init	
	initForDatabase:withProperties:andQualifier:
	free
Connecting to a database database	
	setDatabase:
Managing properties getProperties:	
	setProperties:
	addProperty:

Managing the record prototype+ setDynamicRecordSuperclassName:
+ setDynamicRecordClassName:
setRecordPrototype:
createRecordPrototype
ownsRecordPrototype
recordPrototype
associateRecordIvar:withProperty:
associateRecordSelectors::withProperty:
valueForProperty:

Ordering and ignoring records addRetrieveOrder:for:
removeRetrieveOrderFor:
retrieveOrderFor:
positionInOrderingsFor:
ignoresDuplicateResults
setIgnoresDuplicateResults:

Accessing the database fetch
select
selectWithoutFetching
insert
update
delete
evaluateString:
adaptorWillEvaluateString:

Fetching in a thread fetchInThread
cancelFetch
checkThreadedFetchCompletion:

Limiting a fetch setMaximumRecordsPerFetch:
maximumRecordsPerFetch
recordLimitReached

Using the shared cursor for several binders
setSharesContext:
sharesContext

Managing general resources reset
flush
scratchZone

Appointing a delegate delegate
setDelegate:

Archiving read:
write:

setRecordPrototype:

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

Returns YES if the adaptor associated with the DBBinder's DBDatabase object will accept the giv
(as determined by sending bionder:willEvaluateString: to the DBBinder's delegate), otherwise retu

binder:willEvaluateString: (DBBinder delegate)

addProperty:anObject

Adds the given object (which should conform to the DBProperties protocol) to the DBBinder's list
interested in. The list can't contain duplicates if the property is already present, the addition isn't p
value should be ignored.

Typically, you only use this method if you're building the DBBinder's property list incrementally,
DBBinder to create a record class dynamically. If you're setting your own prototype record object
setRecordPrototype:), you should, rather than use this method, inform the DBBinder of its properti
initForDatabase:withProperties:andQualifier: or setProperties:.

setProperties:, getProperties:, removePropertyAt:

addRetrieveOrder:(DBRetrieveOrder)anOrder for:(id <DBProperties>)aProperty

Establishes the order in which records are retrieved from the database (and stored in the DBBinder
the value of the aProperty property as a retrieval ªkey, º records are retrieved in least-to-greatest or
as anOrder is DB_AscendingOrder or DB_DescendingOrder. If anOrder is DB_NoOrder, the defa
removed from the retrieval order scheme. Returns self.

You can invoke this method for as many properties as you choose, but the order in which the invoc
important: The first invocation establishes the primary retrieval order property, the second establis
property, and so on. If two or more records have the same value for their primary properties, their
according to the values of their secondary properties. If they still can't be distinguished, the decisi
properties, and so on.

Note well that it's the adaptorÐnot the DBBinderÐthat retrieves records. If the adaptor that you'r
support the notion of an ordered retrieval, then this method is for naught.

retrieveOrderFor:, removeRetrieveOrderFor:, positionInOrderingsFor:

(BOOL)areObjectsFreedOnFlush

Returns YES if the objects in the DBBinder's container are freed when the DBBinder is flushed, o
Flushing is explained in the description of the flush method. By default, the objects are freed.

setFreeObjectsOnFlush:, setFlushEnabled:

associateRecordIvar:(const char *)variableName
withProperty:(id <DBProperties>)aProperty

Associates the record object instance variable named variableName with the given property such th
fetched from the database, the value of the named instance variable (in the record object that's crea

variable or one method pair invoking this method with a particular property undoes the effect of a
this or of the associateRecordSelectors:withProperty: method for that property.

associateRecordSelectors::withProperty:

associateRecordSelectors:(SEL)set
:(SEL)get
withProperty:(id <DBProperties>)aProperty

Associates the record object instance methods set and get with the given property such that when a
the database, the value at the property is set through the set method, and when the record is written
the value is retrieved through the get method. Either or both of the selector arguments may be NU
set method must take exactly one argument, the value that's being set the get method must take no
type of the value returned by the get method should match that of the set method's argument.

You should only invoke this method if you're setting your own prototype record object (through th
method). Furthermore, the prototype record must have already been set, and the object must respo
methods (if they're non-NULL). If it doesn't respond, or if aProperty isn't in the DBBinder's list
association isn't made and nil is returned. Otherwise, the method returns non-nil.

Rather than associate a property with a pair of methods, you can associate it with an instance varia
associateRecordIvar:withProperty: method. However, a single property can be associated with onl
or one method pair invoking this method with a particular property undoes the effect of a previous
the associateRecordIvar:withProperty: method for that property.

associateRecordIvar:withProperty:

cancelFetch

Interrupts an asynchronous fetch. You can also use this method after a successful synchronous fet
resources are reclaimed.

fetchInThread, fetch, fetchDone: (DBDatabase)

checkThreadedFetchCompletion:(double)timeout

If you're performing an asynchronous fetch but you're not using the Application Kit's event loopÐ
a particularly safe thing to doÐyou should invoke this message (after invoking fetchInThread) to e
message binderDidFetch: is sent. The argument is the maximum amount of time, in seconds, to wa
Returns nil (and the message isn't sent) if the time limit expires before the fetch completes, otherw

fetchInThread

(id <DBContainers>)container

Returns the DBBinder's container object, as set through setContainer:. The container, which must
DBContainers protocol, holds the record objects that are created when the DBBinder fetches data.
default container and can operate without one, although this impedes some of the object's function
container, a DBBinder can't perform an asynchronous fetch, and its cursor can only be positioned
method.

setDynamicRecordSuperclass: class methods. This method has no effect and returns nil under the

·If the DBBinder's current prototype record object isn't nil.
·If the DBBinder has no properties.
·If the name set through setDynamicRecordClass: names an existing class.
·If the class named by setDynamicRecordSuperclass: doesn't exist.

Upon success, this method returns the class that it created.

This method is automatically invoked when the DBBinder fetches data, thus you needn't invoke it it's a good idea to never invoke this method however, if you doÐfor example, to examine the retu send a setRecordPrototype:nil message to the DBBinder before the next fetch to ensure that the co assembled.

(DBDatabase *)database

Returns the DBDatabase object that's associated with the DBBinder.

initForDatabase:withProperties:andQualifier:, setDatabase:

delegate

Returns the object that will receive notification messages for the DBBinder.

setDelegate:

delete

Deletes from the database each of the DBBinder's record objects.

Before the operation begins, a binderWillDelete: message is sent to the DBBinder's delegate (with argument) if the delegate message returns NO, then the deletion isn't performed and this method re records have been processed, the DBBinder is flushed. If the records were successfully deleted, a message is sent to the delegate and self is returned, otherwise the delegate message isn't sent and n

As each record is deleted, one of two messages is sent to the container's delegate (if the DBBinder container has a delegate, and if the delegate implements the method):

·binder:didAcceptObject: if the record was deleted.
·binder:didRejectObject: is sent if the record couldn't be deleted.

For both methods, the first argument is the DBBinder and the second is the record object. The valu methods are ignored.

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

Tells the adaptor to evaluate and execute the commands that are encoded in aString. The DBBind applied to the evaluation.

Fetches records from the database, forms a record object for each, and places the record objects in container. If the binder has no container, you should use the setNext method, rather than this one,

Before the fetch begins, the DBBinder's delegate is sent a binderWillFetch: message after, it's sen binderWillFetch: returns NO, the fetch isn't performed and this method immediately returns nil.

As each record of data is fetched, a copy of the DBBinder's prototype record object is created to h DBBinder's prototype record hasn't been set, a class is dynamically assembled to fill the need, as e description of createRecordPrototype.

The fetch continues until there's no more data to retrieve, or until the record limit (as set through t setMaximumRecordsPerFetch: method) has been reached.

After the fetch has ended, the DBBinder's cursor is set to the first record in the container (or to the there is no container) and self is returned. If there was no data to fetch, or if there's a fetch in prog has a container), the cursor isn't set and nil is returned.

If the fetch ended by exhausting the source dataÐin other words, it didn't end because the record l should then invoke cancelFetch to reclaim resources that were used during the fetch. Use the reco method to test whether the fetch ended because it reached the limit while there was more data to fe

fetchInThread

Fetches data asynchronously from the database by performing the fetch in a separate thread. The g conditions are as described in the fetch method, but with these differences:

·An asynchronous fetch only works if the DBBinder has a container.

·You shouldn't invoke cancelFetch after invoking this method unless you actually want to abort th

·The record limit set through setMaximumRecordsPerFetch:has no effect on an asynchronous fetch

If there is no container, or if the binderWillFetch: delegate message returns NO, then the fetch isn't method returns nil. Otherwise, this method returns self while the fetch proceeds in the backgroun complete, the binderDidFetch: method is sent to the delegate.

If you're not using the Application Kit's main event loop, then you probably don't want to fetch as are any number of things that you can do, in this situation, that will make your application hang. E lucky you should note that this method, when run without the App Kit, should be followed by an i checkThreadedFetchCompletion:. This synchronizes the fetch thread with the main thread, and en binderDidFetch: message is sent. Good luck.

To be used in an asynchronous fetch, the DBBinder's container must be thread-safe (it must be re- if you limit yourself to DBCursorPositioning methods, such as setTo: and setNext:, you can access regardless of the type of fetch employed.

fetch, cancelFetch, checkThreadedFetchCompletion:

(BOOL)flush

If flushing is enabled, this empties the DBBinder's container. Furthermore, if the DBBinder has b flush, the records that were in the container are freed and the prototype record object is set to nil. I flushing and free-on-flush are enabled. Returns YES if flushing is enabled, NO if not.

This method always interrupts a fetch, if one is in progress, whether or not flushing is enabled.

The following DBBinder methods may cause flush to be invoked:

setFlushEnabled:, setFreeOnFlush.

free

Frees the DBBinder and its records. If the DBBinder owns the prototype record object, it too is fre

(List *)getProperties:(List *)aList

Fills aList with the DBBinder's properties, then returns the List directly and by reference. The ord
the List is that by which they were added to the DBBinder. You mustn't free the contents of aList,
the List itself.

initForDatabase:withProperties:andQualifier:, setProperties:, addProperty

(BOOL)ignoresDuplicateResults

Returns YES if the DBBinder is set to ignore duplicate records during a select. The default is NO.
to support this (the Oracle and Sybase adapters supplied with the Database Kit do).

setIgnoresDuplicateResults:

init

The designated initializer for the DBBinder class, init initializes and returns the DBBinder. All the
DBBinder owns or knows of, such as its container, properties, DBDatabase, and DBQualifier are s
attributes are set as follows:

initForDatabase:withProperties:andQualifier

initForDatabase:aDBDatabase
withProperties:(List *)propertyList
andQualifier:(DBQualifier *)aDBQualifier

Invokes init and then sets the DBBinder's DBDatabase, properties, and DBQualifier as given by th
properties in propertyList are added to the DBBinder's own List, thus the argument may be freed.

init

insert

·binder:didAcceptObject: if the record was inserted.
·binder:didRejectObject: is sent if the record couldn't be inserted.

For both methods, the first argument is the DBBinder and the second is the record object. The valu
methods are ignored.

(BOOL)isFlushEnabled

Returns YES if the DBBinder has flushing enabled, otherwise return NO. The default is YES. See
flush method for more information. (Note that sharing a cursor is incompatible with flushing, so s
the side effect of disabling flushing.)

flush, setFlushEnabled:, setSharesContext:

(unsigned int)maximumRecordsPerFetch

Returns the maximum number of records that will be retrieved during a synchronous fetch, as set t
setMaximumRecordsPerFetch: method. By default, this limit is set to DB_NoIndex, which impose

setMaximumRecordsPerFetch:, recordLimitReached, fetch

(BOOL)ownsRecordPrototype

Returns YES if the DBBinder owns its prototype record objectÐin other words, if it will create a n
(when createRecordPrototype is invoked). If you've set the prototype record object yourself, throu
setRecordPrototype:, then this returns NO.

(unsigned int)positionInOrderingsFor:(id <DBProperties>)aProperty

Returns an integer that indicates the level (primary, secondary, tertiary, and so on) at which the giv
order the records that are retrieved from the database. The ordering position of a particular propert
it was added to the ordering mechanism (amongst the currently ªactiveº ordering properties) throu
addRetrieveOrder:for: method. A return of DB_NoIndex means that the property isn't used in the

addRetrieveOrder:For:

(DBQualifier *)qualifier

Returns the DBQualifier object that was set through setQualifier: or initForDatabase: withPropertie
qualifier is used to qualify values during a select.

setQualifier:, initForDatabase:withProperties:andQualifier:

read:(NXTypedStream *)stream

setMaximumRecordsPerFetch:, maximumRecordsPerFetch, fetch

recordPrototype

Returns the DBBinder's prototype record object. If you've set the object yourself, through setRec
object is returned. Otherwise, this returns nil unless you've previously invoked createRecordProto
this is called from within a subclass implementation of fetch.

setRecordPrototype, createRecordPrototype

removePropertyAt:(unsigned int)index

Removes the property at the given index. To find the index of a particular property, get the DBBi
through the getProperties: method, and then ask for the index by sending indexOf: to the List, pass
argument. Returns the property (or nil if there was none).

setProperties:, addProperty:

removeRetrieveOrderFor:(id <DBProperties>)aProperty

Removes the given property from the list of properties that are used to sort records as they're being
property's retrieve order constant is set to DB_NoOrder. Returns nil if the property hadn't previou
record-sorting list (if it hadn't previously received an addRetrieveOrderFor: message), otherwise s

addRetrieveOrderFor:, positionInOrderingsFor:

reset

Restores the DBBinder to a virgin state. The DBBinder is first flushed (which cancels a fetch, if o
the objects that it has allocated, and any that you've allocated in the scratch zone, are freed. The s
methods automatically cause a reset.

flush, scratchZone

(DBRetrieveOrder)retrieveOrderFor:(id <DBProperties>)aProperty

Returns a constant that indicates the order in which records are retrieved when aProperty is used as
addRetrieveOrder:for: method for a further explanation). The retrieval order constants are:

addRetrieveOrder:for:, positionInOrderingsFor:

select

Selects and fetches data from the database. First, selectWithoutFetching is invoked if that returns
If the method was successful, then fetch is invoked the value returned by fetch is returned by this

selectWithoutFetching, fetch

selectWithoutFetching

Selects records from the database, using the DBBinder's qualifier (as set through setQualifier: or i
withProperties:andQualifier:) to qualify the records that are selected.

Before the operation begins, a binderWillSelect: message is sent to the DBBinder's delegate (with
argument) if the delegate message returns NO, then the select isn't performed and nil is immediate
method. Otherwise, the DBBinder is flushed and the data is selected. If the select was successful,
message is sent to the delegate and self is returned, otherwise the delegate message isn't sent and

If the DBBinder is set to ignore duplicate results, only the first of duplicate records will be selecte

select, setIgnoreDuplicateResults

setContainer:(id <DBContainers>)anObject

Sets the container that's used to store record objects. The argument must either adopt the DBCont
can be a List objectÐDBBinder defines a category of List that allows its instances, and those of its
DBContainers-conforming objects. Most DBBinders are well served using a List as a container.
and practice of containment, see the class description, above.

Returns the previous container.

setDatabase:(DBDatabase *)aDatabase

Sets the DBBinder's database. Returns the previous DBDatabase object.

setDelegate:anObject

Sets the object that receives notification messages for the DBBinder.

setFlushEnabled:(BOOL)flag

Establishes whether the DBBinder is capable of being flushed, as explained in the description of th
default is YES.

flush, setFreeObjectOnFlush:

setIgnoresDuplicateResults:(BOOL)flag

Establishes whether duplicate records are ignored during a select. The default is NO. It's up the a
the Oracle and Sybase adapters supplied with the Database Kit do.

ignoresDuplicateRecords, selectWithoutFetching

setMaximumRecordsPerFetch:(unsigned int)limit

Sets, to limit, the maximum number of records that will be retrieved during a fetch. The limit only
fetches the asynchronous fetch method fetchInThread ignores the record limit.

maximumRecordsPerFetch, recordLimitReached, fetch

(List *)setProperties:(List *)aList

Resets the DBBinder and then adds to it the properties in aList. Returns the argument.

getProperties:, addProperty:, removePropertyAt:

setQualifier:(DBQualifier *)aQualifier

Sets the qualifier that's used during a select. Returns self.

qualifier

setRecordPrototype:anObject

Sets the object that's copied to store the results of a fetch. See the class description for a full expla
prototype object.

recordPrototype, createRecordPrototype

setSharesContext:(BOOL)flag

Establishes whether the DBBinder shares its cursor with other DBBinder objects. The default is N
DBBinder share its cursor disables flushing. Returns self.

Shared cursor behavior depends on the implementation of the adaptor rather than the database it's
Oracle and the Sybase adaptors as a way of achieving atomic updates. Sharing the cursor also prov
efficient use of memory.

sharesContext

(BOOL)sharesContext

Before the operation begins, a binderWillUpdate: message is sent to the DBBinder's delegate (with argument) if the delegate message returns NO, then the update isn't performed and nil is immediat method. After all the records have been processed, the DBBinder is flushed. If the records were s binderDidUpdate: message is sent to the delegate and self is returned, otherwise the delegate messa returned.

As each record is updated, one of two messages is sent to the container's delegate (if the DBBinde container has a delegate, and if the delegate implements the appropriate method):

·binder:didAcceptObject: if the record was updated.
·binder:didRejectObject: is sent if the record couldn't be updated.

For both methods, the first argument is the DBBinder and the second is the record object. The valu methods are ignored.

(DBValue *)valueForProperty:(id <DBProperties>)aProperty

Returns a DBValue object for the given property of the currently pointed-to record. Use the DBCu methods, such as setNext and setTo:, to set the cursor to point to a particular record. The object th by the DBBinder and shouldn't be freed.

write:(NXTypedStream *)stream

Writes the DBBinder to the typed stream stream. Returns self.

binder:aBinder didEvaluateString:(const unsigned char *)aString

Invoked after the given string has been successfully evaluated by DBBinder's evaluateString: meth ignored.

(BOOL)binder:aBinder willEvaluateString:(const unsigned char *)aString

Invoked before the given string is evaluated by DBBinder's evaluateString: method. A return of N evaluation.

binderDidDelete:aBinder

Invoked after the DBBinder has successfully deleted records through the delete method. The retur

binderDidFetch:aBinder

binderDidSelect:aBinder

Invoked after the DBBinder has successfully selected data through the selectWithoutFetching meth
ignored.

binderDidUpdate:aBinder

Invoked after the DBBinder has successfully updated the database through the update method. Th
ignored.

(BOOL)binderWillDelete:aBinder

Invoked before the DBBinder attempts to delete records from the database through the delete meth
will thwart the attempt.

(BOOL)binderWillFetch:aBinder

Invoked before the DBBinder attempts to fetch data through the fetch or fetchInThread method. A
thwart the attempt.

(BOOL)binderWillInsert:aBinder

Invoked before the DBBinder attempts to insert records into the database through the insert metho
thwart the attempt.

(BOOL)binderWillSelect:aBinder

Invoked before the DBBinder attempts to select data from the database through the selectWithoutF
return of NO will thwart the attempt.

(BOOL)binderWillUpdate:aBinder

Invoked before the DBBinder attempts to update the database through the update method. A return
attempt.