ODBCEOAdaptor Framework

Objective–C API Reference

Apple Computer, Inc.

© 1999 Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc., except to make a backup copy of any documentation provided on CD-ROM.

The Apple logo is a trademark of Apple Computer, Inc.
Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for Apple-labeled or Apple-licensed computers.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for typographical errors.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Macintosh, and WebObjects are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Enterprise Objects is a trademark of Apple Computer, Inc.

NeXT, the NeXT logo, OPENSTEP, Enterprise Objects Framework, Objective–C, and WEBSCRIPT are trademarks of NeXT Software, Inc.

Adobe, Acrobat, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

Helvetica and Palatino are registered trademarks of Linotype-Hell AG and/or its subsidiaries.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

ORACLE is a registered trademark of Oracle Corporation, Inc.

SYBASE is a registered trademark of Sybase, Inc.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Windows NT is a trademark of Microsoft Corporation.

All other trademarks mentioned belong to their respective owners.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

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.

The ODBC adaptor defines string constants for use as connection dictionary keys:

- `dataSourceKey`
- `userNameKey`
- `passwordKey`
- `connectionStringKey`
- `typeInfoKey`
- `driverInfoKey`

The value for the `connectionStringKey` contains the user name, password, and data source. If an entry for `connectionStringKey` is present in the connection dictionary, the other login keys (`dataSourceKey`, `userNameKey`, and `passwordKey`) are ignored and the value for `connectionStringKey` is used to connect to the database.

The value for the `typeInfoKey` is a dictionary that is used to cache type information. 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 dictionary for the `typeInfoKey` 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>
```

The value for the `driverInfoKey` is a dictionary that stores information about the driver, such as its name and version.

For more information on the connection dictionary key constants, see the ODBCAdaptor class specification.

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:

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

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

ODBC Data Type	Objective-C Data Type
SQL_VARCHAR	NSString
SQL_CHAR	NSString
SQL_LONGVARCHAR	NSString
SQL_DECIMAL	NSDecimalNumber
SQL_NUMERIC	NSDecimalNumber
SQL_BIGINT	NSNumber
SQL_SMALLINT	NSNumber
SQL_INTEGER	NSNumber
SQL_REAL	NSNumber
SQL_FLOAT	NSNumber
SQL_DOUBLE	NSNumber

ODBC Data Type	Objective-C Data Type
SQL_BIT	NSNumber
SQL_TINYINT	NSNumber
SQL_VARBINARY	NSData
SQL_BINARY	NSData
SQL_LONGVARBINARY	NSData
SQL_TIMESTAMP	NSCalendarDate
SQL_DATE	NSCalendarDate
SQL_TIME	NSCalendarDate

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

Name	External Type	Value Class Name	Other Attributes
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.

Constants

ODBCAdaptor defines the following string constants for use as connection dictionary keys.

Constant	Corresponding value in the connection dictionary
`dataSourceKey`	The name of the data source to connect to.
`userNameKey`	The name of the user to log in as.
`passwordKey`	The user's password.
`connectionStringKey`	The complete string used to connect to the database. If this key is present in the dictionary, `dataSourceKey`, `userNameKey`, and `passwordKey` are ignored.
`typeInfoKey`	Information about the types supported by the driver.
`driverInfoKey`	Information about driver, including the driver name, version, and so on.

For more information on the connection dictionary, see "The Connection Dictionary" (page 4) in the ODBCEOAdaptor Framework introduction.

Additionally, ODBCAdaptor defines a string constant for use as a key in an exception's userInfo dictionary.

Constant	Corresponding value in an exception's userInfo dictionary
`SQLStatesKey`	An array of strings. Each string is a five character code corresponding to an ODBC SQL state.

Method Types

Mapping external types to internal types

+ assignExternalTypeForAttribute:
+ externalTypeForOdbcType:model:
+ getOdbcInfoWithConnectionDictionary:
+ odbcTypeForExternalType:model:
+ odbcTypeForStringRepresentation:
+ resetOdbcInfoWithConnectionDictionary:
+ stringRepresentationForOdbcType:

Access information in the connection dictionary

- odbcConnectionString
+ driverInfoForModel:
+ typeInfoForModel:
- driverInfo
- typeInfo

Getting the default Expression Class

- defaultExpressionClass

Getting ODBC environment information

- 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*.

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.

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

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

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()`.

Instance Methods

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.

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.

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.

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.

odbcDriverInfo

```
- (NSDictionary *)odbcDriverInfo
```

Returns a dictionary summarizing some important information about the driver (driver name, version, support of NOT NULL, and so on). Connects to the database if a connection isn't already in place.

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).

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:`

For more information on using bind variables, see the EOSQLExpression class specification.

Class Methods

bindVariableDictionaryForAttribute:value:

```
- (NSMutableDictionary *)bindVariableDictionaryForAttribute:(EOAttribute *)attribute
   value:value
```

Overrides the EOSQLExpression implementation to return the receiver's bind variable dictionaries. For more information on bind variables, see the discussion in the class description.

lockClause

```
- (NSString *)lockClause
```

Overrides the EOSQLExpression implementation 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 implementation 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.

shouldUseBindVariableForAttribute:

```
- (BOOL)shouldUseBindVariableForAttribute:(EOAttribute *)attribute
```

Overrides the EOSQLExpression implementation 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.

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software.

Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.