GNUstep Database Library Introduction

©2006 Free Software Foundation

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions.

Table of Contents

1 Introduction **1**

2 Concepts **2**

3 Key Value Coding **3**

4 Setting values through KVC **4**

5 Accessing values through KVC **5**

6 Key Paths **6**

7 Type promotion **7**

8 Class specific implementation **8**

9 Classes **9**

10 Model classes **10**

11 Database specific classes **11**

12 Data oriented classes **12**

13 EOModel class **13**

13.1 overview 13

14 EOEntity class **14**

14.1 overview 14
14.2 Class properties 14
14.3 Class name 14
14.4 Primary Key Attributes 14
14.5 External name 14

15 EOAttribute class 15
15.1 overview 15
15.2 Name 15
15.3 Column name 15
15.4 Adaptor value type 15
15.5 External type 16
15.6 Value type 16
15.7 Value class name 16
15.8 Value factory method name 16
15.9 Value factory argument type 16

16 EORelationship class 17
16.1 overview 17

17 EOModelGroup class 18
17.1 overview 18

18 EOAdaptor class 19
18.1 overview 19

19 EOAdaptorContext class 20

20 EOAdaptorChannel class 21

21 EODataSource class 22
21.1 overview 22
21.2 Fetch objects 22
21.3 Creating objects 22
21.4 Inserting objects 22
21.5 Deleting objects 22
21.6 Qualified DataSources 22
21.7 EODatabaseDataSource class 23

22 EOEditingContext class 24
22.1 overview 24

23 EOGenericRecord class 25
23.1 overview 25

24 Model Creation 26
24.1 Example model file 26
24.2 Creating with DBModeler 28

25 Creating a project. 30
25.1 Creating a makefile 30
25.2 Adding Resource Files 30
25.3 A complete GNUmakefile 30

26 Database creation 31
26.1 Creating the database with DBModeler 31

27 Database connection 32

28 Working with data 33

29 Adding some data. 34

30 Working with relationships 36

31 EOInterface 38

32 Introduction 39

33 EODisplayGroup class 40

34 EOAssociation class 41

Index 42

1 Introduction

This document is intended to get people started developing with GDL2. A knowledge of objective-c and relational database concepts is assumed.

While not intended as a thorough reference or a replacement for the API docs and surely omits details for the sake of simplicity it attempts to provide a starting point for people unfamiliar with GDL2 or EOF to get started developing their first application.

If you are reading this document from the GDL2 Sources, most example sources are provided in the ../../Examples/ directory.

2 Concepts

3 Key Value Coding

Key Value Coding is a concept used widely throughout GDL2, it provides a mechanism by where you can access and modify an objects set/accessor methods or even instance variables directly, through a named key.

Additionally some classes may implement KVC in a way specific to the class.

4 Setting values through KVC

Setting values through key value coding will try to call a method '-setKeyName:' with the value as the parameter to -setKeyName: as a parameter failing that, if anObject had an instance variable with the same name as the key that would be modified directly.

If anObject does not respond to '-setKeyName:' and there is no instance variable with the same name as the key, an exception is thrown.

```
[anObject setValue:@"bar" forKey:@"foo"];
```

Will first try to call -setFoo: then attempt to set the instance variable named "foo" to "bar".

5 Accessing values through KVC

Accessing values through Key Value Coding first attempts to call the -keyName method on anObject if it responds. If the object does not respond then it will try to access an instance variable with the name of the key.

If there is no method or instance variable with the name of the key an exception will be thrown.

For example,

```
[anObject valueForKey:@"foo"];
```

Will first try to call -foo, then attempt to return instance variable named foo.

6 Key Paths

Key paths are a list of keys separated by a dot.

The first key accesses the key on the target object through normal KVC, and each subsequent key is sent to the object returned through the previous key in the list.

For example,

```
[anObject valueForKeyPath:@"foo.bar"];
```

Will be equivalent to

```
[[anObject valueForKey:@"foo"] valueForKey:@"bar"];
```

7 Type promotion

When a accessing a key, you may access keys for things such as standard c numerical types, and they will be automatically promoted to their object equivalent

For example:

```
[@"foo" valueForKey:@"length"];
```

Returns a NSNumber object containing '3'.

8 Class specific implementation

By implementing valueForKey: and setValueForKey: classes can implement functionality to contain keys in an instance variable such as a dictionary, but they can also implement something to work on a collection of objects.

For instance NSArray implements KVC to forward key value coding to all objects in the array.

Suppose we have an array contain a few string objects.

```
("Example", "array", "containing", "strings")
```

If we get the value for the key length, it will return an NSArray of NSNumbers

```
(7, 5, 10, 7).
```

9 Classes

10 Model classes

The model related classes are important in that they define a databases structure. Giving GDL2 a way to map a relational database into a set of objects.

11 Database specific classes

The database specific classes loadable through bundles provide a method for GDL2 to connect to and abstract implementation details between different database implementations.

Currently adaptors for SQLite3 and PostgreSQL exist.

12 Data oriented classes

The data oriented classes relate to actual data manipulation and management.

13 EOModel class

13.1 overview

A model represents GDL2s interface to a database. It contains information required to connect to the database along with entities and stored procedures.

All the model classes can be written to and read from property list files in the form of .eomodel or .eomodeld files. While .eomodel files contain a model and all its entities and objects in a single property list, .eomodeld files are a directory with each of the property lists in their own file.

Typically you won't create an model through manual instantiation of the classes but store them in and read them from a property list. We have provided an example .eomodel file See Section 24.1 [Example model file], page 26.

An EOModel Typically has:

1. A Name
2. An adaptor name
3. A connection dictionary
4. An array of entities

14 EOEntity class

14.1 overview

An entity contains information pertaining to a table in a database in the form of attributes and relationships.

Additionally an entity contains:

1. An array of class properties
2. An array of primary key attributes
3. A class name
4. An External name

14.2 Class properties

A class property of an entity can be either an attribute or a relationship. typically class properties are the set of attributes or relationships which are user visible and need to be set or accessed by the user. Primary and Foreign keys attributes are usually derived from other attributes or generated automatically and so they are not typically class properties.

A class property will be available through Key Value Coding for access and modification, in an instance of an Enterprise object.

14.3 Class name

an EOEntity's class name represents the name of the class which will be instantiated when creating an Enterprise Object such as EOGenericRecord or a custom object.

14.4 Primary Key Attributes

Primary key attributes specify which attributes uniquely identify a row in the table, they are typically generated automatically by GDL2. They correspond directly to the relational database concept.

14.5 External name

The external name represents the table name in the database server, and in any SQL the adaptor might generate.

15 EOAttribute class

15.1 overview

An attribute typically maps a table column to an instance variable, in which case the attribute is a class property. Some attributes represent foreign keys which are used to create realationships yet do not correspond to a property in the enterprise object. Other attributes may represent primary keys which needn't be class property either. In fact some parts of framework work more smoothly if primary key attributes and foreign key attributes are not class properties.

Attributes typically contain:

1. A name
2. A column name
3. An adaptor value type
4. An external type
5. A value type
6. A value class name
7. A value factory method name
8. a factory method argument type

Some additional properties an attribute may have:

1. Read only
2. Allows null
3. Width
4. Precision
5. Scale

15.2 Name

The attributes name when the attribute is a class property is used as the key when doing key value coding on an enterprise object.

It also uniquely identifies the attribute in its entity there many not be an attribute with the same name as another attribute or relationship in an entity.

15.3 Column name

The adaptor uses the column name in generating SQL.

15.4 Adaptor value type

Indicates the type of the attribute as contained in the database

Valid adaptor value types are:

1. EOAdaptorNumberType
2. EOAdaptorCharactersType

3. EOAdaptorBytesType
4. EOAdaptorDateType

Corresponding to numerical, string, raw data, and date value types.

15.5 External type

An external type is a string representing an adaptor specific database type different adaptors may use different names where the PostgreSQL adaptor might use 'char'. The SQLite3 Adaptor might use 'TEXT'

it gives you full control on how the data is stored in the specific adaptor where the adaptor value type allows you to specify a few generic values.

15.6 Value type

Value types are a string with a single character such as 'f' for floats 'c' for chars a full list of the standard types is available in the GDL2 API reference for EOAttributes -valueType method.

The value type allows you to further refine the adaptor value type where EOAdaptor-NumberType might represent a integer, float, or double type.

15.7 Value class name

The value class name specifies the class which will be present in an Enterprise Object containing the attribute.

A property of this class will be instantiated when a field is retrieved from the database, similarly a instance of this will be converted into the external type when being sent to the datbase server.

15.8 Value factory method name

When the Value Class Name is a custom object for instance NSImage created from a blob of data. The value factory method name denotes the initializer for the class, used to create a new instance of the custom class.

The value class name is an NSString representing a selector accepting a single argument suitable for passing to the NSSelectorFromString function.

15.9 Value factory argument type

This is the type of the argument sent to the value factory method name.

Valid types are

1. EOFactoryMethodArgumentIsNSData
2. EOFactoryMethodArgumentIsNSString
3. EOFactoryMethodArgumentIsBytes

16 EORelationship class

16.1 overview

A relationship represents a connecton between entities and are described with EOJoin's. A join defines source and destination attributes – The attributes of the joining entity which must match.

A relationship may be of type to-one or to-many. In a to-many the destination will be an array of objects, and a to-one relationships destination a single object.

Typically a relationship is a class property. Yet some relationships may soley be used for flattening other relationships which are class properties, yet need not be class properties themselves.

17 EOModelGroup class

17.1 overview

When models have relationships to other models, they ask their model group.

There is a special model group - the default model group - which contains all the models in the applications resources and the resources of any frameworks the application uses. If your model file is not available through application or framework resources you will need to add it to a model group.

18 EOAdaptor class

18.1 overview

An adaptor abstracts the difference between different database implementations. It can connect to the database with the help of a connection dictionary and create and execute SQL statements.

While an adaptor is made up of many different classes. The EOAdaptor class is sort of an entry point into the different available classes.

And a typical use for the EOAdaptor class is creating an instance of a specific adaptor, either by name or through the adaptor name in a model.

Typical methods for the EOAdaptor class are:

1. -createAdaptorContext
2. -runLoginPanel
3. -assertConnectionDictionaryIsValid
4. +adaptorWithModel:

19 EOAdaptorContext class

An EOAdaptorContext can create an adaptor channel and will transparently handle transactions for the channel, It can begin, commit, roll back transactions.

Additionaly you can enable debugging on the context and its channels.

Typical methods for an EOAdaptorContext:

1. -createAdaptorChannel
2. -setDebugEnabled:

20 EOAdaptorChannel class

An adaptor channel can open and close a connection to the adaptors database server. Along with fetch rows from the database and create, update, and delete rows in the database.

It is the main communication channel for gdl2, in creating the connection to the database, and executing any SQL statements which have been prepared through EOSQLExpression. Though it also has methods for building SQL expressions from entities, and possibly turning the results back into enterprise objects.

Because EOAdaptorChannel can create most SQL statements for you, you'll rarely need to do that yourself, though it is available if needed.

Typical methods for an EOAdaptorChannel:

1. -openChannel
2. -closeChannel
3. -isOpen

21 EODataSource class

21.1 overview

EODataSource is an abstract base class, and implements no real functionality on its own, instead you'll access EODataSource subclass.

A data source represents a collection of rows inside of a table. It can create rows, delete and provide access to the individual rows represented as Enterprise objects.

Typical methods for an EODataSource subclass:

1. -fetchObjects
2. -createObject:
3. -deleteObject:
4. -insertObject:
5. -dataSourceQualifiedByKey:

21.2 Fetch objects

The -fetchObjects method will return an array of enterprise objects. Typically these will be retrieved directly from data in the database server. Then the caller will save the array for access or modification.

21.3 Creating objects

The -createObject: method will create a new enterprise object for insertion into the database. A subclass will generally insert this new object into an editing context. Though the caller is responsible for inserting the object into the data source with -insertObject:.

21.4 Inserting objects

The -insertObject: method will schedule the object for addition into the database server EditingContexts changes are saved to the database.

21.5 Deleting objects

The -deleteObject: method will schedule the object for removal from the database server when the EOEditingContexts changes are saved to the database.

21.6 Qualified DataSources

Subclasses may implement this method to return a detail data source.

A detail data source is a datasource which is created from following a relationship in an object of the receiever: the master object.

in our example you might have a data source for the authors entity and qualify a detail data source, with the toBooks key.

21.7 EODatabaseDataSource class

EODatabaseDataSource class is a subclass of EODataSource.

To initialize an EODatabaseDataSource you'll give it a reference to an EOEditingContext and an entity name.

EODatabaseDataSource initializers:

1. -initWithEditingContext:entityName:
2. -initWithEditingContext:entityName:fetchSpecificationName:

Once initialized, you can call the EODataSource methods on it, to create fetch insert, and delete objects from the datasource.

22 EOEditingContext class

22.1 overview

An editing context is responsible for managing changes to enterprise objects and provides the ability to save and undo those changes. Including inserts, updates, and deletes.

Typical methods of the EOEditingContext class:

1. -saveChanges:
2. -revert:
3. -undo:
4. -redo:
5. -insertObject:
6. -deleteObject:

You may have noticed that there is no mention of a method for modifying an object through an EOEditingContext. As you will modify the objects directly, and EOEditing-Context will merely take note of the changes, and save snapshots of the objects as they are being modified so you can undo those changes.

23 EOGenericRecord class

23.1 overview

EOGeneric record represents an actual row in a table being the default enterprise object it contains no custom business logic and is accessible solely through key value coding.

Where an entity represents the description of the table. It's columns and types. enterprise objects represent the data contained in the table.

EOGenericRecords are generally created with a reference to an entity. They export as keys the class properties of the entity, for access and modification.

If you have an EOGenericRecord from the 'authors' entity of our example model you could set the authors name as so. See Section 24.1 [Example model file], page 26.

```
[anAuthor takeValue:@"Anonymous" forKey:@"name"];
```

And retrieve the author name with:

```
[anAuthor valueForKey:@"name"];
```

24 Model Creation

Models can be created in 3 ways

1. Manual written with property lists.
2. Hard coding the model in objective-c.
3. Creation of plists with the DBModeler application.

while DBModeler provides the easiest way, followed by manually writing the property lists, and hard coding the model is both tedious and complicated.

24.1 Example model file

Below is a example property list model created with DBModeler, it contains a Model for a library 2 entities, author and book

author contains 2 attributes, authorID the primary key number, and name a string book contains 3 attributes, bookID the primary key number, authorID a foreign key number, and title a string.

author and book each contain a relationship author a to-many relationship to each of the authors books, and book a to-one relationship to the books author for the sake of demonstration i'm ignoring books with multiple authors.

it also contains an adaptor name, and an adaptor specific connection dictionary.

```
{
    EOModelVersion = 2;
    adaptorName = SQLite3;
    connectionDictionary = {
        databasePath = "/tmp/example.db";
    };
    entities = (
        {
            attributes = (
                {
                    columnName = authorID;
                    externalType = integer;
                    name = authorID;
                    valueClassName = NSNumber;
                },
                {
                    columnName = name;
                    externalType = varchar;
                    name = name;
                    valueClassName = NSString;
                }
            );
            className = EOGenericRecord;
            classProperties = (
                name,
```

```
            toBooks
        );
        externalName = authors;
        name = authors;
        primaryKeyAttributes = (
            authorID
        );
        relationships = (
            {
                destination = books;
                isToMany = Y;
                joinSemantic = EOInnerJoin;
                joins = (
                    {
                        destinationAttribute = authorID;
                        sourceAttribute = authorID;
                    }
                );
                name = toBooks;
            }
        );
    },
    {
        attributes = (
            {
                columnName = authorID;
                externalType = integer;
                name = authorID;
                valueClassName = NSNumber;
            },
            {
                columnName = bookID;
                externalType = integer;
                name = bookID;
                valueClassName = NSNumber;
            },
            {
                columnName = title;
                externalType = varchar;
                name = title;
                valueClassName = NSString;
            }
        );
        className = EOGenericRecord;
        classProperties = (
            title,
            toAuthor
```

```
            );
            externalName = books;
            name = books;
            primaryKeyAttributes = (
                bookID
            );
            relationships = (
                {
                    destination = authors;
                    isToMany = N;
                    joinSemantic = EOInnerJoin;
                    joins = (
                        {
                            destinationAttribute = authorID;
                            sourceAttribute = authorID;
                        }
                    );
                    name = toAuthor;
                }
            );
        }
    );
    name = library;
}
```

24.2 Creating with DBModeler

To recreate the example model with DBModeler,

select Document, New from the main menu then property Add entity twice. set the name and external names to 'authors' and 'books'

select Document, Set Adaptor Info, and select SQLite, and click Ok, this will bring up the SQLite login panel, where you need to provide it a path for where to create the model file.

Each Adaptor will have different requirements, so each login panel is quite different. Other adaptors may have a server address, username, and database names.

select the authors entity in the outline view, after expanding the model add an attribute to authors by selecting Property, Add attribute set the name and column name to 'authorID', and select the switch button with a key icon, to set it as the primary key for the entity. Set the value class name to NSNumber and the external type to INTEGER

Add another entity, set the name and column names to 'name'. Select the switch button which looks like a jewel icon to set it as a Class Property. Set the Value Class Name to NSString and external type to TEXT.

now do the same with books, name them bookID, authorID, and title. make sure bookID is set as the primary key not authorID in the books entity. And that title is set as a class property.

title is a NSString/TEXT, where authorID and bookID are NSNumber/INTEGER

now add a relationship to authors name it toBooks, and Tools, inspector in the destination table, select To many, and books as the destination entity.

select authorID as the source and destination attributes

add a relationship to books, name it toAuthor. Select author as the destination entity, and authorID as the source and destination attributes.

The select Document, Save, from the main menu.

25 Creating a project.

25.1 Creating a makefile

Creating a GNUmakefile for a GDL2 Project is done throught he well documented gnustep-make makefile system.

they are standard GNUmakefiles but you'll need to include a special file – gdl2.make after common.make

E.g.

```
include $(GNUSTEP_MAKEFILES)/common.make
include $(GNUSTEP_MAKEFILES)/Auxiliary/gdl2.make
```

25.2 Adding Resource Files

Make sure you add your .eomodel or .eomodeld file to your projects resources

```
APP_NAME=foo
foo_RESOURCE_FILES=foo.eomodeld
```

25.3 A complete GNUmakefile

```
include $(GNUSTEP_MAKEFILES)/common.make
include $(GNUSTEP_MAKEFILES)/Auxiliary/gdl2.make

TOOL_NAME=eoexample
eoexample_OBJC_FILES=eoexample.m
eoexample_RESOURCE_FILES=library.eomodel
include $(GNUSTEP_MAKEFILES)/tool.make
```

26 Database creation

Now that we have created a model file, we need to generate the SQL to create the database.

26.1 Creating the database with DBModeler

Select, Generate SQL from the Tools menu, then select the appropriate check boxes,

Create databases, Create tables, foreign key constraints, primary key constraints, and primary key support.

then either save the SQL to a file, or execute it, you may need to login to the database server, but the adaptor for the model should bring up a login panel.

27 Database connection

An example which connects to and then disconnects from the database. provided you have already created the database in previous section

```
#include <Foundation/Foundation.h>
#include <EOAccess/EOAccess.h>
#include <EOControl/EOControl.h>

int
main(int arcg, char *argv[], char **envp)
{
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
  EOModelGroup *modelGroup = [EOModelGroup defaultGroup];
  EOModel *model = [modelGroup modelNamed:@"library"];
  EOAdaptor *adaptor;
  EOAdaptorContext *context;
  EOAdaptorChannel *channel;

  /* Tools don't have resources so we have to add the model manually. */
  if (!model)
    {
      NSString *path = @"./library.eomodel";
      model = [[EOModel alloc] initWithContentsOfFile: path];
      [modelGroup addModel:model];
      [model release];
    }

  adaptor = [EOAdaptor adaptorWithName:[model adaptorName]];
  context = [adaptor createAdaptorContext];
  channel = [context createAdaptorChannel];

  [channel openChannel];

  /* insert code here */

  [channel closeChannel];
  [pool release];
  return 0;
}
```

28 Working with data

29 Adding some data.

Here we have more complete example which writes a record to the database, then fetches the record and updates it and saves the data again, then removes the record.

```
#include <Foundation/Foundation.h>
#include <EOAccess/EOAccess.h>
#include <EOControl/EOControl.h>

int
main(int arcg, char *argv[], char **envp)
{
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
  EOModelGroup *group = [EOModelGroup defaultGroup];
  EOModel *model;
  EOAdaptor *adaptor;
  EOAdaptorContext *context;
  EOAdaptorChannel *channel;
  EOEditingContext *ec;
  EODatabaseDataSource *authorsDS;
  NSArray *authors;
  id author;

  model = [group modelNamed:@"library"];

  /* Tools don't have resources so we have to add the model manually */
  if (!model)
    {
      NSString *path = @"./library.eomodel";
      model = [[EOModel alloc] initWithContentsOfFile: path];
      [group addModel:model];
      [model release];
    }

  adaptor = [EOAdaptor adaptorWithModel:model];
  context = [adaptor createAdaptorContext];
  channel = [context createAdaptorChannel];
  ec = [[EOEditingContext alloc] init];
  authorsDS
    = [[EODatabaseDataSource alloc] initWithEditingContext: ec
                                    entityName:@"authors"];

  [channel openChannel];

  /* Create a new author object */
  author = [authorsDS createObject];
  [author takeValue:@"Anonymous" forKey:@"name"];
  [authorsDS insertObject:author];
```

```
    [ec saveChanges];


    /* Fetch the newly inserted object from the database */
    authors = [authorsDS fetchObjects];
    NSLog(@"%@", authors);

    /* Update the authors name */
    [[authors objectAtIndex:0]
          takeValue:@"John Doe" forKey:@"name"];
    [ec saveChanges];

    NSLog(@"%@", [authorsDS fetchObjects]);

    [channel closeChannel];
    [pool release];
    return 0;
}
```

30 Working with relationships

Heres another more complex example of working with data, we'll add an author, and some books, and then traverse the relationship in a couple of different ways.

```
#include <Foundation/Foundation.h>
#include <EOAccess/EOAccess.h>
#include <EOControl/EOControl.h>

int
main(int arcg, char *argv[], char **envp)
{
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
  EOModelGroup *group = [EOModelGroup defaultGroup];
  EOModel *model;
  EOAdaptor *adaptor;
  EOAdaptorContext *context;
  EOAdaptorChannel *channel;
  EOEditingContext *ec;
  EODatabaseDataSource *authorsDS;
  EODataSource *booksDS;
  id author;
  id book;

  model = [group modelNamed:@"library"];

  /* Tools do not have resources so we add the model manually.  */
  if (!model)
    {
      NSString *path = @"./library.eomodel";
      model = [[EOModel alloc] initWithContentsOfFile: path];
      [group addModel:model];
      [model release];
    }

  adaptor = [EOAdaptor adaptorWithModel:model];
  context = [adaptor createAdaptorContext];
  channel = [context createAdaptorChannel];
  ec = [[EOEditingContext alloc] init];
  authorsDS
    = [[EODatabaseDataSource alloc] initWithEditingContext: ec
                                    entityName:@"authors"];

  [channel openChannel];

  author = [authorsDS createObject];
  [author takeValue:@"Richard Brautigan" forKey:@"name"];
  [authorsDS insertObject:author];
```

```
    booksDS = [authorsDS dataSourceQualifiedByKey:@"toBooks"];
    [booksDS qualifyWithRelationshipKey:@"toBooks" ofObject:author];

    book = [booksDS createObject];
    [book takeValue:@"The Hawkline Monster" forKey:@"title"];
    [booksDS insertObject:book];

    book = [booksDS createObject];
    [book takeValue:@"Trout Fishing in America" forKey:@"title"];
    [booksDS insertObject:book];

    [ec saveChanges];

    /* log the to many relationship from author to books */
    NSLog(@"%@ %@",
          [author valueForKey:@"name"],
          [author valueForKeyPath:@"toBooks.title"]);

    /* log the to one relationship from book to author */
    NSLog(@"%@", [book valueForKeyPath:@"toAuthor.name"]);

    /* traverse to one through the to many through key paths
       logging the author once for each book. */
    NSLog(@"%@", [author valueForKeyPath:@"toBooks.toAuthor.name"]);

    [channel closeChannel];
    [pool release];
    return 0;
}
```

31 EOInterface

32 Introduction

With GDL2 and EOInterface you can develop graphical applications using the gnustep gui libraries. It provides the ability to create connections between records from the database, and graphical controls.

Once a connection has been made between the graphical control and the record, EOInterface will update the record when the data changes in the control, and update the control when the data or the selection changes. EOInterface is composed of the EODisplayGroup class and EOAssociation subclasses.

EODisplayGroup contains the records and manages the selection, and notifies EOAssociations when the selection or selected record changes.

EOAssociation subclasses, associate graphical controls to the display group displaying the data in the display group, and updating the display group when the control changes the data. Multi-record associations such as table views can change the display groups selection.

33 EODisplayGroup class

EODisplayGroup has an EODataSource, and can fetch and create objects, manage the selection, filter the objects for display with qualifiers, and sort them with EOSortOrderings.

If you have loaded the GDL2Palette into Gorm you can create an EODisplayGroup by dragging an entity or a relationship from the outline view in DBModeler, to the document window in Gorm the display group will be associated with an EODataSource and will be encoded/decoded to and from the .gorm file. It will be a top level object, visible in the 'Objects' view of the gorm document. With the name of the entity or relationship dropped.

You can create connections from controls directly to the display group, for example you could connect a button or menu item to EODisplayGroups -selectNext: action by: Selecting the control and control-drag from the control to the display group. In the connect inspector, select -selectNext: and click 'connect'.

Available actions for EODisplayGroup:

1. -fetch:
2. -selectNext:
3. -selectPrevious:
4. -delete:
5. -insert:

Manual creation of a EODisplayGroup by initializing the display group and setting its dataSource:

```
EODisplayGroup *dg;
EODataSource *dataSource;

dg = [[EODisplayGroup alloc] init];
[dg setDataSource:dataSource];
```

34 EOAssociation class

An EOAssociation is an abstract base class. Subclasses of EOAssociation can be created to connect properties of an object in an EODisplayGroup to graphical controls. EOControls contain aspects, objects, and keys, and display groups.

Where the object is a graphical control, the key, being a key appropriate for KVC on an enterprise object, and the aspect is a string describing the use for the key. Each association has their own set of aspects and the aspects supported may vary between different association classes.

Manual creation of an EOControlAssocation:

```
EOAssociation *association;
EODisplayGroup *authorDG;
NSTextField *nameField;

association = [[EOControlAssociation alloc] initWithObject:nameField];
[association bindAspect:@"value" displayGroup:authorDG key:@"name"];
[association establishConnection];
[association release];
```

A few things of note, You can bind multiple aspects to an association. When the connection is broken the association will be released. When 'nameField' is deallocated, the connection will automatically be broken.

EOAssociations can be created transparently by Gorm with the GDL2Palette. To create an association with Gorm, Select a control and control-drag from a control to an EODisplayGroup.

In the connect inspector there is a pop up button which contains a list of the association classes which are usable with the control. Select an association class and the first column in the browser changes to a list of the aspects available. Selecting an aspect in the browser and the second column in the browser will list the available class properties connectable to the aspect.

Unfortunately while not all association classes and aspects are implemented. They will unfortunately show up in the connect inspector.

Index

C

class, EOAdaptor 19
class, EOAttribute 15
class, EODataSource 22
class, EOEditingContext 24
class, EOEntity 14
class, EOGenericRecord 25
class, EOModel 13
class, EOModelGroup 18
class, EORelationship 17

K

KVC, Key Value Coding 3

M

model creation 26