initFromFile:

Describing the model source directory
- name
- currentAdaptorName
- defaultAdaptorName
- defaultLoginString
- currentLoginString
- loginStringForUser:

Describing the database model entityNamed:
- getEntities:

Revising the data dictionary emptyDataDictionary
- loadDefaultDataDictionary

Connecting to the database+ findDatabaseNamed:connect:
- connect
- connectUsingAdaptor:andString:
- disconnect
- disconnectUsingString:
- isConnected
- connectionName

Managing transactions beginTransaction
- rollbackTransaction
- commitTransaction
- isTransactionInProgress
- areTransactionsEnabled
- enableTransactions:

Using a delegate delegate

connect, connectUsingAdaptor:andString:

(BOOL)arePanelsEnabled

Returns YES if the adaptor upon which the DBDatabase object is built is allowed to display panels
By default, panels are enabled you can disallow by passing NO to the setPanelsEnabled: method.

setPanelsEnabled:

(BOOL)areTransactionsEnabled

Returns YES if the DBDatabase's adaptor allows transaction contexts to be established. The meth
transactions aren't allowed or if the DBDatabase isn't currently connected to the server.

enableTransactions:

(BOOL)beginTransaction

Tells the adaptor to set up a transaction context. Exactly how the transaction is implemented depe
typically, a virtual copy of subsequently fetched data is created (by the server), thus ªstabilizingº t
transaction is in progress. When you've finished reading and modifying the data, you send commi
DBDatabase, which attempts to write the data back to the server, or rollbackTransaction, which sir
away. You're allowed to set up only one transaction context at a time.

since the previous beginTransaction will be irreversibly made in the database. Returns YES if the
committed. If the server rejects the data, this method returns NO.

beginTransaction, rollbackTransaction

(BOOL)connect

Opens a connection to the server, using the default adaptor name and login string. Returns YES i
successfully established by this method. Note well that this method returns NO if the DBDatabase

defaultAdaptorName, defaultLoginString, disconnect

(const unsigned char *)connectionName

Returns the name of the adaptor's current connection to the server (as defined by the adaptor itself
isn't connected, this returns an empty string.

(BOOL)connectUsingAdaptor:(const char *)adaptorName
andString:(const unsigned char *)aString

Opens a connection to the server using the adaptor identified by adaptorName, and the login string
supply the adaptor name if the login string aString is NULL, the method uses the default login strin
invoke this method if you want to connect to an adaptor other than the one named in the model thr
DBDatabase was created. For a ªnormalº connection, use the findDatabaseNamed:connect: class
instance method.

Returns YES if the connection is made. Note well that this method returns NO if the DBDatabase

connect, disconnect, disconnectUsingString:

(const char *)currentAdaptorName

Returns the name of the adaptor through which the DBDatabase is connected to the server. This m
the DBDatabase isn't currently connected.

Typically, the current adaptor is the same as the default adaptorÐin other words, it's the adaptor th
DBDatabase's model. The one case in which the current and default adaptors may differ is if the D
connected through the connectUsingAdaptor:andString: method.

defaultAdaptorName, defaultLoginString, currentLoginString

(const unsigned char *)currentLoginString

Returns the login string that was used to form the connection to the server. If the DBDatabase isn'
this method returns NULL.

Returns the name of the adaptor that's named in the DBDatabase's model. This is the adaptor that form a connection to the server. To use some other adaptor, you must name it in an invocation of t connectUsingAdaptor:andString: method.

currentAdaptorName, defaultLoginString, currentLoginString

(const unsigned char *)defaultLoginString

Returns the login string that's given in the DBDatabase's model. This is the login string that, by d connection to the server. To use some other string, you must pass it in an invocation of the connec andString: method.

currentLoginString, defaultAdaptorName, currentAdaptorName

delegate

Returns the DBDatabase's delegate.

setDelegate:

(const char *)directory

Returns the full pathname of the model file that the DBDatabase represents.

name

(BOOL)disconnect

Closes the connection to the database. Returns YES if the connection was successfully closed.

disconnectUsingString: connect, + findDatabaseNamed:connect:

(BOOL)disconnectUsingString:(const unsigned char *)aString

Closes the connection to the database by sending it the command aString. Returns YES if the conn successfully closed.

disconnect, connect, + findDatabaseNamed:connect:

emptyDataDictionary

Frees the information that the DBDatabase found in its model. Specifically, the entity names (and adaptor name, and login string are all erased. You should only need to invoke this method if you v default data dictionary (and that should be rare). Returns self.

loadDefaultDataDictionary

(id <DBEntities>)entityNamed:(const char *)aName

Returns the entity named aName from the DBDatabase object's list of entities, or nil if it isn't foun
is gotten from the DBDatabase's model.

getEntities:

(BOOL)evaluateString:(const unsigned char *)aString

Asks the server to evaluate the string aString, which must be a valid statement in the server's query

db:willEvaluateString:usingBinder: (delegate method)

(List *)getEntities:(List *)aList

Fills aList with the DBEntities objects that represent the model's entities. The method also returns

entityNamed:

initFromFile:(const char *)aPath

Initializes and returns the DBDatabase object from the database model information in the bundle id
aPath. Model information (database name, login string, adaptor name, and entities) are read from t
should need to invoke this method. In general, you should use the class method findDatabaseNam
DBDatabase object.

(BOOL)isConnected

Returns YES if the DBDatabase is connected to the server.

connect

(BOOL)isTransactionInProgress

Returns YES if a transaction has been started (by beginTransaction) and has not yet been committe

beginTransaction, commitTransaction, rollbackTransaction

(const unsigned char *)loginStringForUser:(const char *)aUser

Returns the login string for the database server user identified by aUser.

(const char *)name

Returns the name of the model that the DBDatabase object represents.

directory

(BOOL)rollbackTransaction

Causes the server to roll back all changes since a preceding beginTransaction. Returns YES if the
successful. Returns NO if the server couldn't roll back the transaction, or if there wasn't a transac

beginTransaction, commitTransaction, dbWillRollbackTransaction: (delegate method)

setDelegate:anObject

Makes anObject the DBDatabase's delegate. Returns self.

delegate

setPanelsEnabled:(BOOL)flag

Tells the DBDatabase to suppress (or not) the attention panels that it displays (in response to serve
By default, panels are enabled. You should disable a DBDatabase's panels if you're creating an a
on its own, or that doesn't have a graphic interface. Returns self.

arePanelsEnabled, db:notificationFrom:message:code: (delegate method)

db:aDatabase log:(const char *)fmt, ...

Invoked when the DBDatabase experiences a particularly important, stressful, or otherwise notable
argument is a log entry that can be written to a file, displayed in the user interface, spat to standard
shelf along with the object's other trophies and mementoes. The format of the log entry argument
the example implementation shown below demonstrates how to turn the argument into text (which
the user interface):

(BOOL)db:aDatabase
notificationFrom:anAdaptor
message:(const unsigned char *)msg
code:(int)errorCode

Invoked (by the adaptor) when the server encounters an exceptional situation. The arguments are:

·aDatabase is the DBDatabase object.
·anAdaptor is the object that represents the adaptor.
·msg is a string that describes the error.
·errorCode is an integer constant, defined by the server, that represents the error.

The return value is ignored.

If the delegate doesn't implement this method, and if panels are enabled, an attention panel that dis
errorCode values is presented to the user.

setPanelsEnabled:

(BOOL)db:aDb
willEvaluateString:(const unsigned char *)aString
usingBinder:aBinder

Invoked before aString, which must be expressed in the server's query language, is sent to the serv
Whether the string is actually sent depends on the value that's returned by this method: If this met
it isn't implemented), the string is sent a return of NO prevents the evaluation.

This method is invoked when the DBDatabase receives an evaluateString: message, and when the
is about to perform a data operation, such as selecting or updating.

evaluateString:

dbDidCommitTransaction:aDatabase

Invoked just after a transaction is committed. The return value is ignored.

dbWillCommitTransaction: (delegate method)

dbDidRollbackTransaction:aDatabase

Invoked just after a transaction is rolled back. The return value is ignored.

dbWillRollbackTransaction: (delegate method)

Invoked just before a transaction is rolled back. The return value is ignored.

dbDidRollbackTransaction: (delegate method)