Dynace

ODBC User Manual
Version 4.02
June 25, 2002

by Blake McBride

This manual was typeset with the TEX typesetting system developed by Donald Knuth.

1 ODBC Tutorial

1.1 Opening and Closing a Database

Prior to any use of a database, it must be opened. Two functions are available to accommodate the function as follows:

`OpenDatabase::Database` [OpenDatabase]

```
DB = gOpenDatabase(Database, source, id, pw)

object  DB;             /*  database object  */
char    *source;        /*  data source name as defined with
                            the ODBC administrator  */
char    *id;            /*  the user id to log into the
                            database with  */
char    *pw;            /*  the password associated with
                            the user id   */
```

This will open the specified database using the specified user id and password. `NULL` is returned of the database cannot be opened.

`QueryDatabase::Database` [QueryDatabase]

```
DB = gQueryDatabase(Database, wind)

object  DB;             /*  database object  */
object  wind;           /*  parent window object or NULL  */
```

This will query the user to select a database and then open the specified database using the user ID and password entered using the ones configured with the ODBC administrator.

The following method is used to close a database.

`Dispose::Database` [Dispose]

```
DB = gDispose(DB)

object  DB;             /*  database object  */
```

This procedure will close the database and `NULL` out the `DB` variable.

1.2 Statements

1.2.1 Introduction

One concept which is important to fully understand is the *Statement* object. A Statement object allows the program a means to execute an SQL command and then have access to the data returned by that particular command. Once a command's data is no longer needed, the Statement object may be re-used for other (and arbitrary) SQL commands. A single database may have numerous Statement objects associated with it at any given point. This number is only limited by the particular database vendor.

1.2.2 Creating and Disposing of Statement Objects

`NewStatement::Database` [NewStatement]

```
stmt = gNewStatement(DB);

object  DB;     /*  the database object      */
object  stmt;   /*  a new statement object   */
```

This will create a new statement object;

`Dispose::Statement` [Dispose]

```
gDispose(stmt);

object  stmt;   /*  the statement object   */
```

This will dispose of the statement object;

1.3 Adding Records

1.3.1 Overview

Adding of records is accomplished in three steps as follows.

1. Select the file or table in which records are to be added. This only needs to be done once even if multiple records are to be added.
2. Assign the data to the fields.
3. Add the record.

Once a record is added, additional records may be added by looping through steps 2 and 3. Data assigned to a field which is invariant need not be re-assigned.

1.3.2 Selecting the Table

Selecting the table can be accomplished in two ways. The most common way would be via the `gInsert` method. Alternatively, if you are already in the middle of a select command in which all the fields are selected in the desired table and there are no joins, as follows:

```
gDBSelect(stmt, "select * from mytable");
```

you need not perform any additional function. The table is already selected.

`Insert::Statement` [Insert]

```
r = gInsert(stmt, table);

object  stmt;   /*  the statement object  */
char    *table; /*  the table to use      */
int     r;      /*  0 on success          */
```

This method selects a table for subsequent record adds.

1.3.3 Assigning Field Data

Once the desired table is selected, field data may be assigned using the following methods.

```
stmt = gFldSetString(stmt, fld, str);

stmt = gFldSetChar(stmt, fld, ival);

stmt = gFldSetShort(stmt, fld, ival);

stmt = gFldSetUnsignedShort(stmt, fld, uival);

stmt = gFldSetLong(stmt, fld, lval);

stmt = gFldSetDouble(stmt, fld, dval);


object  stmt;   /*  the statement to operate on  */
char    *fld;   /*  the field name               */
int     ival;   /*  an integer value             */
unsigned int uival;  /*  unsigned value          */
long    lval;   /*  long value                   */
double  dval;   /*  double value                 */
```

The object returned is simply the statement object passed. Any attempt to use a non-existing field will cause an error in which the application will be terminated. If you assign a value of the wrong type, the system will convert it for you. Date fields are simply converted to `long`'s of the form `YYYYMMDD`.

1.3.4 Adding The Record

Once the data has been assigned the record may be added as follows.

`AddRecord::Statement` [AddRecord]

```
r = gAddRecord(stmt);

object  stmt;   /*  the statement object  */
int     r;      /*  result                */
```

This will cause the data to be added to the table. Zero is returned upon success, other values indicate an error. Most errors are trapped and cause the application to abort, however.

1.4 Reading Records

1.4.1 Overview

Reading records is accomplished in three steps as follows.

1. Select a set of records from which to access data.
2. Select a particular record.
3. Access the field data.

1.4.2 Selecting a Record Set

Selecting a record set is accomplished by executing a normal SQL `select` command as follows.

`DBSelect::Statement` [DBSelect]

```
r = gDBSelect(stmt, cmd);

object  stmt;   /*  the statement object    */
char    *cmd;   /*  the SQL select command  */
int     r;      /*  result                  */
```

This method will execute an arbitrary SQL select command given in `cmd`. This command may contain joins, order by, and where clauses as well as any other legal SQL select command. zero is returned upon success. For example:

```
gDBSelect(stmt, "select * from myfile");
```

`DBSelectOne::Statement` [DBSelectOne]

```
r = gDBSelectOne(stmt, cmd);

object  stmt;   /*  the statement object    */
char    *cmd;   /*  the SQL select command  */
int     r;      /*  result                  */
```

This method is a convenience for the case when a single specific record is desired. When this method is used there is no need to also select a specific record. Data fields may be accessed immediately. zero is returned upon success.

1.4.3 Selecting a Record

The following functions allow the selection of a specific record within a selected set. Note that if `gDBSelectOne` is used, none of these functions should be used.

```
r = gFirstRecord(stmt);

r = gLastRecord(stmt);

r = gNextRecord(stmt);

r = gPrevRecord(stmt);

r = gSetAbsPos(stmt, pos);

r = gSetRelPos(stmt, pos);

object  stmt;   /*  the statement object           */
long    pos;    /*  relative or absolute position  */
int     r;      /*  0=success                      */
```

1.4.4 Accessing Field Data

The following functions are used to access field data.

```
sval = gFldGetString(stmt, fld);

cval = gFldGetChar(stmt, fld);

sval = gFldGetShort(stmt, fld);

uval = gFldGetUnsignedShort(stmt, fld);

lval = gFldGetLong(stmt, fld);

dval = gFldGetDouble(stmt, fld);

oval = gFldGetValue(stmt, fld);

object  stmt;   /*  the statement object  */
char    *fld;   /*  the field name        */
short   sval;   /*  return values         */
char    cval;
unsigned short uval;
long    lval;
double  dval;
object  oval;
```

If the field type is not the same as that requested, the system will convert it for you.

1.5 Changing and Deleting Records

The procedure for changing or deleting records is similar to that of reading records. Steps 1 and 2 are the same. In addition, once a record had been read it may be immediately changed or deleted without the need to re-execute the first two steps. Step 3 is as follows.

`UpdateRecord::Statement` [UpdateRecord]

```
r = gUpdateRecord(stmt);
        or
r = gDeleteRecord(stmt);

object  stmt;   /*  the statement object  */
int     r;      /*  result (0=success)    */
```

This method will either update or delete the last selected record.

1.6 Executing SQL Commands

The following methods allow the execution of arbitrary SQL commands.

`Execute` [Execute]

```
r = gExecute(DB, cmd);
     or
r = gExecute(stmt, cmd);

object  DB;     /*  database object    */
object  stmt;   /*  statement object   */
char    *cmd;   /*  SQL command        */
int     r;      /*  result (0=success) */
```

These methods allow for the execution of arbitrary SQL command when no specific data access is required (you aren't trying to access field values).

`ExecuteFile` [ExecuteFile]

```
r = gExecuteFile(DB, file);
       or
r = gExecuteFile(stmt, file);

object  DB;     /*  database object    */
object  stmt;   /*  statement object   */
char    *file;  /*  file name          */
int     r;      /*  result (0=success) */
```

The method will cause all the SQL commands in a file to be executed against database `DB`. A file extension of `.sql` is assumed.

1.7 Dropping Tables

`DropTable::Database` [DropTable]

```
r = gDropTable(DB, tbl);

object  DB;     /*  database object    */
char    *tbl ;  /*  table              */
int     r;      /*  result (0=success) */
```

The method will drop table `tbl`. If the table didn't already exist this command will be ignored.

1.8 Clearing Fields

`Clear::Statement` [Clear]

```
r = gClear(stmt);

object  stmt;   /*  statement object   */
int     r;      /*  result (0=success) */
```

This method will clear (zero out) all the fields. It may only be used after a select command. It is often used after some records have been read and a new record is to be added without an intervening record set selection.

1.9 WDS Interface

The following methods are used to associate ODBC data fields to dialog controls.

`AttachDialog::Statement` [AttachDialog]

```
stmt = gAttachDialog(stmt, dlg);

object  stmt;   /*  statement object   */
object  dlg;    /*  a dialog object    */
```

This method is used to associate all controls within a dialog to fields associated with a statement in which the fields and controls have the same name. Once this association is formed subsequent use of the dialog will cause the data to be entered into the associated data fields. This method should only be used subsequent to a `gDBSelect`, `gDBSelectOne`, or a `gInsert`.

`AssociateCtl::Statement` [AssociateCtl]

```
stmt = gAssociateCtl(stmt, fld, ctl);

object  stmt;   /*  statement object   */
char    *fld;   /*  a database field   */
object  ctl;    /*  a control object   */
```

This method is used to associate control `ctl` with database field `fld`. Once this association is formed subsequent use of the control will cause the data to be entered into the associated data field. This method should only be used subsequent to a `gDBSelect`, `gDBSelectOne`, or a `gInsert`.

Method Index

Table of Contents