



Technical Information

Product Description

f90SQL is a library of functions and subroutines that work as an interface between your Fortran programs and Microsoft Windows Open Database Connectivity (ODBC) API. *f90SQL* offers a convenient and familiar way to directly read and write data from your Fortran programs to many applications formats. The list below includes a few of the most common formats, but your options are basically unlimited. As long as the application offers an ODBC interface to its data files (almost all DBMS in the market today do), *f90SQL* lets you read/write data to the native application's format directly from your Fortran programs.

Common Application Formats Accessible Through *f90SQL*

- Excel and Lotus 1-2-3 spreadsheets
- Microsoft Access
- FoxPro
- Paradox
- Oracle
- Ingres
- Informix
- Microsoft SQL-server
- Any application format that offers an ODBC interface to its data files

Key Benefits

- Use SQL language to access application data, regardless of the native format of the data.
- Read or write data directly from Fortran to almost all database systems available for Windows, including SQL-server, MS-Access, Oracle, Ingres, Sybase, and many others.
- *f90SQL* offers interfaces to all standard ODBC 3.5 API functions.
- Create queries dynamically at run-time. *f90SQL* is not embedded SQL, so your queries do not become fixed by a pre-compiler. Your programs can develop new queries based on run-time logic.
- Your Fortran programs get direct access to the application's data. No more clumsy, time consuming and error-prone extractions to ASCII files or importing data generated from your programs into other database applications.
- *f90SQL* subroutines use the familiar Fortran protocol to receive and return parameters. You do not have to deal with DLLs or ODBC-API calling conventions.
- *f90SQL* adds little overhead to your applications. It offers a small memory footprint and maximum performance.
- A full manual, with numerous examples, will guide you step by step on how to access data from your Fortran applications.
- *f90SQL* is supported by *Canaima Software*, a consulting company with ample experience in scientific and database programming.

System Requirements

- An Intel-based computer running Microsoft Windows 95, Windows 98 or Windows NT/2000.
- A Fortran compiler. *f90SQL* is currently available for Digital Visual Fortran (ver. 5.0 or 6.x), Absoft Pro-Fortran (ver. 5.0 or 6.x), Lahey Fortran 90 (ver. 4.5x), Lahey/Fujitsu Fortran 95 (ver. 5.x) and Salford Fortran 95 (ver. 1.28 and higher)
- ODBC drivers for the proprietary format you want to access (these drivers are usually provided by the application's vendors).

Documentation

Complete tutorial and reference manual in electronic format (HTML and HTML-Help). Printed documentation is also available for a nominal fee.

License, Royalties and Run-time Fees

f90SQL is licensed on a developer-seat basis, *i.e.* each developer or programmer in your organization using the *f90SQL* library should obtain a license. There are no associated run-time fees or royalty payments. Multi-developer, campus-wide, and corporate site licenses are also available.

Support

Support for *f90SQL* is provided by *Canaima Software*. A *f90SQL* license includes six months of unlimited e-mail support and one-year free upgrades. Phone support and extended e-mail support can also be purchased from *Canaima Software*.

Price: US\$ 249.00 per developer-seat + Shipping & Handling

For More Information

Contact:

Canaima Software, Inc.

P.O. Box 13162

La Jolla, CA 92039

USA

Phone: (619) 233-6831

e-mail: sales@canaimasoft.com

Lahey Computer Systems, Inc.

865 Tahoe Boulevard

P.O. Box 6091

Incline Village, NV 89450-6091

Phone: (702) 831-2500

e-mail: sales@lahey.com

or visit our web page at www.canaimasoft.com

Copyright ©1998-2000 *Canaima Software*.

All trademarks are property of their respective owners.

ODBC API Functions Supported by f90SQL¹

Connecting to data sources:

Function	Conformance	Purpose
SQLAllocHandle	ISO 92	Obtains environment, connection, statement, or descriptor handles
SQLConnect	ISO 92	Connects to a specific driver by data source name, user ID, and password.
SQLDriverConnect	ODBC	Connects to a specific driver using a connection string, or requests that the Driver Manager and driver display connection dialog boxes for the user.
SQLBrowseConnect	ODBC	Returns successive levels of connection attributes and valid attribute values, and connects to the data source when all connection attributes have been specified.

Obtaining information about a drivers and data sources:

Function	Conformance	Purpose
SQLDataSources	ISO 92	Returns a list of available data sources.
SQLDrivers	ODBC	Returns a list of installed drivers and their attributes.
SQLGetInfo	ISO 92	Returns information about a specific driver and data source.
SQLGetFunctions	ISO 92	Returns supported driver functions.
SQLGetTypeInfo	ISO 92	Returns information about supported data types.

Setting and retrieving driver attributes:

Function	Conformance	Purpose
SQLSetConnectAttr SQLGetConnectAttr	ISO 92	Sets and retrieves connection attributes.
SQLSetEnvAttr SQLGetEnvAttr	ISO 92	Sets and retrieves environment attributes.
SQLSetStmtAttr SQLGetStmtAttr	ISO 92	Sets and retrieves statement attributes.

Setting and retrieving descriptor fields:

Function	Conformance	Purpose
SQLGetDescField SQLSetDescField	ISO 92	Returns and sets the value of a single descriptor field.
SQLGetDescRec SQLSetDescRec	ISO 92	Returns and sets the values of multiple descriptor fields.

Preparing SQL requests:

Function	Conformance	Purpose
SQLPrepare	ISO 92	Prepares a SQL statement for later execution.
SQLBindParameter	ODBC	Assigns storage for parameters in a SQL statement.
SQLGetCursorName	ISO 92	Returns the cursor name associated with a statement handle.
SQLSetCursorName	ISO 92	Specifies a cursor name.
SQLSetScrollOptions	ODBC	Sets options that control cursor behavior.

Terminating statements and closing connections:

Function	Conformance	Purpose
SQLFreeStmt	ISO 92	Ends a statement and frees resources associated with the statement.
SQLCloseCursor	ISO 92	Closes a cursor.
SQLCancel	ISO 92	Cancels an SQL statement.
SQLEndTran	ISO 92	Commits or rolls back a transaction.

¹ Some vendor's ODBC drivers do not support all the ODBC 3.5 API functions.

Function	Conformance	Purpose
SQLDisconnect	ISO 92	Closes the connection.
SQLFreeHandle	ISO 92	Releases environment, connection, statement, or descriptor handles

Retrieving results and information about results:

Function	Conformance	Purpose
SQLRowCount	ISO 92	Returns the number of rows affected by a SQL statement.
SQLNumResultCols	ISO 92	Returns the number of columns in the result set.
SQLDescribeCol	ISO 92	Describes the columns in the result set.
SQLColAttribute	ISO 92	Describes attributes of columns in a result set.
SQLBindCol	ISO 92	Assigns storage and specifies the data type of a result column.
SQLFetch	ISO 92	Returns multiple result rows.
SQLFetchScroll	ISO 92	Returns scrollable result rows.
SQLGetData	ISO 92	Returns part or all of one column of one row of a result set (useful for long data values).
SQLSetPos	ODBC	Positions a cursor within a fetched block of data, and allows an application to refresh data in the rowset, or update or delete data in the result set.
SQLBulkOperations	ODBC	Performs bulk insertions and bulk bookmark operations, including update, delete, and fetch by bookmark.
SQLMoreResults	ODBC	Determines whether there are more result sets available and, if so, initializes processing for the next result set.
SQLGetDiagField SQLGetDiagRec	ISO 92	Returns additional diagnostic information (single and multiple fields of the diagnostic data structure).

Catalog functions:

Function	Conformance	Purpose
SQLColumnPrivileges	ODBC	Returns a list of columns and associated privileges for one or more tables.
SQLColumns	X/Open	Returns a list of column names in specified tables.
SQLForeignKeys	ODBC	Returns a list of column names that make up foreign keys.
SQLPrimaryKeys	ODBC	Returns the list of column names that make up a table's primary key.
SQLProcedureColumns	ODBC	Returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures.
SQLProcedures	ODBC	Returns the list of procedure names stored in a data source.
SQLSpecialColumns	X/Open	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated by a transaction.
SQLStatistics	ISO 92	Returns statistics about a single table and the list of indexes associated with the table.
SQLTablePrivileges	ODBC	Returns a list of tables and the privileges associated with each table.
SQLTables	X/Open	Returns a list of table names stored in a data source.

In addition to the previous functions, *f90SQL* provides utility functions, constant definitions, and Fortran data structures that facilitate the use of ODBC from your Fortran applications. All *f90SQL* functions receive and return arguments according to the common Fortran convention.

ODBC and SQL meet DVF with Canaima Software's *f90SQL*.

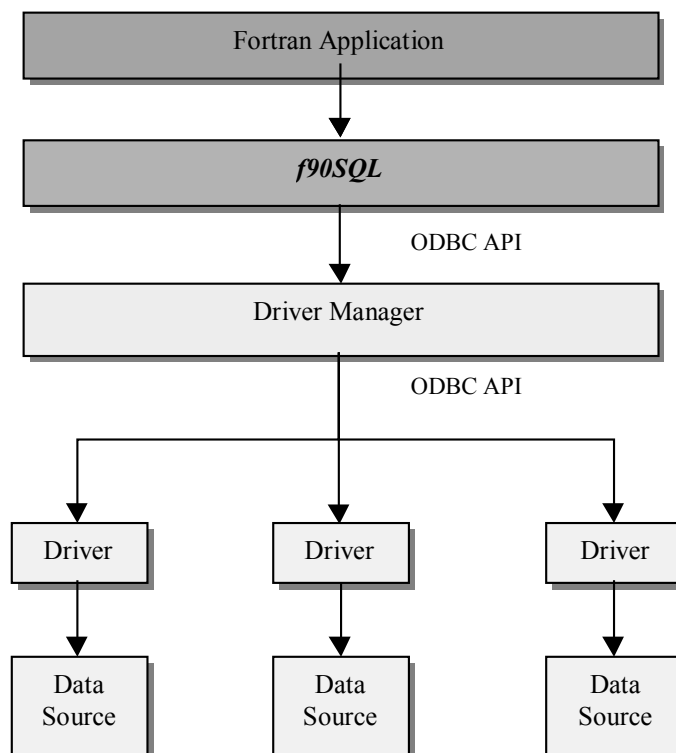
By Marco A. Garcia

(Reprinted with permission from DIGITAL Visual Fortran Newsletter, Issue III, October 1998)

Most scientists and engineers who need to handle large amounts of data are aware of the convenience of keeping this information in databases. In addition to allowing indexed access to the data, Database Management Systems (DBMS) generally offer many tools that facilitate the organization and maintenance of the information they store. One of the most exciting developments in the area of database technology is the current trend towards source interoperability. The idea behind this is to offer end-users a framework in which to access data that is independent of the format and, in many cases, the location of the data. In Microsoft Windows environments, this interoperability can be achieved through the use of the Open Database Connectivity (ODBC) standard.

ODBC is an Application Programming Interface (API), i.e. a more or less coherent library of functions and subroutines that can be called from your application. One of the main advantages of the ODBC-API is that an application can use the same interface to access data stored in many different proprietary formats. The application calls functions in the ODBC interface, which are implemented in database-specific modules called drivers (see Figure 1). The use of drivers isolates applications from database-specific calls in the same way that printer drivers isolate word processing programs from printer-specific commands. The programmer creating the application uses the same set of instructions to access the data, independently of whether the data is stored in, for example, a spreadsheet, a table in a SQL server, or a Paradox file. Also, because drivers are loaded at run time, a user has only to add a new driver to access a new DBMS; it is not necessary to recompile or relink the application.

Figure 1. *f90SQL* and ODBC Architecture



If you are wondering what applications offer access to their proprietary formats through ODBC drivers, the list is likely to be longer than this article. Just to mention a few, using ODBC your Fortran applications can access data stored in Excel and Lotus 1-2-3 spreadsheets, Microsoft Access, FoxPro, Paradox, Oracle, Ingres, Informix, Microsoft SQL-server, Progress, Btrieve, ADABAS D, SQLBase, DB2, Sybase, OpenVMS RMS, and many others.

Although Microsoft's ODBC Software Development Kit claims that the ODBC-API is language-independent, the truth is that Windows' APIs can be difficult to use from any language other than C. For a Fortran programmer, direct calls to the ODBC-API means large amounts of developing time devoted to deal with the problems of passing pointers, converting strings from C format to Fortran format and back, taking care of whether variables are passed by value or by reference and a myriad of other little details. The complications that result from calling these functions from Fortran frequently are enough to discourage programmers from using them. Many programmers decide to develop their numerical applications in C or Visual Basic because they needed access to information stored in databases, even though Fortran could produce better performance.

In this article I will introduce f90SQL, a new tool that works as an interface to the ODBC-API, making it more accessible from Fortran programming environments. I will also illustrate how f90SQL can be used to add functionality to your DIGITAL Visual Fortran applications. To do this, I will briefly explain the main steps used to write an ODBC application and sketch a simple Fortran program in which I apply these principles using f90SQL.

Most ODBC applications follow a set of standard steps to read data from a source:

1. Initialize the ODBC environment and connect to the data source. In this step your Fortran application allocates the space necessary for the ODBC environment and the connection to the data source. The data source can be a file (for example, an Excel workbook) or a Data Source Name (DSN) that has been created using the ODBC-Administrator in your Windows Control Panel.
2. Initialize a statement. In this step the application allocates a "statement space" and sets the statement's attributes. You can think of a statement space as a buffer where the data returned by your queries will be temporarily stored. This step is also the point at which you tell ODBC whether you want, for example, a dynamic or a static cursor associated with your data. (A cursor is a link between the data in your application and the data stored in the server.)
3. Build and execute a query. In ODBC applications, queries are constructed using Structured Query Language (SQL) syntax. SQL offers a common language to retrieve/write information from/to all type of data sources. The data retrieved by your SQL query is called a "record set", and is temporarily stored in the statement space associated to the query. A record set is organized as a table, with rows (records) and columns (fields). Your application does not have direct access to the record set instead you "link" Fortran variables to each column, as explained in the next step.
4. Fetch and retrieve the data. To access the information in a record set, your application must "link" or "bind" Fortran variables to the columns in the record set. For example, if one of the columns in the record set contains integer values, then you bind a Fortran integer variable to that column. If another column has character values, you bind a Fortran character variable to that column, and so on. Once you have bound all or some of the columns of the record set you can start fetching the rows. Each time you fetch a row, ODBC will automatically update the Fortran variables with the values in the associated columns of the record set. A nice feature of ODBC and f90SQL is that you can bind a vector to a record set column and use a "block" cursor. With this technique your application can fetch and retrieve many rows at once.

5. Process the data. Your Fortran application now has access to the data in the form of Fortran variables, so you can use it to perform any desired operations.
6. Clean up. Once the application has finished using the data, you need to release the memory used by the statement, the connection and the ODBC environment.

For applications that write to a data source, steps 3 and 4 are slightly different, although in essence the process is the same. In this case, however, ODBC updates the rows and columns in the record set with the values you put in the associated (bound) Fortran variables.

Now that we have a general idea of the process involved in retrieving information from a data source, let's see how we can implement these steps in Fortran using f90SQL:

```

Program DVFNLEExample

!load f90SQL modules
use f90SQLConstants
use f90SQL

implicit none

!Declare ODBC handles (used to identify ODBC spaces)
integer(SQLHENV_KIND):: EnvHndl
integer(SQLHDBC_KIND):: ConnHndl
integer(SQLHSTMT_KIND):: StmtHndl

!Other relevant ODBC variables
integer(SQLRETURN_KIND)::iRet
integer(SQLSMALLINT_KIND)::ColNumber

!Fortran variables used to bind record-set columns
character(len=10)::CharVar
integer(SQLINTEGER_KIND)::IntVar
real(SQLREAL_KIND)::RealVar

!STEP 1: Initialize ODBC environment and connect to data source

!Allocate an environment space.
!The handle to the environment space is returned in variable EnvHndl
call f90SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, EnvHndl, iRet)

!Set ODBC version (3.x in this case)
call f90SQLSetEnvAttr(EnvHndl, SQL_ATTR_ODBC_VERSION, &
                    SQL_OV_ODBC3, iRet)

!Allocate a connection space.
!The handle to the connection space is returned in variable ConnHndl
call f90SQLAllocHandle(SQL_HANDLE_DBC, EnvHndl, ConnHndl, iRet)

!connect to the data source
call f90SQLConnect(ConnHndl, 'MyDatabaseDSN', 'MyUserName', &
                  'MyPasswrд', iRet)

!STEP 2: Initialize statement space
!The handle to the statement space is returned in variable StmtHndl

```

```

call f90SQLAllocHandle(SQL_HANDLE_STMT, ConnHndl, StmtHndl, iRet)

!STEP 3: Build and execute a SQL query to request data from data source
call f90SQLExecDirect(StmtHndl,'select * from MyTable', iRet)

!STEP 4: Fetch and retrieve the data

!Bind Fortran variables to result-set columns
!ColNumber represents columns in the result-set table
call f90SQLBindCol(StmtHndl, ColNumber, SQL_F_CHAR, CharVar, &
                  f90SQL_NULL_PTR, iRet)

call f90SQLBindCol(StmtHndl, ColNumber, SQL_F_INTEGER, IntVar, &
                  f90SQL_NULL_PTR, iRet)

call f90SQLBindCol(StmtHndl, ColNumber, SQL_F_REAL, RealVar, &
                  f90SQL_NULL_PTR, iRet)

!Fetch each row of the record set
do
    !Fetch a row
    call f90SQLFetch(StmtHndl, iRet)
    !Check if the last row of the record-set has been fetched
    if (iRet.eq.SQL_NO_DATA) exit

    !At this point, Fortran variables CharVar, IntVar, and
    !RealVar have the data of the associated columns of the current
    !record set's row

    !STEP 5: Process the data
    !INSERT YOUR PROCESSING INSTRUCTIONS HERE
enddo

!STEP 6: Clean up
!Release statement space
call f90SQLFreeHandle(SQL_HANDLE_STMT, StmtHndl, iRet)
!Release connection space
call f90SQLFreeHandle(SQL_HANDLE_DBC, ConnHndl, iRet)
!Release ODBC environment
call f90SQLFreeHandle(SQL_HANDLE_ENV, EnvHndl, iRet)

stop
end

```

There are several aspects of the previous program that you may have noticed. The program makes heavy use of kind-values for defining variables. These kind-values are defined in the f90SQL modules. They isolate the programmer from many details related to the use of ODBC variables. f90SQL modules also define many constants that can be used as arguments to its subroutines. Your programs are not restricted to use the pre-defined kind-values and constants, but using them guarantees that your applications will be compatible with future releases of the ODBC-API and f90SQL. You may have also noticed that, when f90SQLAllocateHandle is called to allocate spaces, you do not indicate the size of the space. ODBC takes care of this for you. In addition, ODBC returns a handle to the allocated space (in variable StmtHndl, for example), which your application uses to identify the space. An application can have several connection and/or statement spaces open at the same time.

Another conspicuous feature of the program is the loop to fetch the record set a row at a time. This is not the most efficient method to move data from a record set into your Fortran variables. I used this method for the sake of clarity, but as explained earlier, you can eliminate the loop completely and retrieve all the rows at once by using block cursors. Finally, you may have noticed the complete lack of error-checking instructions. Again this was done to keep the code as simple as possible. Each time you call a f90SQL subroutine or function, the procedure returns a status value in the argument iRet. This value is the same as the status value returned by the ODBC function called by the f90SQL procedure. An application should check iRet frequently, to ensure the called subroutines completed successfully.

Perhaps the most impressive characteristic of the ODBC-API is that you can use the program presented above to access data stored in almost any format. For example, if the queried table (MyTable) is stored in a Microsoft Access Database and later moved to an Oracle server, you may not have to compile your application again, a change to the DSN definition using the ODBC-Administrator could be enough. In addition, the same program with a few changes (change the SQL statement and add or remove calls to f90SQLBindCol) can be used to access any data from any table.

Hopefully this short article has given you a taste of how easy it is to access database information from your DIGITAL Visual Fortran applications, and perhaps, sparked some ideas of how to take advantage of this new functionality. f90SQL, the ODBC interface for DIGITAL Visual Fortran, was developed by Canaima Software. You can obtain more information about f90SQL, as well as a complete tutorial on f90SQL/ODBC, at <http://www.canaimasoft.com/>