The SybaseEOAdaptor Framework

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

Introduction

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

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

Class	Description
SybaseAdaptor	Represents a single connection to a Sybase database server, and is responsible for keeping login and model information, performing Sybase-specific formatting of SQL expressions, and reporting errors.
SybaseChannel	Represents an independent communication channel to the database server its SybaseAdaptor is connected to.
SybaseContext	Represents a single transaction scope on the database server to which its adaptor object is connected.
SybaseSQLExpression	Defines how to build SQL statements for SybaseChannels.

The Connection Dictionary

The connection dictionary contains items needed to connect to a Sybase 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. For Sybase databases the required keys are as follows (all are defined constants):

hostName
databaseName
userName
password

The connection dictionary can optionally include three other keys (which are also defined constants): **sybasePasswordEncryption**, **LC_ALL**, and **primitiveTypeMap**. **sybasePasswordEncryption** provides support for Sybase password encryption. **LC_ALL** declares to the Sybase server the character set being used by the client (such as eucjis, ascii7, or iso_1). For a complete list of types available for this field, see your Sybase documentation. **primitiveTypeMap** describes the mapping of user-defined data types to their base Sybase type (such as varchar or datetime). For more information on user-defined data types, see "Data Type Mapping".

To add any of these optional keys and appropriate values to your connection dictionary, you can manually edit your model file. For example:

```
connectionDictionary = {databaseName = People;
    hostName = "";
    LC_ALL = eucjis;
    password = "";
    primitiveTypeMap = {id = varchar; ssn = char(9); };
    sybasePasswordEncryption = YES;
    userName = "";
};
```

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. Alternatively you can add the optional keys to a model's connection dictionary programmatically.

The default character set for non-Japanese systems is iso_1 (that is, ISO Latin 1), while the default character set for Japanese systems is eucjis. You only need to add the LC_ALL key to your connection dictionary if you are using a character set other than your system's default.

Error Handling

SybaseAdaptor, SybaseContext, and SybaseChannel can raise exceptions due to programming errors that result in invalid argument values or internal inconsistencies. In addition, messages, errors, and failure status returned from the Sybase SQL Server and client libraries can also result in EOGeneralAdaptorExceptions. When an exception results from a callback to the CS_CLIENTMSG_CB (Sybase ClientMessage callback) or the CS_SERVERMSG_CB (Sybase ServerMessage callback), all of the information passed into this routine is available in the userInfo dictionary contained by the exception. When an exception is raised in response to a Sybase ClientMessage callback, you can get the information provided by the client library as follows:

```
clientMsgDict = [[localException userInfo]
    objectForKey:@"sybaseClientMessageDictionary"];
```

The **clientMsgDict** contains the following keys which have values corresponding to those sent in the callback function that raised the exception: msgstring, osstring, sqlstate, severity, msgnumber, osnumber, status.

Similarly, when the exception is raised in response to a Sybase ServerMessage callback, you can get the information provided by the server as follows:

```
svrMsgDict = [[localException userInfo]
    objectForKey:@"sybaseServerMessageDictionary"];
```

The **svrMsgDict** contains the following keys which have values corresponding to those sent in the callback function that raised the exception: text, svrname, proc, sqlstate, msgnumber, state, severity, line, status.

Locking

All adaptors use the database server’s native locking facilities to lock rows on the server. The Sybase adaptor locks a row by using the HOLDLOCK keyword in SELECT statements. 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.

The semantics of the HOLDLOCK keyword are such that when you lock a row other users can’t update it, but it doesn’t guarantee that your update will succeed. This is because other users could be holding a lock on the same row. However, you can still read rows that are locked by other users.

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 SybaseAdaptor.

Sybase Data Type	Objective-C Data Type	Java Data Type
binary	NSData	NSData
bit	NSNumber	Number
char	NSString	String
datetime	NSCalendarDate	NSGregorianDate
datetimn	NSCalendarDate	NSGregorianDate
decimal	NSDecimalNumber	BigDecimal

Sybase Data Type	Objective-C Data Type	Java Data Type
decimaln	NSDecimalNumber	BigDecimal
float	NSNumber	Number
floatn	NSNumber	Number
image	NSData	NSData
int	NSNumber	Number
intn	NSNumber	Number
money	NSDecimalNumber	BigDecimal
moneyn	NSDecimalNumber	BigDecimal
nchar	NSString	String
numeric	NSDecimalNumber	BigDecimal
numericn	NSDecimalNumber	BigDecimal
nvarchar	NSString	String
real	NSNumber	Number
smalldatetime	NSCalendarDate	NSGregorianDate
smallint	NSNumber	Number
smallmoney	NSDecimalNumber	BigDecimal
sysname	NSString	String
text	NSString	String
timestamp	NSData	NSData
tinyint	NSNumber	Number
varbinary	NSString	String
varchar	NSString	String

In addition, SybaseAdaptor provides a mapping for user-defined data types. For example, a custom data type **partnumber** defined as char(10) is mapped to NSString—the Objective-C type to which

partnumber’s base data type (char) is mapped. SybaseAdaptor’s implementation of **describeModelWithTableNames:** automatically creates mappings for user-defined data types and saves them in the connection dictionary of the newly created model. Consequently, even models created with EOModeler automatically include information about custom data types.

Since information about custom types 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 user-defined data type information available to them.

Prototype Attributes

The SybaseEOAdaptor Framework provides the following set of prototype attributes:

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

city	varchar	NSString	columnName = CITY
			width = 50

date	datetime	NSCalendarDate	columnName = ""
longText	text	NSString	
money	money	NSDecimalNumber	columnName = "";

phoneNumber	varchar	NSString	columnName = PHONE
			width = 20

rawImage	image	NSData	columnName = RAW_IMAGE

state	varchar	NSString	columnName = STATE
			width = 2

streetAddress	varchar	NSString	columnName = STREET_ADDRESS
			width = 100

tiffImage	image	NSImage	adaptorValueConversionMethodName = TIFFRepresentation
			columnName = PHOTO
			valueFactoryMethodName = "imageWithData:"

uniqueID	int	NSNumber	columnName = "";
			valueType = i

zipCode	varchar	NSString	columnName = ZIP
			width = 10

SQL and User-Defined Transactions

Certain data definition commands, such as CREATE TABLE, can't be executed in a user-defined transaction. However, the database channel and adaptor channel require you to start a transaction before evaluating any SQL. To work around this problem, you need to send the adaptor context or database context a **transactionDidBegin** message to make it think a transaction is in progress. Then you can send it the SQL statement, followed by a **transactionDidCommit** message.

Processing Compute Rows and Stored Procedures

SybaseChannel's delegate methods used for processing compute rows and stored procedures give you access to the three types of non-regular rows supported by Sybase: compute rows, return parameters (from a stored procedure), and status from a stored procedure. Because the access layer can only handle regular table rows, the Sybase adaptor channel normally skips non-regular rows. However, you can use the delegate methods to intercept non-regular rows before they are skipped. These delegate methods are **sybaseChannel:willFetchAttributes:forRowOfType:withComputeRowId:** and **sybaseChannel: willReturnRow:ofType:withComputeRowId:**. The method **sybaseChannel:willFetchAttributes: forRowOfType:withComputeRowId:** is invoked when a row is fetched, while **sybaseChannel: willReturnRow:ofType:withComputeRowId:** is invoked when a row is about to be returned. Based on the type of the row, the delegate can specify the appropriate behavior. This enables you to use data in one of the three non-regular row types and either extract the data from them or use the method **describeResults** to return an array of attributes that describe the properties available in the current result set. Using **describeResults** is appropriate if you're not concerned with format—for example, if you're just writing raw data to a report.

Note: The regular rows in the results from a stored procedure must map to the attributes in the corresponding entity, and must be in alphabetical order.

Generating Primary Keys

Each adaptor provides a database-specific implementation of the method **primaryKeyForNewRowWithEntity:** for generating primary keys. The SybaseChannel's implementation uses a table named eo_sequence_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. The statement used to create the eo_sequence_table is:

```
create table eo_sequence_table (
    table_name varchar(32),
    counter int null
)
```

SybaseChannel uses a stored procedure named eo_pk_for_table to access and maintain the primary key counter in eo_sequence_table. The stored procedure is defined as follows:

```
create procedure
eo_pk_for_table @tname varchar(32) as
```

```
begin
    define @max int

    update eo_sequence_table
    set counter = counter + 1
    where table_name = @tname

    select counter
    from eo_sequence_table
    where table_name = @tname
end
```

The stored procedure increments the counter in the eo_sequence_table row for the specified table, selects the counter value, and returns it. SybaseChannel executes this eo_pk_for_table stored procedure from **primaryKeyForNewRowWithEntity:** and returns the stored procedure's return value.

To use SybaseChannel'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*.

SybaseAdaptor

Inherits From:	EOAdaptor : NSObject
Declared In:	SybaseEOAdaptor/SybaseAdaptor.h

Class Description

A SybaseAdaptor represents a single connection to a Sybase database server, and is responsible for keeping login and model information, performing Sybase-specific formatting of SQL expressions, and reporting errors.

The features SybaseAdaptor adds to EOAdaptor are as follows:

- The ability to specify a client character set and language
- Sybase password encryption

The SybaseAdaptor class has these restrictions: A context can only manage one channel at a time, and the adaptor doesn't support full outer joins because the Sybase server itself doesn't support them.

Method Types

Mapping external types to internal types	+ externalTypesWithModel:
	+ internalTypeForExternalType:model:
	+ primitiveTypeForExternalType:model:
Getting information from the connection dictionary	– connectionKeys
Bracketing calls to ct_connect()	– prepareEnvironmentForConnect
	– resetEnvironmentAfterConnect
Callback methods	– sybaseContextDidDisconnect:
	– sybaseContextWillConnect:

Class Methods

externalTypesWithModel:

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

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

See also: + **internalTypeForExternalType:model:**

internalTypeForExternalType:model:

+ (NSString *)**internalTypeForExternalType:**(NSString *)*extType* **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 *extType* for the model *model*.

See also: + **externalTypesWithModel:**

primitiveTypeForExternalType:model:

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

Returns the primitive type on which a given custom type, defined on the server, is based.

Instance Methods

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.

prepareEnvironmentForConnect

– (void)**prepareEnvironmentForConnect**

A call to this method should preceed all calls to **ct_connect**() to set the **LC_ALL** environment variable setting to the value specified in the model connection dictionary.

See also: – **resetEnvironmentAfterConnect**

resetEnvironmentAfterConnect

– (void)**resetEnvironmentAfterConnect**

A call to this method should follow all calls to **ct_connect**() to set the **LC_ALL** environment variable setting to the value specified in the model connection dictionary.

See also: – **prepareEnvironmentForConnect**

sybaseContextDidDisconnect:

– (void)**sybaseContextDidDisconnect:**(SybaseContext *)*aSybaseContext*

Callback method that is invoked after the associated Sybase context disconnects.

sybaseContextWillConnect:

– (void)**sybaseContextWillDisconnect:**(SybaseContext *)*aSybaseContext*

Callback method that is invoked just before the associated Sybase context disconnects.

SybaseChannel

Inherits From:	EOAdaptorChannel : NSObject
Declared In:	SybaseEOAdaptor/SybaseChannel.h

Class Description

A SybaseChannel represents an independent communication channel to the database server its SybaseAdaptor is connected to. All of a SybaseChannel's operations take place within the context of transactions controlled or tracked by its SybaseContext. A Sybase adaptor context manages one channel, and a channel is associated with only one context.

The feature SybaseChannel adds to EOAdaptorChannel is processing for compute rows and stored procedures (see the framework introduction for more information).

SybaseChannel has two delegate methods; for a complete description, see the SybaseChannelDelegate protocol specification.

SybaseContext

Inherits From:	EOAdaptorContext : NSObject
Declared In:	SybaseEOAdaptor/SybaseContext.h

Class Description

A SybaseContext represents a single transaction scope on the database server to which its adaptor object is connected. Since a Sybase server supports multiple concurrent transaction sessions, the adaptor may have several adaptor contexts. A SybaseContext may in turn have a SybaseChannel, which handles actual access to the data on the server.

The features the SybaseContext class adds to EOAdaptorContext are methods for returning Sybase-specific data structures that describe characteristics of the context. The method **contextPointer** returns the Sybase global context pointer, so that you can make direct calls to the Sybase client library. The method **connection** returns the SybaseContext's CT library connection (CS_CONNECTION *).

The SybaseContext can have a delegate, which gives you access to all messages returned from the Sybase client library or from the Sybase Server. See the SybaseContextDelegate protocol specification for a complete description. SybaseContext also provides the following callback methods for use by the SybaseChannel:

- sybaseChannelDidClose
- sybaseChannelDidEndFetching
- sybaseChannelWillBeginFetching
- sybaseChannelWillOpen

Method Types

Getting the context pointer	+ contextPointer
Setting the login time out interval	+ loginTimeOutInterval + setLoginTimeOutInterval:
Setting the time out interval	+ setTimeOutInterval: + timeOutInterval

Managing the connection	– connect – connection – currentChannel – disconnect – isConnected
Setting the max text size default	– maxTextSizeDefault – setMaxTextSizeDefault:
Setting the current exception	– raiseCurrentException – setCurrentException:

Class Methods

contextPointer

+ (void *)**contextPointer**

Returns the Sybase global context pointer (CS_CONTEXT *). You can use this to make direct calls to the Sybase client library.

loginTimeOutInterval

+ (int)**loginTimeOutInterval**

Returns the login time out interval used by SybaseContext.

See also: + **setLoginTimeOutInterval:**

setLoginTimeOutInterval:

+ (void)**setLoginTimeOutInterval:**(int)*seconds*

Sets the login time out interval value SybaseContext uses during the creation of new channels. The default is 0, which means that there is no time out.

See also: + **loginTimeOutInterval**

setTimeOutInterval:

+ (void)**setTimeOutInterval:**(int)*seconds*

Sets the time out interval valueSybaseContext uses during the creation of new channels. The default is 0, which means that there is no time out.

See also: + **timeOutInterval**

timeOutInterval

+ (int)**timeOutInterval**

Returns the time out interval used by SybaseContext.

See also: + **setTimeOutInterval:**

Instance Methods

connect

– (void)**connect**

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

See also: – **disconnect**

connection

– (void *)**connection**

Returns the CT library connection (CS_CONNECTION *) for the receiver.

currentChannel

– (SybaseChannel *)**currentChannel**

Returns the SybaseChannel currently associated with the receiving context.

disconnect

– (void)**disconnect**

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

See also: – **connect**

isConnected

– (BOOL)**isConnected**

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

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

maxTextSizeDefault

– (int)**maxTextSizeDefault**

Returns the maximum number of bytes to be returned from a Sybase image to text field. The default is set to INT_MAX, as defined for the host machine. This number can be overwritten on a per-channel basis by sending the appropriate SQL to the channel using the **evaluateExpression:** method.

See also: – **setMaxTextSizeDefault:**

raiseCurrentException

– (void)**raiseCurrentException**

If the receiver has an exception, raises it.

See also: – **setCurrentException:**

setCurrentException:

– (void)**setCurrentException:**(NSException *)*exception*

Sets to *exception* the receiver's current exception.

When the SybaseAdaptor encounters an error, it uses the error message to build an NSException and stores the exception in the SybaseContext using this method. The exception can then be reviewed by other components to determine if the error is fatal.

See also: – **raiseCurrentException**

setMaxTextSizeDefault:

– (void)**setMaxTextSizeDefault:**(int)*textSize*

Sets to *textSize* the receiver’s default textsize. Any channels created after this method has been invoked will use the newly specified *textSize*.

See also: – **maxTextSizeDefault**

SybaseSQLExpression

Inherits From:	EOSQLExpression : NSObject
Declared In:	SybaseEOAdaptor/SybaseSQLExpression.h

Class Description

SybaseSQLExpression defines how to build SQL statements for SybaseChannels.

Method Types

Generating SQL for attributes and values	+ formatValue:forAttribute:
Getting the server type ID	+ serverTypeIdForName:
Getting the lock clause	– lockClause

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 Sybase type code (such as 47, 56, or 55) for *typeName* (such as “char”, “int”, or “decimal”).

Instance Methods

lockClause

– (NSString *)**lockClause**

Overrides the EOSQLExpression method **lockClause** to return the SQL string used in a SELECT statement to lock selected rows, which is @“HOLDLOCK”.

SybaseChannelDelegate

Adopted By:	SybaseChannel delegate objects
Declared In:	SybaseEOAdaptor/SybaseChannel.h

Protocol Description

SybaseChannel's delegate methods used for processing compute rows and stored procedures give you access to the three types of non-regular rows supported by Sybase: compute rows, return parameters (from a stored procedure), and status from a stored procedure. Because the access layer can only handle regular table rows, the Sybase adaptor channel normally skips non-regular rows. However, you can use the delegate methods to intercept non-regular rows before they are skipped. These delegate methods are **sybaseChannel:willFetchAttributes:forRowOfType:withComputeRowId:** and **sybaseChannel: willReturnRow:ofType:withComputeRowId:**. The method **sybaseChannel:willFetchAttributes: forRowOfType:withComputeRowId:** is invoked when a row is fetched, while **sybaseChannel: willReturnRow:ofType:withComputeRowId:** is invoked when a row is about to be returned. Based on the type of the row, the delegate can specify the appropriate behavior. This enables you to use data in one of the three non-regular row types and either extract the data from them or use the method **describeResults** to return an array of attributes that describe the properties available in the current result set. Using **describeResults** is appropriate if you're not concerned with format—for example, if you're just writing raw data to a report.

Note: The regular rows in the results from a stored procedure must map to the attributes in the corresponding entity, and must be in alphabetical order.

The SybaseChannel adaptor defines the following constants against which you can compare the returned row type:

- SybaseRegularRow
- SybaseComputeRow
- SybaseReturnParameterRow
- SybaseReturnStatusRow

Instance Methods

sybaseChannel:willFetchAttributes:forRowOfType:withComputeRowId:

– (NSArray *)**sybaseChannel:**(SybaseChannel *)*channel*
willFetchAttributes:(NSArray *)*attributes*
forRowOfType:(SybaseRowType)*rowType*
withComputeRowId:(int)*computeRowId*

Invoked whenever a row is fetched. The delegate can return **nil**, which causes the row to be skipped, or can return a substitute set of attributes that is appropriate for the type of row being fetched. Delegates can have the channel fabricate a set of attributes for the current non-regular row by calling **describeResults**. See the protocol introduction for a list of defined constants for *rowType*.

For example, the following implementation checks the row type; if it's a regular row, it simply returns the attributes. If it's not a regular row type, **describeResults** is used to return an array of attributes that describe the properties available in the current result set. Note that **describeResults** always describes the current row type.

```
- (NSArray *)sybaseChannel:(SybaseChannel *)channel
    willFetchAttributes:(NSArray *)attributes
    forRowOfType:(SybaseRowType)rowType
    withComputeRowId:(int)computeRowId
{
    if (rowType == SybaseRegularRow)
        return attributes;

    attributes = [(EOAdaptorChannel *) channel describeResults];
        return attributes;
}
```

sybaseChannel:willReturnRow:ofType:withComputeRowId:

- (BOOL)**sybaseChannel:**(SybaseChannel *)*channel*
willReturnRow:(NSDictionary *)*row*
ofType:(SybaseRowType)*rowType*
withComputeRowId:(int)*computeRowId*

Invoked once a row has been read from the database and packaged into the dictionary. Delegates can return YES to cause the row to be returned from **fetchAttributes:WithZone:**, or they can return NO to cause the row to be skipped. See the protocol introduction for a list of defined constants for *rowType*.

For example, the following implementation checks each row type and uses **NSLog()** to output a message describing the row's type. In this example all rows are returned, but you could use this template to selectively return or not return rows based on type.

```
- (BOOL)sybaseChannel:(SybaseChannel *)channel
```

```
    willReturnRow:(NSDictionary *)row ofType:(SybaseRowType)rowType
    withComputeRowId:(int)computeRowId
{
    switch (rowType) {
    case SybaseRegularRow:
        break;
    case SybaseComputeRow:
        NSLog(@"Returning compute row");
        break;
    case SybaseReturnParameterRow:
        NSLog(@"Returning return parameter row");
        break;
    case SybaseReturnStatusRow:
        NSLog(@"Returning return status row");
        break;
    }

    return YES;
}
```

SybaseContextDelegate

Adopted By:	SybaseContext delegate objects
Declared In:	SybaseEOAdaptor/SybaseContext.h

Class Description

The SybaseContext delegate object allows developers access to all the messages returned from the Sybase client library or the Sybase Server. If your implementation of these delegate methods returns NO, the SybaseContext will not report the message (or error). If your implementation returns YES, the SybaseContext will continue as usual. Most messages are reported in exceptions, but messages with a severity of 0 are simply ignored.

Instance Methods

sybaseContext:shouldReportClientMessage:

– (BOOL)**sybaseContext:**(SybaseContext *)*context*
shouldReportClientMessage:(NSDictionary *)*clientMessage*

Invoked when an exception results from a callback to the CS_CLIENTMSG_CB (Sybase ClientMessage callback). Gives the delegate the opportunity to substitute *clientMessage* as the userInfo dictionary.

sybaseContext:shouldReportServerMessage:

– (BOOL)**sybaseContext:**(SybaseContext *)*context*
shouldReportServerMessage:(NSDictionary *)*serverMessage*

Invoked when an exception results from a callback to the CS_SERVERMSG_CB (Sybase ServerMessage callback). Gives the delegate the opportunity to substitute *serverMessage* as the userInfo dictionary.