

filePro's ODBC objects

To provide low-level access to ODBC data sources, filePro now includes ODBC “objects”. These are called `ODBC_CONNECTION`, and `ODBC`.

The `ODBC_CONNECTION` object

The first part of creating an ODBC recordset is to establish a connection to an ODBC server. This is done by creating an `ODBC_CONNECTION` object:

```
handle = new ODBC_CONNECTION( [CONNECTION_STRING] )
```

where “`CONNECTION_STRING`” is the connection string for the server. If none is specified, or an empty value is provided, Windows will ask for the data source at runtime. Also, if insufficient information is provided in the connection string, such as not including the password for a password-protected dataset, Windows will ask for that information at runtime.

The return value is a handle to the new object. If the value is less than or equal to zero, the connection failed.

Once the connection has been established, this handle is used to create the ODBC recordset object.

Example connection strings:

```
Driver={Microsoft Access Driver (*.mdb)};DBQ=c:\mdb\filepro.mdb;  
FileDSN=CustomerFileDSN;
```

Note that multiple recordsets can be created on a single connection.

The `ODBC` object

The second part of creating an ODBC recordset is to create the ODBC object itself:

```
handle = new ODBC( connection_handle [ , query ] )
```

where “`connection_handle`” is the handle of the `ODBC_CONNECTION` object created earlier and “`query`” is the optional SQL query to be used to create the recordset.

The return value is a handle to the new object. If the value is less than or equal to zero, the recordset creation failed. (Note that, even if you specify an invalid query, the creation can still succeed. It will simply contain no data.)

If a query is not specified when the object is created, you must specify it later by using:

```
ODBC handle QUERY query [ DYNASET|SNAPSHOT ] [ BOOKMARKS ]
```

The optional `DYNASET` and `SNAPSHOT` flags force those modes to be used for the recordset. If neither is specified, the driver's default mode is used. The optional `BOOKMARKS` flag enables bookmark support, if supported by the driver. (Note that enabling bookmarks may slow down data retrieval, so don't use it if you aren't going to be using bookmarks on the recordset.)

A query must be executed prior to accessing any of the records within the recordset.

Note that you can replace the recordset within an ODBC object by executing another `QUERY` command. Doing so will close the existing recordset and create a new one.

Moving around within the recordset

Once a recordset has been created, you can move around within it with the following methods:

```
ODBC handle GETFIRST  
ODBC handle GETLAST
```

```
ODBC handle GETPREV
ODBC handle GETNEXT
```

These will move the current record to the first, last, previous, and next records in the set, respectively.

Note that you only have sequential access within recordsets. This is a limitation of ODBC.

Note that ODBC does not include a method of determining the number of records returned within a recordset. The only way to do so is to `GETNEXT` through the set, counting the records, until EOF is reached. If there are no records being modified on the table by another process, you can run a separate query with “`SELECT COUNT (*)` ...” using the same `WHERE` clause. (If the table *is* being modified by another process, there is no guarantee that both queries will return the same set of records.)

Note that with some ODBC servers, the `GETLAST` operation may take a long time on large recordsets. If you need to access the last record and scroll backwards through the recordset, it is recommended that you instead use an `ORDER BY` clause in descending order, and scroll forwards from the first record.

Bookmarks

Although there is only sequential access through ODBC recordsets, it is possible (if the server supports it) to “bookmark” your current position and return to it later.

To retrieve a bookmark to the current position, you use the “`BOOKMARK GET`” method:

```
ODBC handle BOOKMARK GET bookmark
```

where “bookmark” is the variable in which to store the bookmark. This variable must be at least 16 bytes long, and of type “*”.

If the bookmark cannot be made (for example, you didn’t specify the `BOOKMARKS` flag on the query), the variable will be set to blank. (ie: `bookmark=""` will test true.) In that case, `@ODBCERROR[]` will contain the error. If the bookmark succeeds, the `bookmark` variable will contain something other than all blanks. (ie: `bookmark=""` will test false.) The bookmark data is not in a human-readable format.

Once a bookmark has been made, you can return to that position within the same recordset at any time by using the “`BOOKMARK SET`” method:

```
ODBC handle BOOKMARK SET bookmark
```

where “bookmark” is the same variable used to `GET` the bookmark earlier.

Note that there is no guarantee that a bookmark will remain valid for anything other than the current recordset. Even executing a `REQUERY` may invalidate the bookmark.

Note that there can be multiple bookmarks per recordset.

Refreshing a recordset

Not every ODBC data source can return a dynamic data set. Rather, they return a “snapshot” of the records at the time that the query was executed. (ie: if records are updated, added, or deleted while the recordset is open, you won’t see those changes.) In order to get an updated snapshot, you need to re-execute the query.

Rather than requiring that your code keep track of the query used, you can simply “requery” the recordset using the same query as the current set:

```
ODBC handle REQUERY
```

You will be positioned back to the first record, if any.

Closing a recordset

Once you are finished with a recordset, you can close it without deleting the ODBC object. (This allows future queries to be executed on the same connection, without the overhead of creating a new object.)

```
ODBC handle CLOSE
```

Once the recordset is closed, no more operations may be performed on it, aside from `DELETE` and `QUERY`.

Accessing columns within the recordset

Accessing the columns within a recordset is done with the new `@ODBC[]` system array, which is accessed using the syntax:

```
@ODBC.handle[subscript]
```

where “handle” is the handle to the ODBC recordset object, and “subscript” is the column number or name.

As with all system arrays, subscript zero returns the number of elements within the array, and accessing an out-of-range subscript returns an empty string.

If the number of elements is zero, then there are no columns available. This could be due to query returning no records, or positioning the current record to BOF, EOF, or a deleted record.

An enhancement from other system arrays is that “subscript” can be the name of the column, in addition to the numeric column number. For example:

```
@ODBC.handle["zipcode"]
```

This is especially important when doing “`SELECT *`” clauses on older ODBC servers, where the order of the columns cannot be guaranteed. (Or even on newer servers, where the database can have columns inserted within the existing structure, thereby changing the column number of the following ones.)

Note that there is currently only read-only access to the recordset.

Accessing column information

There are “class members” within the `@ODBC[]` array to access information about the columns.

To return the names of the columns, use the “`.NAME`” member:

```
@ODBC.handle.NAME[column]
```

Note that expressions and aggregate functions have default names generated by ODBC. To explicitly specify a name for the column, you need to append “`AS name`” to the expression, such as “`SELECT MAX(sales) AS MaxSales ...`”

To return the types of the columns (as a server-specific name), use the “`.TYPE`” member:

```
@ODBC.handle.TYPE[column]
```

To return the types of the columns (as a server-independent number), use the “`.TYPENUM`” member:

```
@ODBC.handle.TYPENUM[column]
```

I have not found a consolidated list of ODBC type numbers. However, I have gathered the following from numerous sources:

SQL_GUID	-11	SQL_BIGINT	-5
SQL_WLONGVARCHAR	-10	SQL_LONGVARBINARY	-4
SQL_WVARCHAR	-9	SQL_VARBINARY	-3
SQL_WCHAR	-8	SQL_BINARY	-2
SQL_BIT	-7	SQL_LONGVARCHAR	-1
SQL_TINYINT	-6	SQL_UNKNOWN_TYPE	0

SQL_CHAR	1	SQL_REAL	7
SQL_NUMERIC	2	SQL_DOUBLE	8
SQL_DECIMAL	3	SQL_DATE	9
SQL_INTEGER	4	SQL_TIME	10
SQL_SMALLINT	5	SQL_TIMESTAMP	11
SQL_FLOAT	6	SQL_VARCHAR	12

Note that these members can also use the field name as the subscript. (Yes, even the “.NAME” member can take a name as its subscript, as redundant as that may seem.)

Accessing the record state

To determine the current record state, the following members are defined:

```
@ODBC.handle.EOF
@ODBC.handle.BOF
@ODBC.handle.DELETED
@ODBC.handle.STATE
```

The EOF, BOF, and DELETED members hold whether the current record is end-of-file, beginning-of-file, or deleted, respectively. The value is “0” for false, and “1” for true.

Note: If both EOF and BOF are true, then the recordset contains no records.

The STATE member holds one of the following:

- | | |
|-------------------------------|---------------------|
| 0 – The recordset is not open | 3 – End-of-file |
| 1 – The record contains data | 4 – Deleted record |
| 2 – Beginning-of-file | 5 – Empty recordset |

Accessing raw SQL statements

Not every SQL action involves a returned recordset. For example, INSERT INTO or DELETE clauses. (Actually, SELECT is the only clause that *does* return a recordset.) Such statements need to be executed differently than SELECT clauses.

```
ODBC_CONNECTION handle EXECSQL sql_statement
ODBC handle EXECSQL sql_statement
```

Although such SQL statements are passed directly to the ODBC server via the ODBC_CONNECTION object, for convenience you can also use the ODBC recordset handle, in which case the statement will be passed to the connection object to which the recordset object is attached.

Handling errors

Currently, there really isn’t much error handling ability included.

If a query fails, for example if no records are selected due to a WHERE clause, then @ODBC.handle[“0”] will return zero, indicating no data is available. (Note that this is also true should you scroll to BOF or EOF, or position to a deleted record.)

The system array @ODBCERROR[] will contain the text of the most recent ODBC failure. Subscript 1 will contain the human-readable text of the error, and subscript 2 will contain the state information from which you can programmatically extract the error information. For more information, see

http://msdn.microsoft.com/library/en-us/vcmfc98/html/_mfc_cdbexception.3a3a.m_strstativeorigin.asp

Additional features needed

Ability to close connections, without deleting the object.

Add, update, and delete capabilities within a recordset. (This may not be necessary, as it can be done with the EXECSQL verb.)

Better error handling.

Example

The following example opens a connection to a Microsoft Access database, and allows you to scroll through it. Press “Q” to enter a query. This will display the first record returned. (Press Enter to clear the listbox.) Pressing “F”, “L”, “P”, or “N” will scroll to the first, last, previous, or next record, respectively. Pressing “G” or “S” will get or set the a bookmark, respectively. Pressing “E” will show the text of the most recent ODBC error.

```
end
declare connection(5,.0,g), recordset(5,.0,g), myquery(200,*,g)
declare bookmark(16,*,g)
@once:
  connection = new ODBC_CONNECTION("Driver={Microsoft Access Driver (*.mdb)};
    DBQ=c:\mdb\filepro.mdb;")
  recordset = new ODBC(connection)
end
showit:
  If: @odbc.recordset["0"] gt "0"
  Then: xx = listbox(@odbc.recordset)
  If: not showit
  Then: msgbox "Record contains no data: " < @odbc.recordset.state
  return
@keyQ:
  input popup myquery "Enter query: " default
  If: myquery = ""
  Then: end
  odbc recordset query myquery
  gosub showit
end
@keyN:
  odbc recordset getnext ; gosub showit ; end
@keyP:
  odbc recordset getprev ; gosub showit ; end
@keyF:
  odbc recordset getfirst; gosub showit ; end
@keyL:
  odbc recordset getlast ; gosub showit ; end
@keyE:
  errorbox @odbcexception["1"] & "\n---\n" & @odbcexception["2"]
end
@keyG:
  odbc recordset bookmark get bookmark
  If: bookmark = ""
  Then: errorbox "Failed to get bookmark" ; end
  msgbox "Record has been bookmarked" ; end
@keyS:
  odbc recordset bookmark set bookmark ; gosub showit ; end
```

Proposed for version 1.1

The following are *proposed* for version 1.1 of fileProODBC.

Updating of records via the ODBC object

Rather than requiring using the `EXECSQL` method and manually building an `UPDATE`, `INSERT`, or `DELETE` query, there will be methods for updating or deleting the current record in the recordset, or to insert new records.

Copying columns between recordsets and arrays

To make accessing all of the columns within the current record less cumbersome, you can copy the entire record into an array as follows:

```
ODBC handle COPY TO arrayname
```

This will do a field-by-field copy of each column in the current record to the specified fields in the array, as if you had executed the following for each column:

```
arrayname[subscript] = @ODBC.handle[subscript]
```

Note that if the array has less elements than the recordset has columns, only the first n columns will be copied. If the array has more elements, the rest of the array is set to blanks.

Once in the array, you can manipulate the data just as any other “normal” filePro field. For example, if the array is mapped to dummy fields, those fields can be placed on the screen or output format.

(Once updating via the ODBC object is included.) You can also update the current record by copying the array back to the ODBC recordset:

```
ODBC handle COPY FROM arrayname
```

A method for copying the column names and other information to an array will also be included.

Connection wizard

A wizard to help build connection strings will be written.

Error handling

More error handling will be available.

To do: include a method for accessing the reason for failure.

```
handle = new ODBC_CONNECTION( [CONNECTION_STRING] )
handle = new ODBC( connection_handle [ , query ] )
ODBC handle QUERY query [ DYNASET|SNAPSHOT ] [ BOOKMARKS ]
ODBC handle [ GETFIRST | GETLAST | GETPREV | GETNEXT ]
ODBC handle BOOKMARK [ GET | SET ] bookmark
ODBC handle REQUERY
ODBC handle CLOSE
@ODBC.handle[subscript]
@ODBC.handle.NAME[column]
@ODBC.handle.TYPE[column]
@ODBC.handle.TYPENUM[column]
@ODBC.handle.EOF
@ODBC.handle.BOF
@ODBC.handle.DELETED
@ODBC.handle.STATE
ODBC_CONNECTION handle EXECSQL sql_statement
ODBC handle EXECSQL sql_statement
@ODBCERROR[]
```