Accessing Files and Databases with ODBC Facilities
W32 App Builder Version
Programmer's Guide

I. Introduction

The Open Database Connectivity facility or ODBC is a standard derived to permit different applications, and different programming languages, to access foreign databases. ODBC is a design method for creating drivers to access databases, and a specific set of SQL commands that can be used to perform database access.

Microsoft ships various ODBC drivers free of charge with various of their products, including operating systems (such as Windows 95 and Windows NT), and applications (such as MS Word and MS Excel). CET W/32 programs can use the ODBC features of W/32 BASIC to open, read, write, update and close foreign databases.

II. Using ODBC Features

To use ODBC with a CET W/32 program, you must have the correct ODBC driver installed on your system. ODBC drivers are available from a number of sources. Often, they are included with the product (e.g. the ODBC driver for Microsoft Access is on the Access release disks).

Once you have the correct ODBC drivers installed and a data source configured (according to the documentation for the driver), you can access that data source with a CET W/32 program.

The standard OPEN, CLOSE, MAT READ, WRITE, MAT WRITE, MAT READNEXT and MAT READPREV statements are used to access an ODBC data source. All values written to or read from the data source are strings. (W/32 will convert all string values to/from the native data source type automatically.)

To open an ODBC data source, use the OPEN statement with the INDEXED option, in either UPDATE or INPUT mode. The filename argument determines which ODBC data source is opened and may be indicated in one of two formats:

Format 1: ODBC*xx;*

The first format brings up a dialog box so the user may select from any of the currently configured data sources. The *xx* in ODBC*xx* is either **SS** for a snapshot or **DS** for a Dynaset. A snapshot contains records that are valid when selected the first time, while a dynaset maintains consistency with the actual data source by checking if the record is still current and getting a new record if it is not. A snapshot is quicker to use as it does not have to constantly re-query the data source.

Format 2: ODBC*xx*;DSN=<*data souce name*>;"

The second format opens a specific data source name that was set when you configure the data source through the ODBC administrator. For example, the following statement would open a data source named "Test Data" for reading and writing on channel #5 in snapshot mode:

```
OPEN #5:"ODBCSS;DSN=Test Data;", UPDATE INDEXED
```

Once the OPEN statement establishes a connection to the data source, the key field in the READ statement is used to obtain specific information about the data source and access the records. The key may be set to one of the following reserved words, or it can be an SQL SELECT statement.

TABLES Returns all table names of type "table" that are found in the data source. Two values are returned for each table - name and type.

ALLTABLES Returns all table names, including system tables. Two values are returned for each table - name and type.

TYPES Returns information about the data types supported. The nine values returned are:

1. The native data base type
2. The associated SQL type
3. The column precision
4. Whether or not the column is case sensitive
5. Whether or not the column is searchable
6. Whether or not the column is an auto-increment column
7. The column prefix
8. The column suffix
9. The CREATE PARMS for the column

The information returned by TYPES is only required for advanced operations on the data source, such as adding new tables or columns.

COLUMNS:*table* Returns all of the column names in the specified table. Three values are returned for each column: the name, field precision, and field type.

REQUERY Re-queries the data source to make sure all records are current. No information is returned to the program.

An SQL SELECT statement may also be used to select a group of records based on special criteria. Once a group of records has been selected, the individual records are read using READNEXT or READPREV. If no records are retrieved, the READ statement will return an EOF condition.

The SQL SELECT statement has many options. (Refer to an SQL document for more information.) The records retrieved from the data source and the number of fields returned are based on the syntax used. The simplest form of a SELECT statement is:

SELECT <*column names*> FROM <*tables*>

where

column names Is either a comma separated list of column names or "*" to select all columns.

tables Is the name of the table to be accessed.

For example, the following SELECT statement as the key field for READ may be used to retrieve the two columns "Name" and "Comments" from a table named "Authors":

READ #5,"SELECT Name,Comments FROM Authors": NUMCOLS%

A READ where the key is a SELECT statement returns the number of selected columns. The actual records could be accessed as follows. Notice that the key field for READNEXT and READPREV is not used, so it is set to an empty string.

```
dim result$(2)
while 1
   mat readnext #5,"": result$
   if eof(5) then goto done
   print result$(1), result$(2)
   wend
done:
```

The WRITE statement is used to add, update or delete records. If the WRITE operation fails, an EOF condition will be generated.

To delete or update a record, first use READNEXT/READPREV statements to position the pointer at the correct record in the record set. The key field for the WRITE statement can have three values: "UPDATE", "DELETE" or "ADDNEW".

To update the Comment column for Author "Joe Smith" with “a new comment” perform the following:

```
dim result$(2)
while 1
   mat readnext #5,"": result$
```

```
   if eof(5) then goto notfound
   if result$(1) = "Joe Smith"
      result$(2) = "a new comment"
      mat write #5,"update": result$
      goto done
      ifend
   wend
notfound:
   print "Couldn't find Joe Smith"
done:
```

To delete the entire record for "Joe Smith":

```
dim result$(2)
while 1
   mat readnext #5,"": result$
   if eof(5) then goto notfound
   if result$(1) = "Joe Smith"
      write #5,"delete": ””$
      goto done
      ifend
   wend
notfound:
   print "Couldn't find Joe Smith"
done:
```

Note that an empty string is required as the value to be written It is not used, but the compiler will generate an error if a WRITE statement is encountered with no values to write.

The following example shows how to use the TYPES reserved word. (It is assumed that the data source supports a maximum of 30 types.)

```
dim type$(30,9)
mat read #1, "TYPES": type$
print \ print "Data Types:" \ print
for I% = 1 to 30
  if type$(i%,1) = "" then goto endtype
  print type$(i%,1) + " (" + type$(i%,2) + ")" + " (" + type$(i%,3) + ")" + "
  (" + type$(i%,4) + ")" + " (" + type$(i%,5) + ")"
  print " (" + type$(i%,6) + ")" + " (" + type$(i%,7) + ")" + " (" +
  type$(i%,8) + ")" + " (" + type$(i%,9) + ")"
  wait
next i%
endtype:
```

Note that the environment variable B_ODBCERR may be set to one in your **w32app.w32** file to display messages for any ODBC errors that may occur.

The following sample program shows how to browse a data source, select the table and columns to be retrieved, and display the selected data. Another sample called **odbcdemo** has been included with the W/32 Application Builder in the subdirectory **\cetsamp\odbc**. This program uses a special form designed to illustrate exactly how the ODBC features work.

```
REM Allow up to 20 table and column names.
      dim tables$(20,2), col$(20,3), rslt$(20)
REM Open a snapshot for input only.  Choose the data source at run-time.
      open #1: "ODBCSS;", input indexed
tables:
      REM Get the tables defined for this data source.
         mat read #1, "TABLES": tables$
      REM Check for EOF.  (EOF on TABLES is an error.)
         if eof(1)
            msgbox "No tables defined"
            quit
            ifend
      REM Display the tables in this data source.
```

```
      print \ print "Available Tables:" \ print
      for I% = 1 to 20
         if tables$(i%,1) = "" then BREAK
         print TAB(8);tables$(i%,1) + " (" + tables$(i%,2) + ")"
         next i%
   REM Let the user choose which table to browse.
      print \ print \ print "Select a Table name (or Q to quit):";
      linput tname$
      if tname$ = "Q" then quit
      mat read #1, "COLUMNS:"+tname$: col$
   REM Check for EOF.  (EOF on columns is an error.)
      if eof(1)
         msgbox "No columns in the selected table"
         goto tables
         ifend
   REM Display the columns in this table.
      print \ print \ print "Available Column names:" \ print
      for I% = 1 to 20
         if col$(i%,1) = "" then BREAK
         print TAB(8);col$(i%,1) + " (" + col$(i%,2) + ")" + " (" +
col$(i%,3) + ")"
         next i%
   REM Let the user choose which columns to retrieve.
      print \ print \ print "Select Column names (comma seperated) for
retrieval (or * for all):";
      linput cname$
      mat read #1, "SELECT "+cname$+" FROM "+tname$: a$
   REM Check for EOF.  (EOF on SELECT means no records were selected.)
      if eof(1)
         msgbox "EOF on SELECT"
         goto tables
         ifend
   REM Display the selected columns from the record set.
      while 1
         print \ print
         mat readnext #1, key$: rslt$
         if eof(1) then BREAK
         for I% = 1 to 10
            if rslt$(i%) = "" then BREAK
            print TAB(8);
            print rslt$(i%)
            next i%
         wait
         wend
      print "End of Record Set Reached"
      wait
   REM Select another table.
      print CRT$("CLEAR");
      goto tables
```

IV. New W/32 ODBC Features

Since its introduction in Version 8.13 of W/32 App Builder, CET has introduced a number of enhancements to the ODBC facility.

The maximum length of an SQL SELECT statement is now 2048 characters. The previous limit was 253.

A new keyword "RESULTCOLUMNS" has been added for READ. After a successful SELECT statement has been executed, this keyword may be used to retrieve a description of the columns that were selected. There are three values per column; the column name, precision and type. Only use RESULTCOLUMNS. For example:

```
READ #1, "SELECT * FROM section": NUMCOLS%
```

```
IF EOF(1) THEN ...   \ REM Always check for EOF on a SELECT
WHILE NOT EOF(1)
   DIM col$( NUMCOLS%, 3)
   MAT READ #1,"RESULTCOLUMNS": col$ ...
```

After any SELECT statement, you are positioned at the beginning of the record set. Four new keywords have been added to READ to allow you to move through the records. They are:

MOVEFIRST Moves prior to the first record. (READNEXT returns the first record.)

MOVELAST Moves after the last record. (READPREV returns the last record.)

MOVE:num Moves relative to the current record. Negative values move back. Positive values move forward. Zero does nothing. Returns EOF if moving backward and you have reached the beginning or if moving forward and you are at the end.

STATUS Returns the zero-based index of the current record. The first record will be record 0.If at EOF, then -1 is returned. If the current index cannot be determined, then -2 is returned.

Note that if you do a MOVELAST and then a READPREV or MOVE:-1, the last record will be current, but STATUS will return -2 because ODBC does not yet know the index of the last record.

The total number of records can not be determined without scrolling through all of the records.If you are performing READNEXT operations from the beginning, then STATUS can determine the current record number. For example:

```
READ #1,"any SELECT statement": numcols$
IF EOF(1) THEN ..... \ REM Always check for EOF on a SELECT
WHILE NOT EOF(1)
   READ #1,"MOVE:1":dummy$
WEND
READ #1,"STATUS": lastrec%
```