Using the ODBC Function Library

1. Background

The AMPS language contains a set of functions which allow QEI Exec to exchange data with ODBC data sources. These functions are documented in the help system, and you can obviously use them directly to write macros. They operate at quite a low level however, so PCF has written a standard function library to simplify the development of macros requiring ODBC. The library is shipped as a standard macro named ODBC.SRC with QEI Exec version 3.

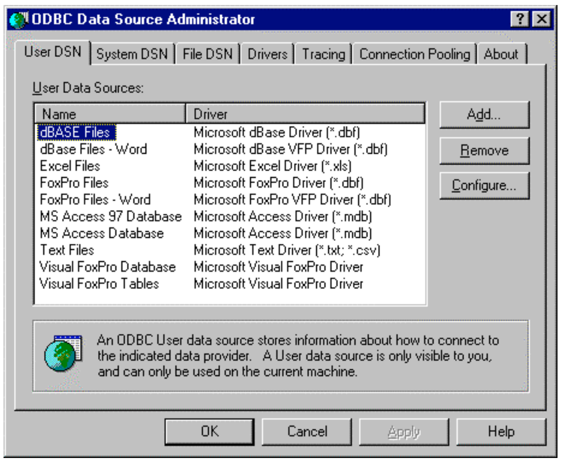
The document provides a number of example macros demonstrating techniques for using the ODBC functions, and it is assumed that the reader is reasonably familiar with both the QEI Command Language and AMPS. It is also not intended as an SQL teaching tool, although the examples should provide enough information to allow users without detailed knowledge of SQL to create suitable macros in most cases.

2. ODBC Basics

The ODBC (Open DataBase Connectivity) standard was originally developed by Microsoft to allow applications to connect in a consistent fashion with sources of data such as spreadsheets or databases. It has been widely adopted by many software vendors as a means of delivering program based access to information held within their applications.

An ODBC session consists of two components. These are the application storing the data (the Data Source) and the application reading or writing the data (the Client). At the current time, QEI Exec is only capable of operating as an ODBC client; that is, it can read and write data via ODBC. It is not capable of acting as a Data Source to which other applications can connect, although this may well change in future releases.

To set up a Data Source, you need to go the Control Panel on your PC and click on the ODBC icon. If you have more than one ODBC icon, click the one labelled "32 bit". You should see a property sheet similar to the one shown below – the list of available data sources will depend on which applications are installed on your machine.



Name	Driver
dBASE Files	Microsoft dBase Driver (*.dbf)
dBase Files - Word	Microsoft dBase VFP Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
FoxPro Files	Microsoft FoxPro Driver (*.dbf)
FoxPro Files - Word	Microsoft FoxPro VFP Driver (*.dbf)
MS Access 97 Database	Microsoft Access Driver (*.mdb)
MS Access Database	Microsoft Access Driver (*.mdb)
Text Files	Microsoft Text Driver (*.txt; *.csv)
Visual FoxPro Database	Microsoft Visual FoxPro Driver
Visual FoxPro Tables	Microsoft Visual FoxPro Driver

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

To create a new Data Source, select the User DSN, System DSN or File DSN tab, select the appropriate data source from the list, click the Add button and go through the dialogs that appear, which will be different for each type of driver. The fundamental thing to realise is that a data source has a name associated with it – the Data Source Name (DSN) – and also perhaps a User ID and Password. These are often required if your Data Source is an enterprise class database such as SQL Server or Oracle.

The final point to bear in mind regarding ODBC is that all exchanges of data with the Data Source use the SQL database query language. This is true regardless of the actual way in which the application storing the data manages it internally – for instance, Excel does not support SQL if you use it in the normal fashion, but the ODBC driver for Excel has to map SQL queries from ODBC clients to the appropriate internal behaviour.

3. The ODBC Library

The ODBC library consists of a number of routines that carry out the operations required in typical ODBC sessions. These include:

- Connecting to a Data Source
- Defining Queries
- Executing Queries
- Disconnecting from a Data Source
- Managing Errors

All these routines can be called from other macros using the AMPS `run` command with a number of arguments, as shown below:

```
run "WINMAC:odbc.mac", "Connect", 0, hDB%, dsn$, uid$, pwd$, status%, msg$
```

The first argument is a string that specifies what function is being requested. The remaining arguments are used to exchange data with the routine being called; their exact meaning depends on the routine itself. The table below lists the routines in the library, the arguments required and the purpose of the routine. An asterisk next to an argument indicates that the routine uses it to return data to the calling macro.

MODE	**QID**	**HDB**	**S1**	**S2**	**S3**	**ERR**	**MSG**	**FUNCTION**
Init	n/a	n/a	n/a	n/a	n/a	err*	msg*	initialise query array
Connect	n/a	hdb*	DSN	UID	PWD	err*	msg*	connect to database
BuildQuery	qid*	hdb	TBL	COL	SQS	err*	msg*	create query in one go
InitQuery	qid*	hdb	TBL	COL	SQS	err*	msg*	create part 1 of query
ExtendQuery	qid	n/a	n/a	n/a	SQS	err*	msg*	create part n of query
CloseQuery	qid	n/a	n/a	n/a	n/a	err*	msg*	finish off query build
RunQuery	qid	n/a	FIL	QUO	SEP	err*	msg*	(re)run a query
GetRowData	qid	n/a	RW*	QUO	SEP	err*	msg*	get row of data
StopRowData	qid	n/a	n/a	n/a	n/a	err*	msg*	stop getting row data
DeleteQuery	qid	n/a	n/a	n/a	n/a	err*	msg*	delete a query
Disconnect	n/a	hdb	n/a	n/a	n/a	err*	msg*	disconnect from database

The following list describes in more detail the meaning of the various arguments as they apply to the routines listed above:

qid	Query identifier. An integer representing a query, returned by BuildQuery or InitQuery, which is used to subsequently run or delete the query.
hdb	Database connection handle. An integer representing a connection to an ODBC data source (DSN) which is returned by Connect. It is then used to identify the database in BuildQuery, InitQuery and Disconnect.
DSN	The Data Source Name associated with an ODBC data source.
UID	The user id associated with the ODBC data source specified by DSN
PWD	The password associated with the user id UID.
TBL	The name of the database table referenced by the query.
COL	The list of result columns associated with the query.
SQS	The SQL query statement. This is the string used to create the query. It can be split into several parts if the overall length is more than 255 characters.
FIL	The name of a file that will contain the input data or results associated with a query.
RW	A string variable that will contain the next row of data to be returned by the query.
QUO	A character identifying the quote character in the file.
SEP	A character identifying the field delimiter in the file.

Please also note the following points:

- There would normally never be any need to call the Init routine from a macro – this routine is called once during system startup (SYSINIT.SRC) to initialise some data structures and should not be called again.
- The final two arguments are used to return status information to the calling macro. The first is an integer argument which will be set to 1 if any problems occurred; the second is a string which will contain a descriptive message relating to the error.
- The low level AMPS functions permit up to 512 simultaneous database connections, each with an effectively unlimited number of associated queries. A query is therefore uniquely identified by the combination of the database connection handle and the query handle. Given that this capability is likely to far exceed the needs of most users and is a little complex, the ODBC library routines store the database and query handles in an array and all reference to queries is simply by the array index (the qid parameter). The maximum number of concurrent queries available is therefore limited to the size of the array; this is set to 10 in the current version of QEI Exec.
- There are two ways of constructing a query. The simplest way is to use the BuildQuery routine. An alternative method, which is necessary when the SQL statement exceeds 256 characters in length and cannot be stored in a single AMPS variable, is to use InitQuery, followed by as many calls to ExtendQuery as are necessary to accommodate the statement, then CloseQuery.

4. A Simple ODBC Session

Let us now look at how one might use the ODBC functions by developing a simple macro that updates the descriptions on a number of activities within a QEI Exec database. This will allow us to develop a basic framework for an ODBC session that will be further expanded in later examples.

Suppose a relational database has been set up with a DSN of PROJDATA, and that no user or password is required for access. The database contains a table called TASKS which has several columns. One column called TID holds the Task ID which is equivalent to the Activity Name in QEI Exec, and another called OPERATION is mapped to the Description. We will assume that the macro is run with the current database holding AIs corresponding to the tasks in the external database.

The first step is therefore to connect to the external database:

```
rem establish database connection
  dsn$ = "PROJDATA"
  uid$ = ""
  pwd$ = ""
  hDB% = 0
  qry% = 0
  run "WINMAC:odbc.mac", "Connect", 0, hDB%, dsn$, uid$, pwd$, status%, msg$
  if status%
    gosub ODBCError:
  endif
```

The routine ODBCError handles problems cleanly – we will examine its contents later. Now we have a connection to the database, we can create our query and retrieve the information we want. The SQL statement we want to execute is

SELECT TID, OPERATION FROM TASKS ORDER BY TID

Which will generate a result set containing the Name and Description. In this macro, we will write this data out to a temporary file and then use the interface functions in QEI Exec to perform the update. To create and execute the query we write:

```
rem create temporary file name using expanded logical
  file$ = f_list$("QEIWRK:\.", 2, 0)
  file$ = substr$(file$, 1, length%(file$)-1)
  file$ = file$ + "q" + getsys$(2) + ".tmp"
rem define quote and delimiter characters
  quo$ = """"
  sep$ = ","
rem build query
  tbl$ = "TASKS"
  col$ = "TID, OPERATION"
  sqs$ = "SELECT " + col$ + " FROM " + tbl$ + " ORDER BY TID"
```

```
run "WINMAC:odbc.mac", "BuildQuery", qry%, hDB%, tbl$, col$, sqs$, status%, msg$
  if status%
    gosub ODBCError:
  endif
rem execute query
  run "WINMAC:odbc.mac", "RunQuery", qry%, 0, file$, quo$, sep$, status%, msg$
  if status%
    gosub ODBCError:
  endif
```

We now have a file containing the data we require; the next step is simply to define a suitable interface format and update the QEI Exec database:

```
rem define file format
  send "interface"
  send "  format custom"
  send "    type variable delimited with ’,’ quoted with ’""’"
  send "    acti"
  send "      actnam character 20"
  send "      desc   character 50"
  send "    return"
  send "  return"
rem perform update
  send "  update act ’" + file$ + "’"
  send "return"
```

Finally, we need to tidy up by deleting our temporary file and disconnecting from the database. The ODBC Library will automatically delete any queries associated with the database so we don’t need to do this explicitly:

```
rem delete data file
  i% = rm%(file$)
rem disconnect from database and stop
  run "WINMAC:odbc.mac", "Disconnect", 0, hDB%, "", "", "", status%, msg$
  end
```

Finally, we need to examine the ODBCError routine. For the moment any error will simply cause the macro to terminate after disconnecting from the database:

```
ODBCError:
  rem manage fatal errors cleanly
  run "WINMAC:error.mac", "ODBC Error", msg$
  run "WINMAC:odbc.mac", "Disconnect", 0, hDB%, "", "", "", status%, msg$
return
```

The full text of this macro is shown as Listing 1 at the end of this document.

5. An Alternative Strategy

The macro we have just developed makes use of an intermediate text file to hold the result set. This has the advantage that the update can then be carried out at high speed using the low level data interfacing functions within QEI Exec, but may not always be a suitable approach, particularly if the updating of each element in the QEI Exec database requires a separate query. We can modify our macro to demonstrate an alternative technique that retrieves data one row at a time, as shown in the pseudocode below:

```
Build table of AIs in the database
For each AI:
  Get name
  Query via ODBC to get OPERATION for TID with matching name
  Set description to data retrieved
```

Assuming we have first connected to the database as in the example above, we then need some code that builds a table of all the AIs and loops through them until no more are left. This is a very standard AMPS construction:

```
  send "table create name r"
  send "table add ai"
  errset 0
  send "tabf"
  while not error%(2) do
    gosub Update:
    errset 0
    send "tabn"
  enddo
```

We have also isolated the query processing to a separate subroutine, which makes it easier to read the code. Let us now look at the subroutine itself, which can be described in pseudocode like this:

```
Get AI name
Build query to get OPERATION for TID with matching name
Run query
Set AI description to data retrieved
Delete query
```

Here is the actual AMPS code:

```
Update:
  send "q /name$ name"
  rem build query
  col$ = "OPERATION"
  tbl$ = "TASKS"
  sqs$ = "SELECT " + col$ + " FROM " + tbl$ + " WHERE TID=’" + name$ + "’"
  qry% = 0
  run "WINMAC:odbc.mac", "BuildQuery", qry%, hDB%, tbl$, col$, sqs$, status%, msg$
  if status%
    gosub ODBCError:
  endif
  rem execute query
  run "WINMAC:odbc.mac", "GetRowData", qry%, 0, desc$, quo$, sep$, status%, msg$
  if status%
    gosub ODBCError:
  endif
  rem reformat and update
  desc$ = quote$(desc$, "")
  send "desc " + quote$(desc$, "’")
  rem delete query
  run "WINMAC:odbc.mac", "DeleteQuery", qry%, 0, "", "", "", status%, msg$
  if status%
    gosub ODBCError:
  endif
return
```

Note that we have assumed there will only be one row in the result set – we are reading the first one returned by the GetRowData routine and discarding any others. Also note that the row returned only contains one column, which will be a string quoted with double quotes. These have to be stripped off before the description field is updated. If the returned row had contained several columns, the parsing and reformatting required might have involved some quite complex code. Issues relating to data types, parsing and formatting are discussed in more detail in Section 8.

If the result set had contained several rows, we would have needed to call the GetRowData routine repeatedly until it returned an empty string to retrieve them all. If we had only needed some of them, the StopRowData routine can be used to clear the result set so that a subsequent GetRowData call will retrieve the first row of the result set again.

The full source for this macro is shown as Listing 2 at the end of this document. Some variable declarations are in slightly different places but the algorithm is the same.

6. Writing Data from QEI Exec

We have so far only used SELECT statements in macros, to extract data from a remote application and update the QEI Exec database. It is perfectly possible to use INSERT or UPDATE statements to write data from QEI Exec into the other application, and once again (for the INSERT statement) we have the option of using an external file to perform a bulk transfer operation.

Looking at the UPDATE statement first, let us consider writing out the description and duration back into the relational database for each AI in the QEI Exec database. The duration is held in the relational database in a column of type Integer, while the description is in a Character column.

Looking at the macro in Listing 2, the only change we need to make is to the Update routine, which we will rename to WriteData. The pseudocode for the routine is:

```
Get AI name, description and duration
Build query to update DAYS and OPERATION for TID with matching name
Run query
Delete query
```

The source code looks like this:

```
WriteData:
  send "q /name$ name"
  send "q /desc$ desc"
  send "q /dura% dura"
  tbl$ = "TASKS"
  col$ = "DAYS, OPERATION"
  rem build query
  sqs$ = "UPDATE " + tbl$ + " SET DAYS=" + dura% + ", OPERATION=" + quote$(desc$, "’")
  sqs$ = sqs$ + " WHERE TID=" + quote$(name$, "’")
  run "WINMAC:odbc.mac", "BuildQuery", qry%, hDB%, tbl$, col$, sqs$, status%, msg$
  if status%
    gosub ODBCError:
  endif
  rem execute query
  run "WINMAC:odbc.mac", "RunQuery", qry%, 0, "", "", "", status%, msg$
  if status%
    gosub ODBCError:
  endif
  rem delete query
  run "WINMAC:odbc.mac", "DeleteQuery", qry%, 0, "", "", "", status%, msg$
  if status%
    gosub ODBCError:
  endif
return
```

Note that you have to specify the list of columns which are being updated, even though you don’t embed the list in the statement. You can also see that the code has to put single quotes round fields defined as Character data in the relational database in order to generate syntactically correct statements. The full source code for this macro is available as Listing 3 at the end of the document.

Now let us consider the INSERT statement, which is used to add new rows to a table. This comes in two forms, as shown in the examples below:

```
INSERT INTO TABLE1 (COL1, COL2) VALUES (’VAL1’, VAL2)
INSERT INTO TABLE1 (COL1, COL2) VALUES (?, ?)
```

The first form specifies the data values explicitly in the statement, and so we could write another macro similar to the last one which uses this statement to append new rows to the TASKS table. The second form assumes that the data values to be inserted are held in a text file and have been formatted with the appropriate quote and delimiter characters. Let’s use this form of the statement to append all AIs with their CODE attribute set to "NEW" to the TASKS table - the QEI commands for building a table containing only these AIs are:

```
send "table create name r"
send "table add ai with code eq ’NEW’"
```

We then need to create the file holding the data. We could do this via the QEI Exec interface, using the TABLE modifier in the ACTIVITY command to export data only for those AIs in the table, or we can use the AMPS file management functions to create the file. As we used the interface in the first example, we’ll use AMPS this time – the pseudocode for the heart of the process is then:

```
Build table of AIs with CODE = "NEW"
Open file for output
For each AI:
  Get name, description and duration
  Write data to file in correct format
Close file
Build query to update table from file
Run query
Delete file
```

The full source for this macro is shown as Listing 4 at the end of this document. Look how the macro has to explicitly format the fields in the data file to match the declared type of each column in the relational database.

7. More on SQL Statements

This section is a roundup of various SQL related topics which have not been covered by the previous Sections.

Column List	The list of columns passed to the ODBC routines can be space delimited if desired, although you cannot then reuse it as part of the statement itself as it would generate a syntax error. For SELECT statements you can also use the standard shorthand notation "*" to specify all columns.
Transactions	Each query forms an atomic transaction which will only be committed to the database when it has successfully completed. You can create compound statements by using the InitQuery and ExtendQuery routines, terminating each statement with the ";" delimiter.
Other Statements	It is possible to use the ODBC Library to create and delete tables (if you have the appropriate permissions) via the CREATE TABLE and DROP TABLE statements. Aliases (eg SELECT COL1 AS C1) are supported as long as the real columns are specified in the Column List. The aliases are only of use within the query as the output file contains no column headers. Subqueries are supported as they do not affect the final set of result columns. In general, provided that the driver accepts the syntax, SQL statements not covered explicitly (eg SELECT INTO) will work, but see the next topic for known restrictions.
Restrictions	Aggregate functions such as COUNT(*) and AVG(*) are not yet supported. You cannot yet define a result set based on columns from a number of tables (eg after a JOIN operation). It is best to match the case of table and column definitions in your macro to those in the database as some drivers are case sensitive. It is not a good idea to try to work with tables or columns whose names contain spaces (supported by some applications) – you need to rename them or use CREATE VIEW to create a view in which they appear with sensible names.

8. Data Types, Parsing and Formatting

The biggest problem with using ODBC is matching up the format of the data between QEI Exec and that used by the other application. As we have seen in the examples so far, the ODBC Library routines will return character data quoted with a character of your choice, and numeric data will not be quoted. The fields within the data will be delimited by another character of your choice. Even with these options, the data within the fields may not correspond to the format expected by QEI Exec and so cannot be transferred via the interface. If this is the case, then you have to write an AMPS macro that is capable of correctly parsing the lines of data produced by the query.

In general, text and numeric fields are not a problem – that is reserved for date fields. This is because the exact representation of the date (and time) are not only application specific but may also depend on the current locale set in the operating system on your machine. That is to say, not only may it vary depending on whether your ODBC data source is Access or Oracle, but it may also depend on whether your PC is set up for the UK or the US. Furthermore, the representation used by date fields retrieved via ODBC may not work when you wish to insert or update them!

For example, using Access 2000 on a PC set up for the UK locale, a date field is returned as a string dd/mm/yyyyhhmm, without quote characters. If one tries to use this date format to update an Access database via it rapidly becomes clear that even though Access accepts UK dates correctly when viewed on screen, it always assumes dates are in mm/dd/yyyy format (ie US format) when entered using an INSERT or UPDATE.

There are two ways around this: The first (explained in the Access help system) is to make use of an Access-specific function called DateValue which will accept dates formatted correctly for the current locale. So, to update a date field in Access via ODBC with a UK formatted date you could create a statement along the lines of

```
UPDATE PROJDATA SET START_DATE=DateValue('dd/mm/yyyy') WHERE ACTNAME='TASK01'
```

This statement will work as required, but will stop working if you the database was ported to (say) SQL Server as the DateValue function is specific to Access. The second solution gets around this problem by using what is called an "ODBC escape clause" for dates – the statement would then be written

UPDATE PROJDATA SET START_DATE={d 'yyyy-mm-dd'} WHERE ACTNAME='TASK01'

These escape clauses are guaranteed to work across databases, so the query above will work if the database is ported from Access.

The ODBC escape clauses for date, time and timestamp data are:

`{d 'value'}`	where `value` is a date in *yyyy-mm-dd* format
`{t 'value'}`	where `value` is a time in *hh:mm:ss* format
`{ts 'value'}`	where `value` is a timestamp in *yyyy-mm-dd hh:mm:ss[.f…]* format

What this means is that you may need to write some date conversion routines to reformat date fields if you are unable to transfer data via the QEI interface, which offers reasonably sophisticated facilities for reading and writing dates and times. If this is the case, you may also need some code that can parse a line containing multiple delimited fields, some of which may be surrounded by quote characters and contain quote characters themselves (represented as pairs of quote characters). You may find the code in Listing 5 useful – this is a subroutine which will parse an input line and return a specific field from it. It assumes that fields are quoted with double quotes, the delimiter character is held in a variable named `sep$`, and the line of data in `row$`. If you call the routine having set the variable `fn%` to the number of the field you want (first field is field 0) then it will return the contents of the field in `f$`.

9. Debugging macros which use ODBC

The most common problem with macros using the ODBC Library is that the syntax of the statement being passed is incorrect. It is possible to check out exactly what is being passed to the routines in the ODBC Library by setting the macro into debug mode. This can be done by creating a registry key

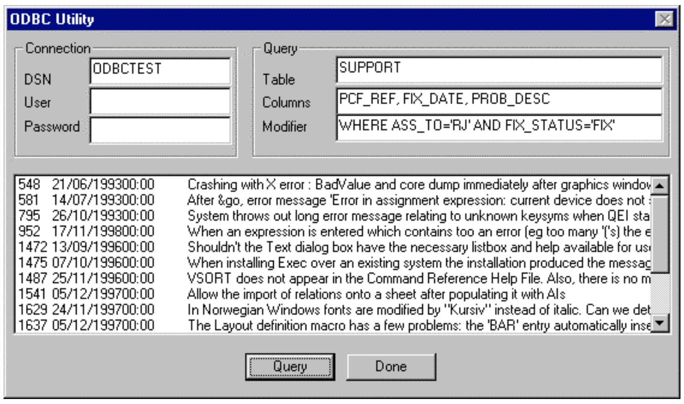
HKEY_LOCAL_MACHINE\SOFTWARE\PCF\QEI Exec\<version>\DEBUG

where <version> is the version of QEI Exec in use (currently 2.3 or 3), and creating within it a string value named ODBC which is set to "1". If you then open the QEI command window (F4) and click once on the Dismiss button to allow the GUI to operate normally then you will see trace information appear each time any of the ODBC Library routines are called - this includes the statements themselves. You should also double check the exact spelling and case of all the table and column names used by the statement – missing or incorrect values can have devastating consequences on the stability of the macro!

Finally, you can use an interactive SQL query tool (supplied with all relational databases, and many available for free download off the Internet) to test the exact syntax of statements before embedding them in macro code. QEI Exec comes with a simple interactive tool in the form of the ODBCDEMO macro, which can can be associated with a button on a custom toolbar or run from the command line by typing

```
&run "WINMAC:odbcdemo.mac"
```

This will bring up the dialog shown below:



DSN: ODBCTEST
User:
Password:
Table: SUPPORT
Columns: PCF_REF, FIX_DATE, PROB_DESC
Modifier: WHERE ASS_TO='RJ' AND FIX_STATUS='FIX'

```
548  21/06/199300:00   Crashing with X error : BadValue and core dump immediately after graphics windov
581  14/07/199300:00   After &go, error message 'Error in assignment expression: current device does not :
795  26/10/199300:00   System throws out long error message relating to unknown keysyms when QEI sta
952  17/11/199800:00   When an expression is entered which contains too an error (eg too many '('s) the e
1472 13/09/199600:00   Shouldn't the Text dialog box have the necessary listbox and help available for us
1475 07/10/199600:00   When installing Exec over an existing system the installation produced the messag
1487 25/11/199600:00   VSORT does not appear in the Command Reference Help File. Also, there is no m
1541 05/12/199700:00   Allow the import of relations onto a sheet after populating it with AIs
1629 24/11/199700:00   In Norwegian Windows fonts are modified by "Kursiv" instead of italic. Can we det
1637 05/12/199700:00   The Layout definition macro has a few problems: the 'BAR' entry automatically inse
```

You enter the database connection parameters in the left pane, the query in the right (it only supports SELECT statements) and all being well the results are displayed in the large list box.

Revision 3, May 18th 2001
Richard Jebb

Listing 1

```
rem establish database connection
  dsn$ = "PROJDATA"
  uid$ = ""
  pwd$ = ""
  hDB% = 0
  qry% = 0
  run "WINMAC:odbc.mac", "Connect", 0, hDB%, dsn$, uid$, pwd$, status%, msg$
  if status%
    gosub ODBCError:
  endif

rem create temporary file name using expanded logical
  file$ = f_list$("QEIWRK:\.", 2, 0)
  file$ = substr$(file$, 1, length%(file$)-1)
  file$ = file$ + "q" + getsys$(2) + ".tmp"
rem define quote and delimiter characters
  quo$ = """"
  sep$ = ","
rem build query
  tbl$ = "TASKS"
  col$ = "TID, OPERATION"
  sqs$ = "SELECT " + col$ + " FROM " + tbl$ + " ORDER BY TID"
run "WINMAC:odbc.mac", "BuildQuery", qry%, hDB%, tbl$, col$, sqs$, status%, msg$
  if status%
    gosub ODBCError:
  endif
rem execute query
  run "WINMAC:odbc.mac", "RunQuery", qry%, 0, file$, quo$, sep$, status%, msg$
  if status%
    gosub ODBCError:
  endif

rem define file format
  send "interface"
  send "  format custom"
  send "    type variable delimited with ’,’ quoted with ’""’"
  send "    acti"
  send "      actnam character 20"
  send "      desc   character 50"
  send "    return"
  send "  return"
rem perform update
  send "  update act ’" + file$ + "’"
  send "return"

rem delete data file
  i% = rm%(file$)
rem disconnect from database and stop
  run "WINMAC:odbc.mac", "Disconnect", 0, hDB%, "", "", "", status%, msg$
  end



ODBCError:
  rem manage fatal errors cleanly
  run "WINMAC:error.mac", "ODBC Error", msg$
  run "WINMAC:odbc.mac", "Disconnect", 0, hDB%, "", "", "", status%, msg$
return
```

Listing 2

```
rem establish database connection
  dsn$ = "PROJDATA"
  uid$ = ""
  pwd$ = ""
  hDB% = 0
  qry% = 0
  run "WINMAC:odbc.mac", "Connect", 0, hDB%, dsn$, uid$, pwd$, status%, msg$
  if status%
    gosub ODBCError:
  endif

rem define key variables
  quo$ = """"
  sep$ = ","
  tbl$ = "TASKS"
  col$ = "OPERATION"

rem iterate over AIs
  send "table create name r"
  send "table add ai"
  errset 0
  send "tabf"
  while not error%(2) do
    gosub Update:
    errset 0
    send "tabn"
  enddo

rem disconnect from database and stop
  run "WINMAC:odbc.mac", "Disconnect", 0, hDB%, "", "", "", status%, msg$
  end


Update:
  send "q /name$ name"
  rem build query
  sqs$ = "SELECT " + col$ + " FROM " + tbl$ + " WHERE TID=’" + name$ + "’"
  run "WINMAC:odbc.mac", "BuildQuery", qry%, hDB%, tbl$, col$, sqs$, status%, msg$
  if status%
    gosub ODBCError:
  endif
  rem execute query
  run "WINMAC:odbc.mac", "GetRowData", qry%, 0, desc$, quo$, sep$, status%, msg$
  if status%
    gosub ODBCError:
  endif
  rem reformat and update
  desc$ = quote$(desc$, "")
  send "desc " + quote$(desc$, "’")
  rem delete query
  run "WINMAC:odbc.mac", "DeleteQuery", qry%, 0, "", "", "", status%, msg$
  if status%
    gosub ODBCError:
  endif
return


ODBCError:
  rem manage fatal errors cleanly
  run "WINMAC:error.mac", "ODBC Error", msg$
  run "WINMAC:odbc.mac", "Disconnect", 0, hDB%, "", "", "", status%, msg$
return
```

Listing 3

```
rem establish database connection
  dsn$ = "PROJDATA"
  uid$ = ""
  pwd$ = ""
  hDB% = 0
  qry% = 0
  run "WINMAC:odbc.mac", "Connect", 0, hDB%, dsn$, uid$, pwd$, status%, msg$
  if status%
    gosub ODBCError:
  endif

rem define key variables
  tbl$ = "TASKS"
  col$ = "DAYS, OPERATION"

rem iterate over AIs
  send "table create name r"
  send "table add ai"
  errset 0
  send "tabf"
  while not error%(2) do
    gosub WriteData:
    errset 0
    send "tabn"
  enddo

rem disconnect from database and stop
  run "WINMAC:odbc.mac", "Disconnect", 0, hDB%, "", "", "", status%, msg$
  end


WriteData:
  send "q /name$ name"
  send "q /desc$ desc"
  send "q /dura% dura"
  rem build query
  sqs$ = "UPDATE " + tbl$ + " DAYS=" + dura% + ", OPERATION=" + quote$(desc$, "’")
  sqs$ = sqs$ + " WHERE TID=" + quote$(name$, "’")
  run "WINMAC:odbc.mac", "BuildQuery", qry%, hDB%, tbl$, col$, sqs$, status%, msg$
  if status%
    gosub ODBCError:
  endif
  rem execute query
  run "WINMAC:odbc.mac", "RunQuery", qry%, 0, "", "", "", status%, msg$
  if status%
    gosub ODBCError:
  endif
  rem delete query
  run "WINMAC:odbc.mac", "DeleteQuery", qry%, 0, "", "", "", status%, msg$
  if status%
    gosub ODBCError:
  endif
return


ODBCError:
  rem manage fatal errors cleanly
  run "WINMAC:error.mac", "ODBC Error", msg$
  run "WINMAC:odbc.mac", "Disconnect", 0, hDB%, "", "", "", status%, msg$
return
```

Listing 4

```
rem establish database connection
  dsn$ = "PROJDATA"
  uid$ = ""
  pwd$ = ""
  hDB% = 0
  qry% = 0
  run "WINMAC:odbc.mac", "Connect", 0, hDB%, dsn$, uid$, pwd$, status%, msg$
  if status%
    gosub ODBCError:
  endif

rem define quote and delimiter characters
  quo$ = """"
  sep$ = ","
rem get temporary file name & open for writing
  file$ = "QEIWRK:q" + getsys$(2) + ".tmp"
  open o 1 file$

rem iterate over AIs
  send "table create name r"
  send "table add ai with code eq ’NEW’"
  errset 0
  send "tabf"
  while not error%(2) do
    send "q /name$ name"
    send "q /desc$ desc"
    send "q /dura% dura"
    write 1 quote$(name$, quo$) + sep$ + quote$(desc$, quo$) + sep$ + dura%
    errset 0
    send "tabn"
  enddo
  close 1

rem build query
  tbl$ = "TASKS"
  col$ = "TID, OPERATION, DAYS"
  sqs$ = "INSERT INTO " + tbl$ + " (" + col$ + ") VALUES (?, ?, ?)"
  run "WINMAC:odbc.mac", "BuildQuery", qry%, hDB%, tbl$, col$, sqs$, status%, msg$
  if status%
    gosub ODBCError:
  endif
rem execute query
  run "WINMAC:odbc.mac", "RunQuery", qry%, 0, file$, quo$, sep$, status%, msg$
  if status%
    gosub ODBCError:
  endif

rem delete data file
  i% = rm%(file$)
rem disconnect from database and stop
  run "WINMAC:odbc.mac", "Disconnect", 0, hDB%, "", "", "", status%, msg$
  end


ODBCError:
  rem manage fatal errors cleanly
  run "WINMAC:error.mac", "ODBC Error", msg$
  run "WINMAC:odbc.mac", "Disconnect", 0, hDB%, "", "", "", status%, msg$
return
```

Listing 5

```
GetField:
  rem extract field number fn% from current line of data row$
  rem into string variable f$, by repeatedly chopping first
  rem field off the line until we reach the one we want. Field
  rem numbering starts at zero.

  rem simple parsing rules:
  rem   if first field starts with a quote character
  rem     search forward for balancing quote character, allowing
  rem     for embedded doubled quote characters in the field
  rem   else
  rem     field runs up to just before next separator

  n$ = row$
  for ii% = 1 to fn% + 1
    if substr$(n$, 1, 1) = ’"’
      jj% = 2
      while 1 do
        f$ = substr$(n$, jj%, 2)
        if f$ = ’"’
          rem EOL - stop
          break
        elseif f$ = ’""’
          rem advance 2
          jj% = jj% + 2
        elseif grep%(f$, ’[^"][^"]’)
          rem advance 2
          jj% = jj% + 2
        elseif grep%(f$, ’[^"]"’)
          rem advance 1
          jj% = jj% + 1
        elseif grep%(f$, ’"[^"]’)
          rem stop
          break
        else
          rem error..
          print "parse error: " + f$
          break
        endif
      enddo
      f$ = substr$(n$, 1, jj%)
      n$ = substr$(n$, jj%+2, 512)
    else
      if index%(n$, sep$)
        f$ = substr$(n$, 1, index%(n$, sep$)-1)
        n$ = substr$(n$, index%(n$, sep$)+1, 512)
      else
        f$ = n$
        n$ = ""
      endif
    endif
  endfor
return
```