MS Excel, MS Access and SQL are no secrets for Xbase++

This article briefly outlines the design goals of the ODBC DatabaseEngine whose preview version is currently being released to the Alaska Software Professional Subscription members. The ODBC DatabaseEngine provides Xbase++ developers with an easy way for accessing data managed by MS Excel, MS Access or SQL database servers. This opens up a completely new field of business opportunities for database application developers.

Introduction and Design Goals

Open Database Connectivity (ODBC) is a programming standard for database access established by Microsoft. This middleware layer is widely accepted and supported by many vendors of Database Management Systems (DBMSs). DBMSs become accessible with appropriate ODBC drivers which expose their capabilities to application programs via the standardized ODBC API.

The general goal of the ODBC DatabaseEngine (ODBCDBE) is to embed the ODBC capabilities within the Xbase++ programming language. This language is especially designed for integrated database access and focuses on both, the requirements for implementing business logic and the needs for user-interface development. This makes the Xbase++ programming language different from other programming languages where database functionality must be considered as an add-on feature that is not integrated in the application development language.

The specific design goals to achieve a seamless integration of both worlds, Xbase++ as programming language and ODBC as database access standard, are defined as follows:

1. Database access and manipulation must remain transparent to the business logic.
2. Existing Xbase-style coding patterns to browse, update and query data must work without code changes.
3. The coding pattern for accessing databases of SQL DBMSs must not differ from coding patterns used for file-based data sources, such as DBF tables or MS Excel spread sheets.
4. ODBC data sources must fit into the work space and work area model of Xbase++ and must be automatically isolated between threads.

As the core requirements to reach these design goals, we have defined our objectives like this:

1. Provide an intelligent ODBC DatabaseEngine. It must be able to automatically detect the capabilities of the accessed ODBC driver in detail and must expose ODBC drivers' functionalities in a unified manner. If necessary, encapsulate data source specific functionalities within the DatabaseEngine, such as uni- or bi-directional navigation or specific update patterns.

2. Provide a class framework to support extended features of an accessed DBMS and support an object oriented coding style and data access methodology.

The ODBCDBE meets the requirements of the first target. The second target is already part of Xbase^{++} to some extent: the Data Access Chain (DAC). The DAC class framework is especially designed to provide next generation DBMS access based on an object oriented methodology. As a result, the integration of the ODBC standard in the Alaska Software technology grants a maximum level of interoperability between application programs and various DBMSs.

Basic Usage Pattern

No matter what data source is to be accessed with the ODBCDBE, there is a simple and standard programming pattern involved. It establishes a connection with the ODBC driver using a DacSession object. An example is this:

```
01: PROCEDURE Main
02:    LOCAL cConnect := "DBE=ODBCDBE;DSN=OrderProcessing"
03:    LOCAL oSession := DacSession():new( cConnect )
04:
05:    IF .NOT. oSession:isConnected()
06:       ? "Unable to connect to server !"
07:       ? oSession:getLastMessage()
08:       QUIT
09:    ENDIF
10:
11:    USE Customer
12:    < Business Logic As Usual >
13:    DbCloseAll()
14:
15:    oSession:disconnect()
16: RETURN
```

This code illustrates the basic usage pattern for the ODBCDBE: (line #2) defines the ODBC connection string, (#3) uses this string to create a DacSession object and (#5) checks if the connection was successfully established. If so, the data source is accessible using standard Xbase^{++} database commands and functions. When the data source is no longer needed, the application program must disconnect from the server (#15).

It must be emphasized that it does not matter for the business-logic part of the code (line #11-13) if the Customer table resides on a SQL server or is managed by MS Access or MS Excel. Data access is coded in the same way for different data sources. The only difference for the Xbase^{++} developer is that connecting to different data sources requires different connection strings. All internal differences between various data sources are intelligently handled by the ODBCDBE.

The Connection String

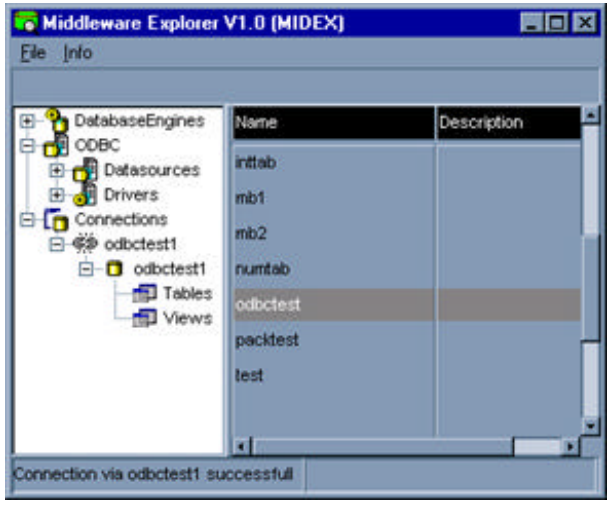
The connection string contains information required for establishing a connection between an application program (client side) and a data source (server side). It is comprised of a list of name=value pairs separated with semi-colons. This string is the most important part for successfully using the ODBCDBE since a connection will fail if connection data is incorrect. If the connection fails, the data source is not available and the ODBCDBE cannot be used. Examples for connection strings are:

```
DBE=ODBCDBE;DRIVER=Microsoft Excel Driver (*.xls); +
FIL=excel 8.0;UID=Admin;DBQ=testdb.xls
```

```
DBE=ODBCDBE;DRIVER=SQL Server;SERVER=ALASKA02;UID=John;PWD=Camelot; +
DATABASE=SalesDepartment;WSID=W35;Trusted_Connection=Yes
```

These two strings illustrate that different ODBC drivers do not use the same name=value pairs in the connection string. What information is required and/or supported by a certain driver is documented in the manuals of the software providing the data source. However, it can be a tedious search job to find the correct connection string "format" for an unknown data source.

As a solution for this problem, we have added a Middleware Explorer (MIDEX) to the ODBCDBE that can be used to inspect all ODBC drivers installed on a computer and to connect to all corresponding data sources.



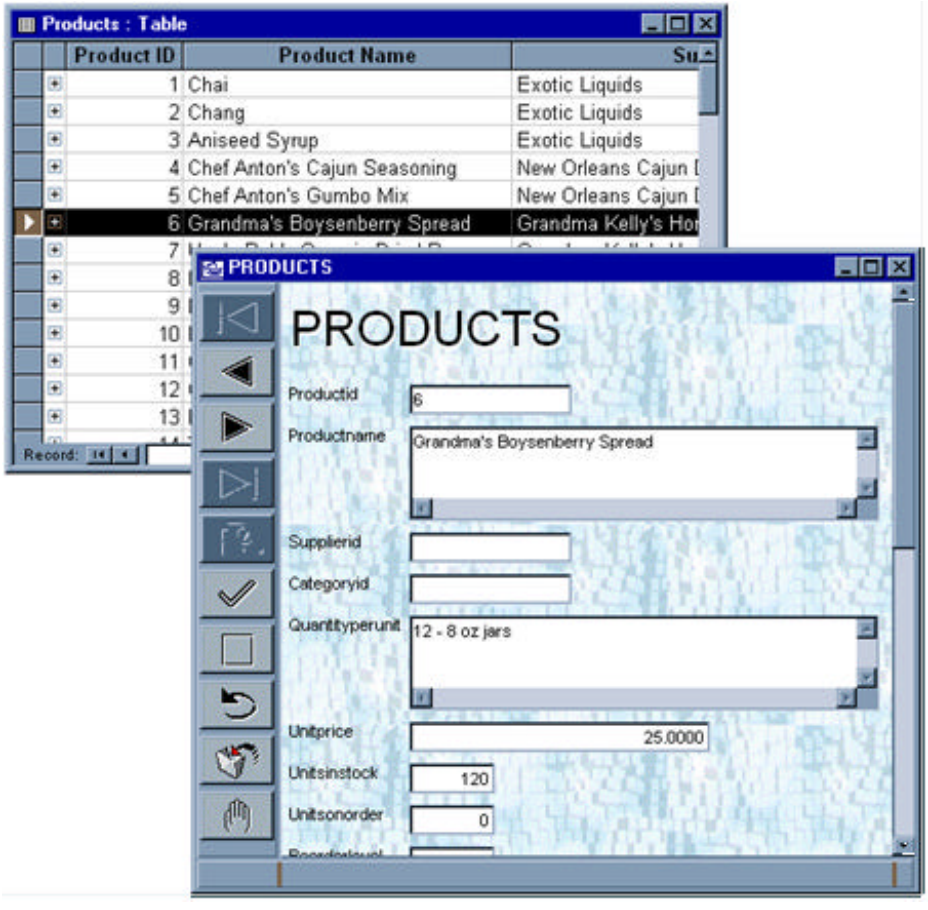
If you do not know how a connection string must be coded, you can use MIDEX to establish a connection to the data source in question, copy the connection string to the clipboard and paste it into your source code editor.

Examples for ODBCDBE usage

This section illustrates how easy it is to access different data sources with the ODBCDBE. In the following we list the Xbase^{++} code used to create the Xbase^{++} applications shown in the examples. In the background of each pictuer you can see screen shots of the software managing the data source.

MS Access

Data used in this example comes from the Northwind example database included in MS Access. The screen shots show an APPEDIT window created with Xbase $^{++}$ and the MS Access browser displaying the same table from the Northwind.mdb file.



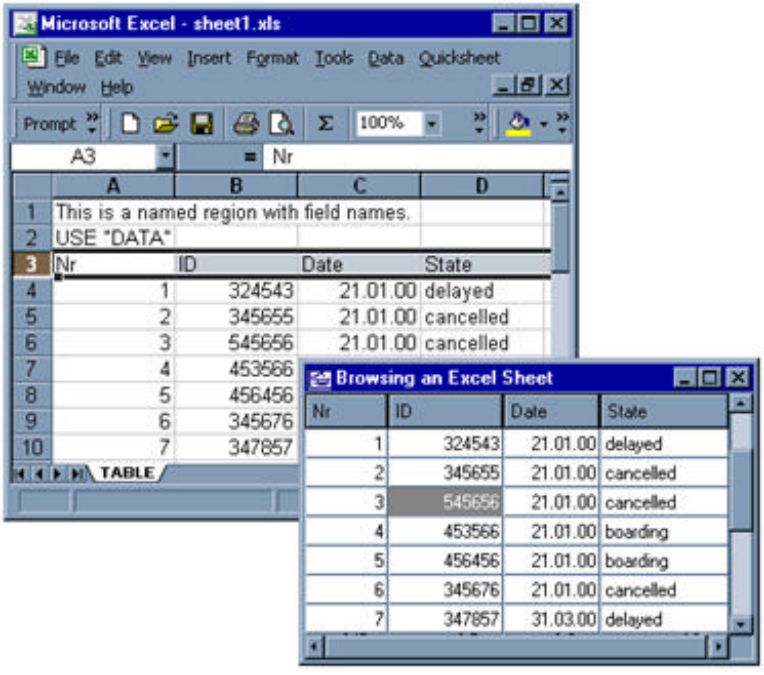
This is the code used to create the screen shot for the APPEDIT window:

```
PROCEDURE Main
   LOCAL cConnect := "DBE=ODBCDBE;DSN=NorthWind"
   LOCAL oSession := DacSession():new( cConnect )

   IF .NOT. oSession:isConnected()
      MsgBox( "Unable to connect to Access database!" )
      QUIT
   ENDIF

   USE Products
   APPEDIT STYLE FANCY POSITION CENTER
   APPDISPLAY

   USE
   oSession:disConnect()
RETURN
```

The connection string includes "DSN=NorthWind". Using the Data Source Name is the easiest way to code a connection string since all connection data details are provided by the ODBC data source as it is defined on your workstation.

MS Excel

The data source taken for these screen shots is the file Sheet1.xls included in a sample program of the ODBCDBE.



To demonstrate that Excel spread sheets can be accessed like a DBF file, we have used the code of a slightly modified Xbase^{++} example program to produce the browser displaying XLS data. You can find this code in ..\SOURCE\SAMPLES\BASIC\GUIBROW\DBBROWSE.PRG, the following code shows only the relevant parts for creating the image.

```
PROCEDURE Main
   LOCAL nEvent, mp1, mp2, oXbp, i, imax
   LOCAL cConnect := "DBE=ODBCDBE;DSN=ExcelSheet1"
   LOCAL oSession := DacSession():new( cConnect )

   IF .NOT. oSession:isConnected()
      MsgBox( "Unable to connect to server !" )
      QUIT
   ENDIF
   USE DATA
   oXbp := GuiStdDialog( "Browsing an Excel Sheet" )
```

```
   // create browser in window
   oBrowse := GuiBrowseDb( oXbp:drawingArea )

   // create columns for all fields
   imax    := FCount()
   aStruct := DbStruct()
   FOR i:=1 TO imax
      oBrowse:addColumn( FieldBlockTrimmed(aStruct[i,1], ;
                           aStruct[i,2]), , aStruct[i,1] )
   NEXT

   // overload resize so that browser fills the window
   oXbp:drawingArea:resize := ;
       {|mp1,mp2,obj| obj:childList()[1]:setSize(mp2) }

   // show browser and set focus on it
   oXbp:show()
   oBrowse:show()
   SetAppFocus( oBrowse )

   DO WHILE nEvent <> xbeP_Close
      nEvent := AppEvent( @mp1, @mp2, @oXbp )
      oXbp:handleEvent( nEvent, mp1, mp2 )
   ENDDO

   USE
   oSession:disConnect()
RETURN
```

A database in an Excel spread sheet is a named cell range (cell ranges are named using Insert->Names from Excel's menu). This means that one spread sheet may contain multiple tables from the ODBC point of view. The first row of the cell range contains character strings describing the name of each column. The column names are used as the field names of the table when it is opened with the USE command in Xbase^{++}.

If there is no named cell range defined in the spread sheet, a table can be opened by explicitly specifying a range of cells using Excel's syntax. The next line of code illustrates this approach:

```
USE "A1:E122" ALIAS Customer
```

In this case, data contained in the cells from A1 to E122 become accessible in a work area having Customer as alias name.

SQL DBMS

Tables are opened in traditional Xbase programming style with the USE command. This is also possible when tables are managed by a SQL DBMS and are accessed via the ODBCDBE. However, there is a major difference between Xbase tables and SQL tables that require some kind of paradigm shift in database programming. The reason for this is given by the fact that Xbase tables are file-based data sources (DBF files) while SQL database tables are non-file-based data sources and are part of a complete data model consisting of various other tables. For example:

```
USE Customer ALIAS Cust NEW   // assume: VIA "DBFNTX"

USE Customer ALIAS Cust NEW   // assume: VIA "ODBCDBE"
```

The first USE command would open a file CUSTOMER.DBF, it requires a file handle in order to make customer data accessible in a work area.

The second USE command does not differ from the first one, except that it is addressed to the ODBCDBE. This DatabaseEngine does not open a file, but instructs the connected SQL DBMS to prepare a resulting data set including all rows of the Customer table. This leads to an important difference to the traditional Xbase programming approach: **USE requests the server to transfer data to the client**. The effect is the same as with a DBF table: data becomes accessible in a work area on the client side of an application and a work area is used.

When data is managed by a SQL DBMS, the USE command submits a request to the server. In its simplest form, USE can be programmed in the traditional Xbase way by specifying the table name. However, it has an extended functionality in that SQL SELECT statements can be submitted. This instructs the server to prepare a set of rows and/or table columns (fields). For example, this is legal code when using the ODBCDBE:

```
cSQL := "SELECT LastName, FirstName, Phone FROM Customer;"
USE (SQL) ALIAS Cust NEW

Browse()
```

This SQL SELECT statement requests the server to prepare a data set of the Customer table including only three columns, or fields. It is coded as a character string which is passed to USE. Note the semicolon at the end of the string: it must be included as the last character and marks the end of the SQL SELECT statement.

The result of this code is that only a limited amount of customer data is transferred to the client application where it becomes accessible in a work area. As a matter of fact, the function FCount() would return 3 (since only 3 table columns were requested to be available in the work area), although the Customer table, as it is defined on the server side, may have multiple other columns storing data for State, Zip, City, Address etc. In turn, the Browse() function, which displays all fields (columns) of a used work area, would list only three columns.

The nature of SQL DBMS allows a server to expose tabular data not only in form of a single database table but also in form of a View. A view may span across several individual tables but can be displayed like a single database table maintained by a DBMS. As a result, the symbolic name of a view can be passed to the USE command in order to display the data defined as a view on the server side.

Conclusion

The USE command has an extended functionality when it is processed by the ODBCDBE:

```
USE "single table name"
USE "SQL SELECT statement;"
USE "View defined on DBMS"
```

The result is alway the same: a request is submitted from the client application to the DBMS (server side). The DBMS transfers requested data to the client where tabular data becomes accessible in a work area.

Summary

The ODBCDBE integrates the ODBC capabilities seamlessly into the Xbase^{++} language. Xbase^{++} developers gain transparent access to various data sources and can use the same coding patterns, no matter whether the data source is a spread sheet, a DBF file or a SQL table. The only thing to consider is whether or not to use a DacSession object for connecting to a data source, and how to build the connection string (MIDEX helps here in case of questions).

We believe that the approach we have chosen is the best way to integrate non-Xbase databases into the Xbase^{++} programming language. We are committed to make Xbase^{++} the programming language of choice for database applications and we will continue to improve Xbase^{++} and the ODBCDBE to provide you with the right tools to run your business. There are still many things in the pipeline of the Alaska Software DevTeam, so stay tuned...