO
			valueFactoryMethodName = "imageWithData:"

uniqueID	NUMBER	NSNumber	columnName = ""
			valueType = i

zipCode	VARCHAR2	NSString	columnName = ZIP
			width = 10

Handling Errors

OracleChannel provides a method for handling errors: **raiseOracleError**. This method is invoked whenever the channel encounters an error reported by the Oracle server. The methods **cursorDataArea**, **hostDataArea**, and **logonDataArea** are used to retrieve Oracle-specific data structures from the channel and context to determine what error has occurred. The **hostDataArea** and **logonDataArea** methods are declared in the OracleContext class.

Generating Primary Keys

Each adaptor provides a database-specific implementation of the method **primaryKeyForNewRowWithEntity:** for generating primary keys. The OracleChannel's implementation uses sequence objects to provide primary key values. The statement used to create the sequence is:

```
create sequence table_SEQ
```

where table is the name of the table for which the adaptor provides primary key values. The adaptor sets the sequence start value to the corresponding table's maximum primary key value plus one.

To use OracleChannel's database-specific primary key generation mechanism, be sure that your database accommodates the adaptor's scheme. To modify your database so that it supports the adaptor's mechanism for generating primary keys, use EOModeler. For more information on this topic, see *Enterprise Objects Framework Developer's Guide*.

Bind Variables

The OracleAdaptor 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 OracleSQLExpression methods to operate on bind variables:

– bindVariableDictionaryForAttribute:value:
– mustUseBindVariableForAttribute:
– shouldUseBindVariableForAttribute:

OracleAdaptor

Inherits From:	EOAdaptor : NSObject
Declared In:	OracleEOAdaptor/OracleAdaptor.h

Class Description

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

The OracleAdaptor class has these restrictions: You can't have nested transactions, and the adaptor doesn't support full outer joins.

Method Types

Mapping external types to internal types
- + externalTypesWithModel:
- + internalTypeForExternalType:model:

Working with channels and contexts
- – adaptorChannelClass
- – adaptorContextClass

Testing the connection dictionary
- – assertConnectionDictionaryIsValid

Getting information from the connection dictionary
- – connectionKeys
- – oracleConnectionString

Coercing fetched values
- – fetchedValueForDataValue:attribute:
- – fetchedValueForDateValue:attribute:
- – fetchedValueForNumberValue:attribute:
- – fetchedValueForStringValue:attribute:

Returning the default expression class
- – defaultExpressionClass

Verifying a qualifier type
- – isValidQualifierType:model:

Class Methods

externalTypesWithModel:

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

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

See also: – **internalTypeForExternalType:model:**

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:**

Instance Methods

adaptorChannelClass

– (Class)**adaptorChannelClass**

Returns the OracleChannel class.

adaptorContextClass

– (Class)**adaptorContextClass**

Returns the OracleContext class.

assertConnectionDictionaryIsValid

– (void)**assertConnectionDictionaryIsValid**

Overrides the EOAdaptor method **assertConnectionDictionaryIsValid** to verify that the receiver can connect to the database with its connection dictionary. Briefly forms a connection to the server to validate the connection dictionary and then closes the connection. 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 message.

connectionKeys

– (NSArray *)**connectionKeys**

Returns an NSArray containing the keys in the receiver's connection dictionary. You can use this method to prompt the user to supply values for the connection dictionary.

defaultExpressionClass

– (Class)**defaultExpressionClass**

Returns the OracleSQLExpression class.

fetchedValueForDataValue:attribute:

– (NSData *)**fetchedValueForDataValue:**(NSData *)*value* **attribute:**(EOAttribute *)*attribute*

Returns *value*.

fetchedValueForDateValue:attribute:

– (NSCalendarDate *)**fetchedValueForDateValue:**(NSCalendarDate *)*date* **attribute:**(EOAttribute *)*attribute*

Returns an NSCalendarDate based on *date* whose millisecond value is set to 0.

fetchedValueForNumberValue:attribute:

– (NSNumber *)**fetchedValueForNumberValue:**(NSNumber *)*numberValue* **attribute:**(EOAttribute *)*attribute*

Returns an NSNumber based on *numberValue* that has been rounded according to the precision and scale specified for *attribute*.

fetchedValueForStringValue:attribute:

– (NSString *)**fetchedValueForStringValue:**(NSString *)*value* **attribute:**(EOAttribute *)*attribute*

Provides default processing for string values. Trims trailing spaces and returns **nil** for 0 length strings.

isValidQualifierType:model:

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

Overrides the EOAdaptor method **isValidQualifierType:model:** to return YES if an attribute of type *typeName* can be used in a qualifier (a SQL WHERE clause) sent to the database server, NO otherwise. *typeName* is the name of a type as required by the database server, such as an Oracle “NUMBER”.

oracleConnectionString

– (NSString *)**oracleConnectionString**

Returns the user name, password, host machine, and server id as a string suitable to be supplied as an argument to orlon().

OracleChannel

Inherits From:	EOAdaptorChannel : NSObject
Declared In:	OracleEOAdaptor/OracleChannel.h OracleEOAdaptor/OracleDescription.h

Class Description

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

The features OracleChannel adds to EOAdaptorChannel are as follows:

- Oracle-specific error handling
- The ability to configure the fetch buffer
- The ability to read a default list of table names from the database

Method Types

Setting channel characteristics	+ oracleTableNamesSQL
	+ setOracleTableNamesSQL:
	– cursorDataArea
	– fetchBufferLength
	– setFetchBufferLength:
Returning information from the server	
	– describeModelWithTableNames:
	– describeTableNames
Error handling	– raiseOracleError

Class Methods

oracleTableNamesSQL

+ (NSString *)**oracleTableNamesSQL**

Returns the SQL statement that will be executed when building a default model.

setOracleTableNamesSQL:

+ (void)**setOracleTableNamesSQL:**(NSString *)*sql*

Sets to *sql* the SQL statement that will be used to return a list of table names from the database. By default, this list is the result of the SQL statement:

```
SELECT TABLE_NAME FROM USER_TABLES ORDER BY TABLE_NAME
```

This setting is used by all OracleChannels in an application. You can specify a different SQL statement using the **defaults write** command, for example:

```
% defaults write NSGlobalDomain OracleTableNamesSQL "SELECT TABLE_NAME FROM..."
```

Once you use **setOracleTableNamesSQL:** to specify a setting, it supersedes values set with the **defaults write** command.

Instance Methods

cursorDataArea

– (struct cda_def *)**cursorDataArea**

If the channel is connected, returns an Oracle-specific data structure (**cda_def**) describing characteristics of the channel. Otherwise, returns NULL. This method is commonly used with the method **raiseOracleError** to determine why an error occurred.

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.

See also: – **describeTableNames**

describeTableNames

– (NSArray *)**describeTableNames**

Overrides the EOAdaptorChannel method **describeTableNames** to return an array of the names of all the tables owned by the current user. Uses the SQL defined in EOOracleTableNamesSQL if it exists.

This method is used in conjunction with **describeModelWithTableNames:** to build a default model.

See also: – **describeModelWithTableNames:**

fetchBufferLength

– (unsigned)**fetchBufferLength**

Returns the size, in bytes, of the fetch buffer. The larger the buffer, the more rows can be returned for each round trip to the server.

See also: – **setFetchBufferLength:**

raiseOracleError

– (void)**raiseOracleError**

Examines Oracle structures for error flags and raises an exception if one is found. Takes an error code and converts it into an error message. This method is invoked whenever the channel encounters an error reported by the Oracle server. This uses **cursorDataArea**, **hostDataArea**, and **logonDataArea** to retrieve Oracle-specific data structures from the channel and context to determine what error has occurred. (The **hostDataArea** and **logonDataArea** methods are declared in the OracleContext class.)

setFetchBufferLength:

– (void)**setFetchBufferLength:**(unsigned)*length*

Sets to *length* the size, in bytes, of the fetch buffer. The larger the buffer, the more rows can be returned for each round trip to the server.

See also: – **fetchBufferLength**

OracleContext

Inherits From: EOAdaptorContext : NSObject

Declared In: OracleEOAdaptor/OracleContext.h

Class Description

An OracleContext 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 OracleContext may in turn have several OracleChannels, which handle actual access to the data on the server.

The features the OracleContext class adds to EOAdaptorContext are methods for returning Oracle-specific data structures that describe characteristics of the context: **hostDataArea** and **logonDataArea**. **hostDataArea** returns the **hda_def** data structure, and **logonDataArea** returns the **lda_def** data structure. If you intend to extend the OracleContext by making calls to the Oracle API, you'll need these data structures.

Method Types

Managing a connection to the server	– connect
	– disconnect
	– isConnected
Returning information about an OracleContext	
	– fetchesInProgress
	– hostDataArea
	– logonDataArea

Instance Methods

connect

– (void)**connect**

Opens a connection to the database server. OracleChannel sends this message to OracleContext when it (OracleChannel) is about to open a channel to the server.

See also: – **disconnect**

disconnect

– (void)**disconnect**

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

See also: – **connect**

fetchesInProgress

– (unsigned)**fetchesInProgress**

Returns the number of fetches the receiver has in progress.

hostDataArea

– (unsigned char *)**hostDataArea**

If the channel is connected, returns an Oracle-specific data structure describing characteristics of the context. Otherwise, returns NULL. This method is commonly used with the OracleChannel method **raiseOracleError** to determine why an error occurred.

See also: – **logonDataArea**

isConnected

– (BOOL)**isConnected**

Returns YES if the receiver has an open connection to the database, NO otherwise.

See also: – **connect**, – **disconnect**

logonDataArea

– (void *)**logonDataArea**

If the channel is connected, returns an Oracle-specific data structure describing characteristics of the context. Otherwise, returns NULL. This method is commonly used with the OracleChannel method **raiseOracleError** to determine why an error occurred.

See also: – **hostDataArea**

OracleSQLExpression

Inherits From:	EOSQLExpression : NSObject
Declared In:	OracleEOAdaptor/OracleSQLExpression.h

Class Description

OracleSQLExpression defines how to build SQL statements for OracleChannels.

Method Types

Generating SQL for attributes and values	+ formatValue:forAttribute:
Getting the server type ID	+ serverTypeIdForName:
Working with no wait locks	+ setUseNoWaitLocks: + useNoWaitLocks
Getting the lock clause	– lockClause
Managing bind variables	– mustUseBindVariableForAttribute: – shouldUseBindVariableForAttribute: – bindVariableDictionaryForAttribute:value:

Class Methods

formatValue:forAttribute:

+ (NSString *)**formatValue:**(id)*value* **forAttribute:**(EOAttribute *)*attribute*

Overrides the EOSQLExpression method **formatValue:forAttribute:** to return a formatted string representation of *value* for *attribute* that is suitable for use in a SQL statement.

serverTypeIdForName:

+ (int)**serverTypeIdForName:**(NSString *)*typeName*

Returns the Oracle type code (such as OraVARCHAR2 or OraNumber) for *typeName* (such as “VARCHAR2” or “NUMBER”).

setUseNoWaitLocks:

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

Sets according to *flag* whether the lock clause of the OracleSQLExpression is @“FOR UPDATE” (block until the row is available) or @“FOR UPDATE NOWAIT” (return an error immediately if an attempt to lock a row would block). By default OracleSQLExpression uses the clause @“FOR UPDATE”—that is, by default it does *not* use NOWAIT locks. This behavior is also controllable through the EOOracleUseNoWaitLocks user default.

See also: + **useNoWaitLocks**

useNoWaitLocks

+ (BOOL)**useNoWaitLocks**

Returns YES to indicate that the OracleSQLExpression uses NOWAIT locks, NO otherwise. The default is NO.

See also: + **setUseNoWaitLocks:**

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. Queries the user default EOOracleUseNoWaitLocks. If this default is not set or if it is set to NO, this method returns the string @“FOR UPDATE”. If the default is set to YES, this method returns @“FOR UPDATE NOWAIT”.

mustUseBindVariableForAttribute:

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

Overrides the EOSQLExpression method **mustUseBindVariableForAttribute:** to return YES if the receiver must use bind variables for *attribute*, NO otherwise. 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: – **bindVariableDictionaryForAttribute:value:**, – **shouldUseBindVariableForAttribute:**

shouldUseBindVariableForAttribute:

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

Overrides the EOSQLExpression method **shouldUseBindVariableForAttribute:** to return YES if the receiver can provide a bind variable dictionary for *attribute*, NO otherwise. A returned value of YES indicates that the receiver should use bind variables for attributes with *attribute*’s external type.

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