Jörg Kuthe

# ForDBC

## Fortran Database Connectivity

Revision date: 19th of August 2014



**QT software GmbH**
**Karl-Friedr.-Gauß-Str. 18**
**45657 Recklinghausen**
**Germany**

**Phone +49(0)2361/4864760**
**Fax +49(0)2361/4856279**

**eMail: info@qtsoftware.de**
**www.qtsoftware.de**
**Shop: www.qtsoftware.com**

# Usage Rights, Limitation of Liability, Copyright

### §1. Property and Ownership

The software described in this document and this document itself is called ForDBC in the following. The ForDBC software consists of files whose names start with the letters "qt" or "ForDBC". A list of them can be found in the chapter “Installation and Launching ForDBC”. In particular these are library files ending on .lib, pre-compiled MODULE files (ending on .mod), Fortran source code files (ending on .f90). All these files are property of the author Jörg Kuthe. The author is represented by QT software GmbH, called QT hereafter, which is authorized to allocate use rights to ForDBC. The copyright at the ForDBC software and this documentation remains with the author.

### §2. Licensee

The licensee is the user of the ForDBC software and the buyer of a licence, which he has obtained from QT in the course of the purchase of the ForDBC licence.

### §3. Licence and Passing on of Components of the ForDBC Software

The licence consists of the right of the licensee to use the ForDBC software for the development of programs, i.e. thus to create executable files whose name ends on .exe. The executable file created with ForDBC may neither have the function of a compiler nor that one of a linker.
The passing on of any other file of ForDBC is not allowed.

### §4. Transfer and Assignment of the Licence

The licensee cannot be represented by another person. This excludes particularly the rental of the ForDBC software. The licensee may sell the use licence if he reports the disposal or assignment of the licence to another licensee to QT in writing. The assignment must include the confirmation, that the selling licensee gives up his rights at the ForDBC. The new licensee must agree to these licence conditions in writing.

### §5. Warranty

QT ensures the function ability of the ForDBC software for the period of 2 years after acquisition of the licence. In the fault case it is up to QT either to refund the paid price to the licensee or to fix the reported error in the ForDBC software. If the money paid for ForDBC is refunded, the licensee looses the right to use the software ForDBC. Complaints or faults have to be verified by the licensee by a program he provides.

### §6. Use Risk and Limitations of Liability

The licensee uses the software ForDBC on risk of his own. QT is liable in maximum amount of the paid price for ForDBC.

### §7. Declaration of Consent

The licensee gives his agreement to these conditions by the acquisition of the licence.



### Other Notes

The author and QT software GmbH acknowledge the rights of the owners at the brand names, trademarks and products named in this document:
Excel is a trademark of Microsoft Corporation, U.S.A..

Windows is a trademark of Microsoft Corporation, U.S.A..
"ProFortran for Windows" is a product of Absoft Corporation, U.S.A..
"Compaq Visual Fortran" is a product of Hewlett-Packard Company, U.S.A..
"Intel Visual Fortran" is a product of Intel Corporation, U.S.A..
"Lahey/Fujitsu Fortran 95 for Windows" is a product of Lahey Computer Systems, Inc., U.S.A..
"Salford FTN95" is a product of Salford Software Ltd., U.K..
"Silverfrost FTN95" is a product of Salford Software Ltd., U.K..
"Excel", "Visual C++", "Visual Studio 2005", "Visual Studio 2008", "Visual Studio 2010", and "Visual Basic" are products of Microsoft Corporation, U.S.A..

# Table of Contents

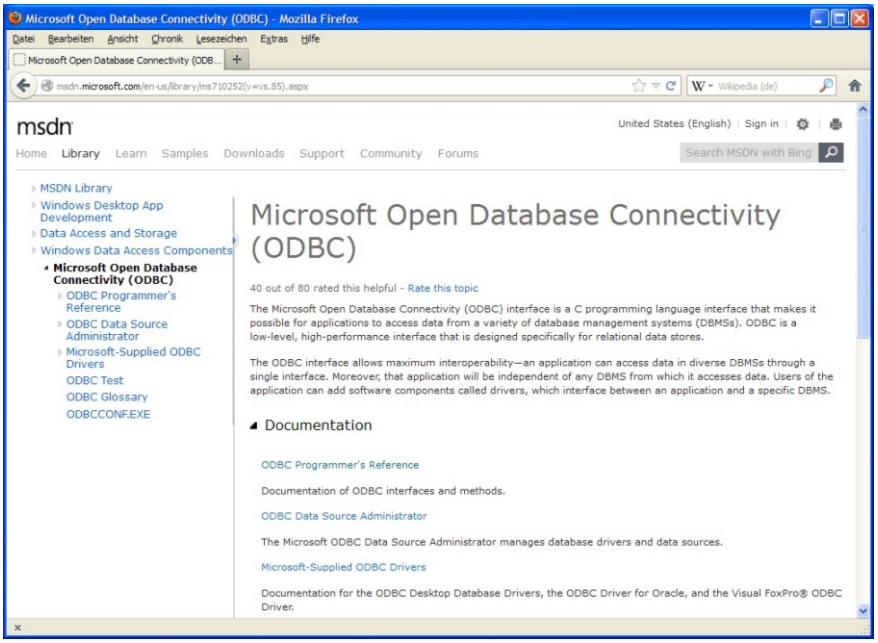
# ForDBC - Fortran Database Connectivitiy

## 1. The Use of ODBC in Fortran 90/95 Programs

With the implementation and the enhancement of the X/Open and SQL access Group specifications, Microsoft has created an interface that permits database vendors to offer programmers a standardized, open interface to their databases. This interface is named ODBC (Open Database Connectivity). It provides programmers with functions that enable them to access databases by means of standard SQL (SQL = Structured Query Language) independently of internal database record formats. Microsoft supplies ODBC with a product named "Microsoft Developer Network" (MSDN) which is the foundation of this introduction [ODBC96 and ODBC98]. At present, it is also part of the documentation of the Microsoft Visual Studio products.

This description can also be found on the Internet:

⇨ http://msdn.microsoft.com/library/




*Illus. 1: The description of the ODBC API on the internet.*

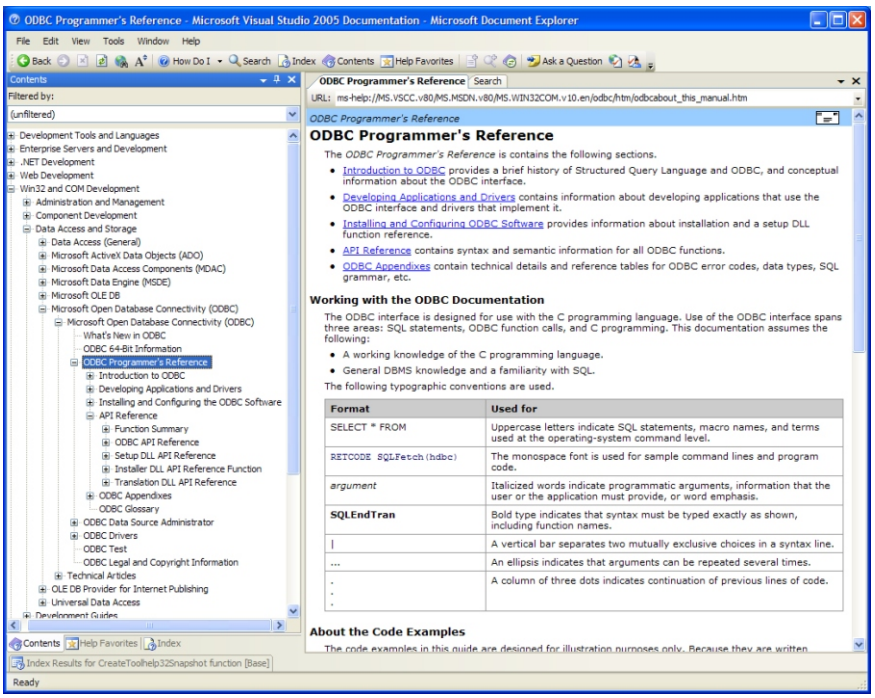
The main objectives of this introduction into ODBC programming with Fortran 90 respectively Fortran 95 (abbreviated Fortran9x in the further documentation) is

- to explain the essential functionality of the ODBC interface

and

- to guide users through their first steps creating ODBC applications.

This document is a helpful relief, since Microsoft's ODBC documentation addresses mainly C/C++ programmers. And accordingly, the specific ODBC data types, the functions and the constants in the ODBC application programming interface (ODBC API) are defined for use in C/C++ programming language only.




*Illus. 2: The description of ODBC API in the online-help of Visual Studio 2005.*

In principle, it is possible to call the ODBC functions in Fortran 77 programs too, but the implementation and the references to the original declarations in the ODBC API are much easier to carry out in Fortran 9x.

To say it again, this document is an introduction. Its use is reasonable only in conjunction with a complete description of the ODBC interface, how it is found for example in [ODBC96] or [ODBC98], and in recent versions of the Microsoft Visual Studio MSDN Online Help. This introduction attempts to convey the fundamentals for:

- the ODBC software installation, including the database definition and configuration,
- the architecture of the ODBC interface,
- and the database communication by means of ODBC (connections, transactions, ODBC-functions calls and data transfer).

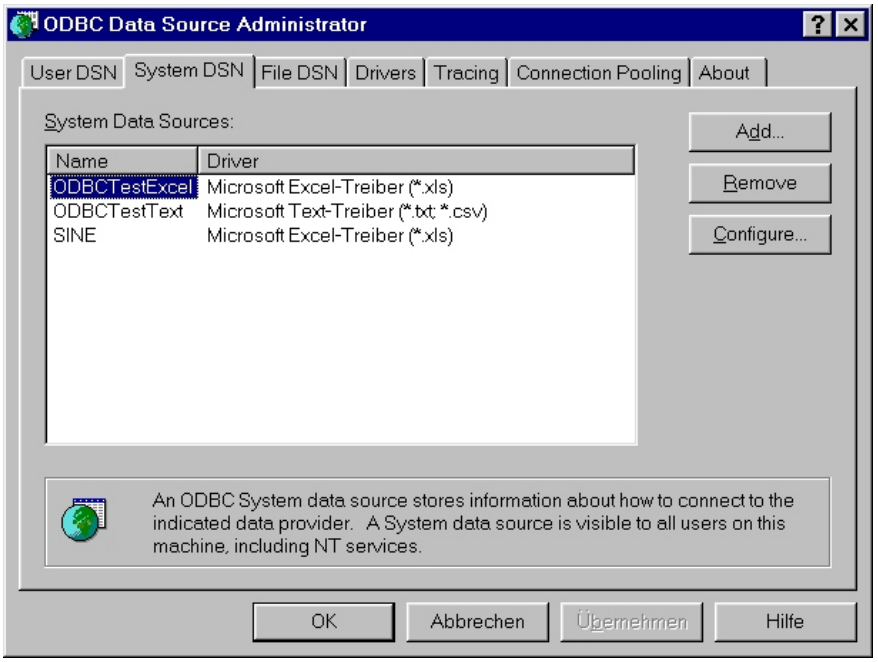
## 2. Installation of ODBC Software

The installation of ODBC software is carried out by a driver specific program which usually is provided with the database system.

For the configuration an ODBC administrator program (e.g.: ODBCAD32.EXE) or a specific set-up program is available. Microsoft's ODBC Software Development Kit contains an ODBC administrator program, which supplies you with detailed information about the driver set-up toolkit and the ODBC administration. [ODBC-I] supplies summarized information for the setup of ODBC applications.

On the target computer, on which an ODBC application under Windows 9x, 200x, NT, XP, Vista 7 etc. (simply Windows in the further documentation) shall run, it is essential that both

- the driver manager, i.e. the ODBC32.DLL as well as the component CTL3D32.DLL,




*Illus. 3: ODBC Adminstrator Program (odbcad32.exe)*

- and the driver, e.g. ODBCJT32.DLL for Excel files (.XLS), dBASE files (.DBF) etc..

are available.

Usually, if a ODBC compliant database system is installed, those components are supplied too. See section 3 for more information on the ODBC administrator program.

Microsoft provides freely available drivers for the use of MS/Access and MS/Excel, which are used in the ForDBC examples. These drivers are usually automatically installed during the installation of MS/Office which is widely used. If you don't have MS/Office, Excel or Access installed, you may download these drivers.

For 32-bit Windows applications they are found in the Microsoft Data Access Components (MDAC), which can be found in the internet at

⇨ http://www.microsoft.com/download/en/details.aspx?id=5793

(if this link is not up-to-date anymore, search for MDAC). Download MDAC (MDAC_TYP.EXE) and execute it. This will install the drivers for MS/Access and MS/Excel.

For 64-bit Windows applications the drivers can be found for example in the “Microsoft Access Database Engine 2010 Redistributable”, which is supplied at

⇨ http://www.microsoft.com/download/en/details.aspx?id=13255

via download.

## 2.1 Data Source

The term "data source" designates the entire data that should be accessed: the associated database management system (DBMS), its computer platform and, if any, the network, which permits access to the information. In order to enable access to a data source, the driver will need appropriate information to establish the connection. These are at least (in accordance with the ODBC Core level - see section "Driver")

- the name of the data source (DSN = data source name)
- a user identification number (user ID), if applicable
- a password, if applicable

ODBC extensions additionally permit to specify e.g. a network address or further passwords.The connection information for each data source is stored formerly in the ODBC.INI file or nowadays in the Windows Registry Database (registry). It is created at installation and is usually managed by an administration program (see below). A section in this ODBC.INI lists the available data sources. E.g.:

```
[ODBC 32 bit Data Sources]
dBASE-files=dBase-driver (*.dbf) (32 bit)
Excel-files=Excel-driver (*.xls) (32 bit)
Currencies=Sybase SQL Anywhere 5.0 (32 bit)
```

In the registry corresponding entries are found in the section

```
HKEY_CURRENT_USER\Software\ODBC\ODBC.INI\ODBC Data
Sources
```

or in the section

```
HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\ODBC Data
Sources
```

Further sections describe the data source in greater detail. They contain the driver name, a description (if any), and the name and path of the database file, and further details necessary for the connection setup. E.g.

```
[dBASE-files]
Driver32=C:\WINDOWS\SYSTEM\odbcjt32.dll

[Excel-files]
Driver32=C:\WINDOWS\SYSTEM\odbcjt32.dll

[Currencies]
Driver=D:\sqlany50\win\wod50w.dll
UID=dba
PWD=sql
Description=Currencies Data Source
Start=dbeng50w
DatabaseFile=Currencies.DB
DatabaseName=DBCurrencies
AutoStop=yes
TranslationName=Sybase SQL Anywhere 5.0 Transla
TranslationDLL=D:\sqlany50\win\wtr50w.dll
TranslationOption=1
Driver32=D:\sqlany50\win32\wod50t.dll
```

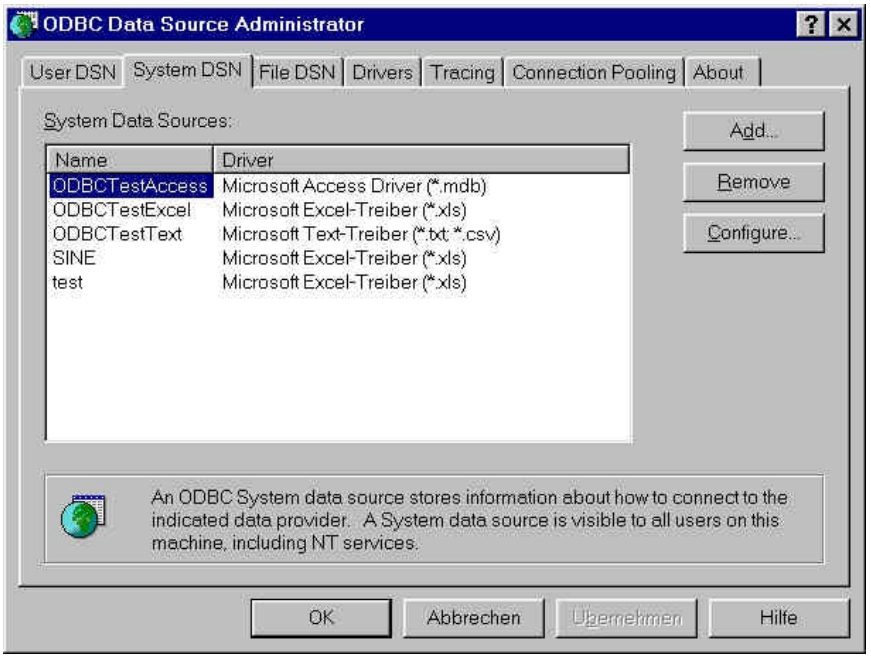
Here, in this example, "Currencies" ist the data source name of a database (filename: Currencies.DB), which is based on "Sybase SQL Anywhere 5.0 (32 bit)".

Hence, any data base for which an ODBC driver is installed can be used as a data source. In general, ODBC drivers are supplied with all well known databases, e.g. from Oracle, Sybase (SQL Anywhere and others), Informix, IBM (DB2) or Microsoft (MS/SQL). In addition, you can find ODBC drivers for MS/Access, MS/Excel, dBASE and even Text files.

# 3. Definition and Configuration of Data Sources under Windows

Before an ODBC application can access a data source, the data source has to be defined first. This can be achieved during runtime of a program (e.g. by calling the function `SQLDriverConnect`, see the sample program T_ODBCDrvConnRd.f90), or the data source is created explicitly by a utility program of the operating system.

On Windows you start the ODBC administrator program (e.g.: ODBCAD32.EXE). One can find it usually in the system control for Windows (" Start menu: Start | System Control "; in Windows XP the entry can be found under "Administration"). For 64-bit Windows two versions of the ODBC administrator program are available. One serves 32-bit drivers and data sources, the other one the 64-bit counterparts.




*Illus. 4: View of tab „System DSN“ in the ODBC Data Source Administrator Program.*

In the ODBC administrator program, you can see three tabs titled “User DSN”, “System DSN” or “File DSN” (DSN = data source name). Each of them heads a dialog which allows to select a driver type from a list and to define a DSN by pressing the key “Add”. The data source will be named, with which the ODBC manager identifies the database and its driver. The ODBC administrator program stores the generated information in the ODBC.INI file or in the registry, respectively. Data source names can be generated at user level ("User DSN"), at system level ("System DSN") and at file level ("File DSN"). This causes that those data sources can be accessed only with appropriate user rights.

## 3.1 Excel Data Source

You can read from and write to Microsoft Excel worksheets, to worksheets within a workbook (workbooks available since Excel 5.0), to arbitrary (unnamed) or specified ranges of cells (e.g. A1:C14) in a worksheet. Some particularities have to be considered when naming the data source:

- Cell range details must be separated by a comma, e.g. "C:\EXCEL\SALES.XLS, A1:C14".
- For a worksheet in an Excel 5.0 or 7.0 workbook, the worksheet should be specified by its name followed by a dollar sign ("$"). E.g. "SHEET1$". Cell ranges are indicated by appending the cell range to the worksheet name. E.g.: "SHEET1$A1:C14".
- In order to address a named range of cells in an Excel worksheet, this name must exist before opening it by your ODBC application (in Excel you name the cell range by marking the range of cells and then selecting in the menu Insert | Name | Set).
- Individual cells and records of a worksheet cannot be addressed directly.

Furthermore, special restrictions apply when using Excel worksheets:

- Multiple access is not possible (the Excel ODBC driver does not support multiple users).

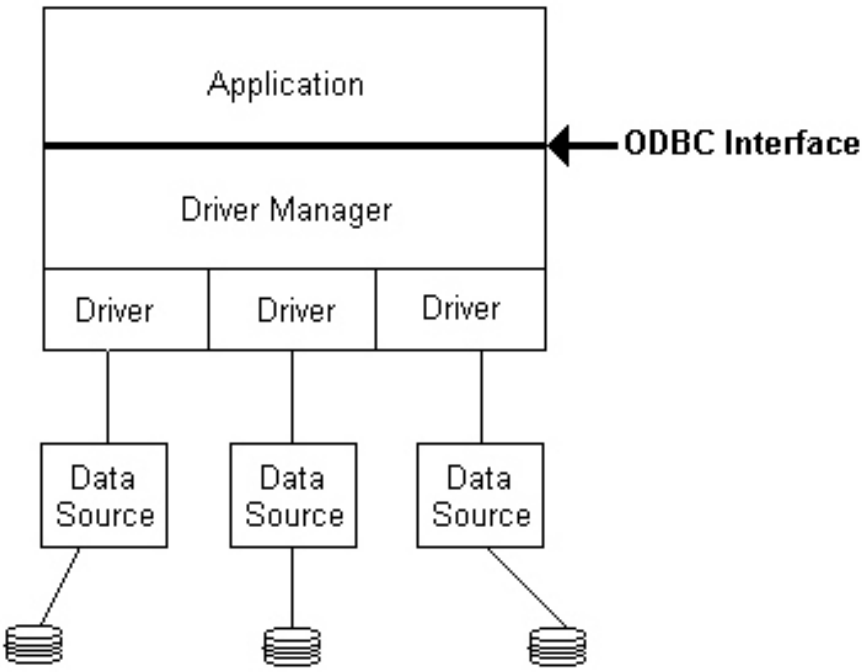
Remark: The documentation of the access on Excel tables in the ODBC API is more than poor. An so the sample programs supplied with ForDBC are more the result of trial and error.

# 4. The Structure of the ODBC Interface

The open database connectivity (ODBC) interface permits applications to access data sources of various database systems (Data Base management Systems; DBMS) using SQL (structured query language - a description can be found for example in MSDN). The advantage over reading from or writing to database files directly is that access via SQL is independent of the internal record structure of the database. And thus, you don't have to consider record formats when programming. Furthermore, using a data source name gives your application more flexibility because changing the location and the configuration of the database is possible mostly without any change of your source code.

Microsoft provides programmers with a software interface, the ODBC application programming interface (ODBC API), that consists basically of functions for:

- connecting with the database or the data source respectively
- creating and administering memory and its assignments for data communication
- accessing the data
- administering transactions
- handling of errors




*Illus. 5: Fundamental structure of an ODBC application.*

The ODBC API functions are usually provided by the driver manager (ODBC32.DLL) and its import library (ODBC32.LIB).

The database functionality itself is supplied by the data base driver which comes with your DBMS. Its functions are called by the driver manager and not by your program.

## 4.1 Driver Manager

The driver manager (driver manager) has the main tasks,

- to carry out various ODBC initializations
- to evaluate the ODBC.INI or the registry, respectively
- to load the driver(s), when the application invokes one of the ODBC Connect functions, `SQLBrowseConnect`, `SQLConnect` or `SQLDriverConnect`
- and to check whether the calls to the ODBC functions are containing valid parameters and if these are supplied in correct sequence.

## 4.2 Driver

The driver usually is a DLL which is provided by the database vendor. It contains the ODBC functions adapted to the needs of the database. The driver's main functions are

- the establishment of the connection to the data source (connect),
- the transmission of queries, data updates and inserts,
- the conversion of data into a desired format,
- the formatting of errors into the standard errors coding format,
- the declaration and the administration of the internal SQL cursor
- and the release of transactions, when the data source requires the explicite initiation of the transactions

Many of these functions are generally not visible to the application. They are called by the driver manager (ODBC API functions).

Two driver types are admitted:

- Single-tier: The driver handles both ODBC API calls and SQL commands.
- Multi-tier: The driver handles the ODBC API calls and passes the SQL commands to the data source.

From the application's point of view there is no difference between both.

## 4.3 ODBC Conformance Levels

ODBC defines conformance levels for drivers regarding the ODBC API and the SQL grammar (including the SQL data types). This way a certain standard of functionality can be granted such that a DBMS vendor is not obliged to supply the complete ODBC API and SQL-functionality if its database may not able to deliver them. The programmer can ensure the functionality of the loaded driver through the ODBC API-functions `SQLGetInfo`, `SQLGetFunctions` and `SQLGetTypeInfo`.

The ODBC API defines a set of core functions and functions of the X/Open and SQL access group call level interface specification. Those and further functions are defined in two function groups of (level 1) and (level 2). The functions of level 2 contain those of level 1. It suffices in most cases that the DBMS driver makes the functions of level 1 available. The chapter

"ForDBC Functions Overview" contains a list of all ForDBC functions and their ODBC levels.

Similarly, the functionality of the SQL grammar consists of core functions (core SQL grammar) that nearly matches with the specifications of the X/Open and SQL access Group SQL CAE of 1992. ODBC pools the SQL functionality in two further groups, the minimal-grammar (minimum SQL grammar) and the extended grammar (extended SQL grammar). The functions and the data types of the core SQL grammar suffices in many cases.

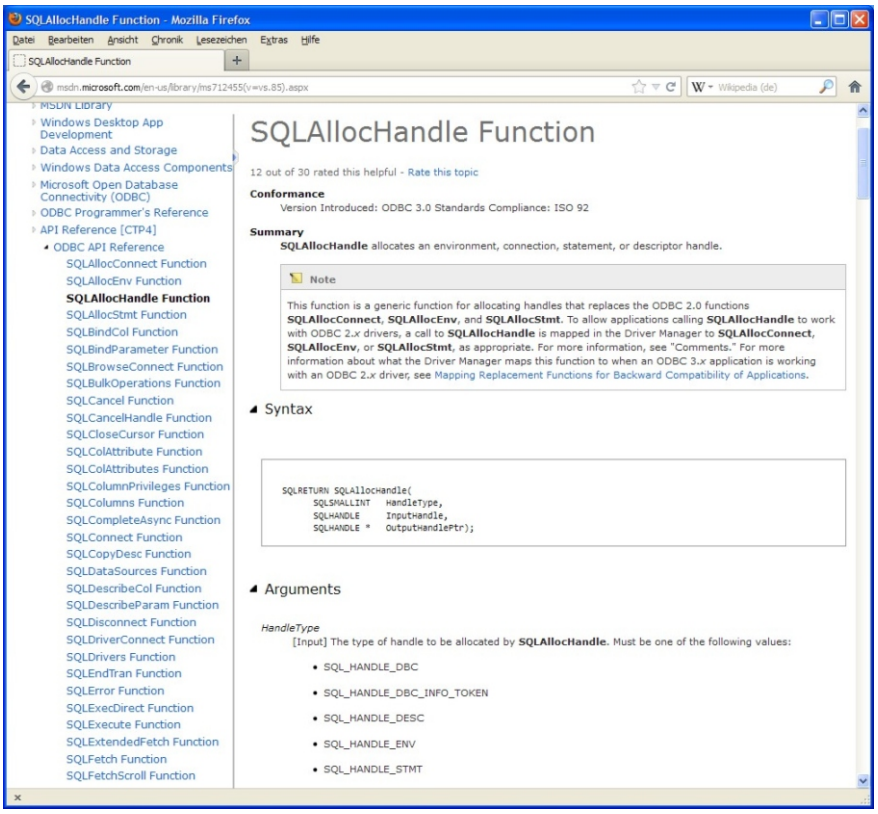
## 4.4 Connections and Transactions

Before an application can use ODBC, it has to be initialized (by calling by `SQLAllocHandle`) creating an

- **environment identification number** (environment handle; *hEnv*).

Necessary for the communication with the data source is a

- **connection identification number** (connection handle; *hDBC*).




*Illus. 6: The description of the ODBC function SQLAllocHandle*

With both handles (*hEnv* and *hDBC*) the application can access the data source. Several connections to the same data source or to others can be active at the same time. Each connection holds a transaction space of its own. Within an active connection one or more SQL statements can be executed.

The transactions for each active connection are managed by the driver (if the driver supports transactions; some don't, for example MS/Access on Windows XP). A COMMIT and a ROLLBACK can be executed either automatically (i.e. after completion of an SQL instruction; set attribute:

SQL_ATTR_AUTOCOMMIT) or explicitly by the application. After a COMMIT or a ROLLBACK, all SQL instructions are reset.

# 5. The Fundamentals to the Calling of ODBC API Functions in Fortran

**The names of all functions of the ODBC API start with "SQL"**. The definitions and the declarations of ODBC constants, types and function prototypes are to be found in the C header files SQL.H, SQLEXT.H and WINDOW.H (these are usually supplied with the C/C++ compiler system). C programs have to include these header files.

Fortran 9x programmers are provided with appropriate Fortran 9x modules which are integrated by means of the USE command:

```
USE ForDBC
```

The module is supplied by the file

```
fordbc.mod
```

The module ForDBC contains references to further modules, in detail the definition of the ODBC specific data types (KINDs) in

```
qtODBCKinds
(File qtodbckinds.mod)
```

and constants (PARAMETERs) in

```
qtODBCDefs
(File qtodbcdefs.mod)
```

The module uses basic C and Windows data types (KINDs). These are defined in the modules

```
qtCKinds
(File qtckinds.mod)
```

and

```
qtODBCCoreKinds
(File qtodbccorekinds.mod)
```

For example, the module qtCKinds defines the C data type LONG,

```
INTEGER :: LONG
PARAMETER ( LONG = SELECTED_INT_KIND(9))
! >10**9, for long integers (32-bit, signed)
```

thus being identical with a 4-bytes INTEGER (INTEGER*4). The module qtODBCCoreKinds defines the data type LP whose value is different for 32-bit and 64-bit systems (some reader may guess now why KINDs are being used).

In qtODBCKinds and qtODBC these data types are used to define ODBC specific data types and constants . E.g.:

```
INTEGER , PARAMETER :: SQLINTEGER = LONG
INTEGER , PARAMETER :: SQLHANDLE = LP
INTEGER , PARAMETER :: SQLHENV = SQLHANDLE
```

This seems confusing and doubtful since finally all types, constants, variables are mapped onto the basic data types, such as INTEGER*4, INTEGER*2 or REAL*4. So, for example, a variable of type SQLHENV is nothing else but a 4-bytes INTEGER. The reason why ODBC data types (or other data types for Windows too) have names of their own, is in the possibility of the more flexible enhancement of ODBC software. Then, at

the end a real advantage is hidden behind such hierarchically built data type declarations: when derived data types have to be modified at the arrival of a new operating system, the modification resulted from the change of the underlying definitions (for example in `qtCKinds`) easily causes a complete change of the derived data types (this case occurred for example at the change of Win16 to Win32 and from Win32 to Win64). For example, the KIND SQLHANDLE becomes a 8-bytes INTEGER in a 64-bit environment, while being a 4-bytes INTEGER in Win32.
For reference reasons to the original documentation of the ODBC interface, it is tried therefore to keep an analogy in the Fortran 9x definitions and declarations to those in C/C++. A C program statement such as follows

```
#include "SQL.H"
#include <string.h>
{
SQLHENV    henv;
SQLHDBC    hdbc;
SQLRETURN  rtc;
rtc = SQLAllocEnv(&henv);
rtc = SQLAllocConnect(henv, &hdbc);
.
}
```

is translated into Fortran then:

```
USE ForDBC
INTEGER (SQLHENV) :: hEnv
INTEGER (SQLHDBC) :: hDbc
INTEGER (SQLRETURN) :: rtc
rtc = SQLAllocEnv( env )
rtc = SQLAllocConnect( env, dbc )
.
END
```

**Due to the peculiarities of Fortran sometimes it is necessary to use ODBC function names in modified forms.** This concerns basically all ODBC functions that permit the use of different types for the same argument. For example: ForDBC defines for `SQLBindCol` the variants `SQLBindColI1`, `SQLBindColI2`, `SQLBindColI4`, `SQLBindColChar` etc.. Unfortunately due to compatibility reasons it was not possible to map these functions onto a single one (`SQLBindCol`) by means of generic interface.

Another unusual feature is to be taken into account when using the type `SQLPOINTER`, e.g.:

```
USE ForDBC
INTEGER (SQLPOINTER) :: ValPtr
INTEGER (SQLINTEGER) :: Value
  .
  .
Value = ...
ValPtr = LOC(Value)          ! LOC() returns address
rtc = SQLSetConnectAttr(dbc, SQL_ATTR_QUIET_MODE, &
                        ValPtr, 0 )
.
END
```

The example shows how the variable `lpAttr` (which is of type SQLPOINTER) obtains the memory location of the variable `Value` by usage of the function `LOC`. Then, the ODBC function `SQLSetConnectAttr` is called.

Whether a pointer or a value itself have to be passed can be seen from the description of the function.

The ODBC function interfaces are gathered in the module

```
qtODBCInterfaces
(file qtodbcinterfaces.mod)
```

The module is listed in the appendix A. It shows the available functions and the necessary data types for the arguments.

Compiler specific definitions can be found in the module

```
qtODBCCompiler
(see files qt_ODBC_compiler.mod
with compiler = CVF, DVF, FTN, IVF, LF90, LF95 etc.)
```

This means that the module name (`qtODBC_Compiler`) remains unchanged and thus you can switch compilers without having to modify the source code of your program.

The driver manager allows an application to access a ODBC driver by means of a suitable .DLL (e.g. ODBC32.DLL.). **When linking the application, the appropriate import library ODBC32.LIB is required**. It is available for both 32-bit and 64-bit systems. Both variants bear the same name! So make sure, the correct path is specified when linking.

## 5.1 Data Transfer between Application and ODBC Driver

The "transfer" of the data between an application and an ODBC driver and driver manager respectively is accomplished by passing arguments when calling the ODBC API functions. In our Fortran programs we use variables of known types like INTEGER, REAL or CHARACTER. From time to time we also have to work with pointers too. Then, we have to specify memory locations. Also, the use of CHARACTER arguments (strings) require attention, because we have to adapt to the typical C string treatment and follow certain rules. This is discussed in the following.

### 5.1.1 CHARACTER/String Arguments

Various ODBC functions expect character arguments (strings) or other values when being called, or they return them. E.g.:

```
szStmt = "SELECT str FROM table1"//CHAR(0)
iRet = SQLExecDirect( hStmt, szStmt, SQL_NTSL )
! The operational length of szStmt is determined
! by the terminating zero.
!
! Here we use the LONG version of SQL_NTS,
! because the INTERFACE of SQLExecDirect requires a
! long INTEGER (SQLINTEGER).
```

Internally the memory location of the variable is passed to the ODBC function here. In case of CHARACTER arguments, Fortran usually also passes a hidden length argument (the declared length of the variable that you can query by the `LEN()` function). However, with ODBC functions the length has to be given explicitly as you can see from the description of such a ODBC function. When specifying length values, the following rules apply:

- The length must be greater than or equal to 0. It specifies the actual number of characters (bytes) in the CHARACTER variables. If the length is given, then strings need not to be terminated by null (i.e. the last character does not have to be an ASCII 0 value, = CHAR (0)).
- You can omit the length specification, but then you have to use the constants SQL_NTS or SQL_NTSL, respectively (see example above) and you have to terminate the character strings you pass by ASCII 0

(CHAR(0)). The operational length of the string is determined internally by the driver. Remark: SQL_NTSL is the 4 bytes INTEGER variant, SQL_NTS the 2 bytes INTEGER variant. Which one to use depends on the INTERFACE of the ODBC function.

**Character strings being returned will always be zero-terminated.** There is no blank padding, as usual in Fortran!

### 5.1.2 Missing Values

By default, databases permit the identification of missing data. I.e. tables may contain cells or elements to which no value has been assigned. For this situation there isn't any equivalent in Fortran ("a variable without value does not exist"). One often handles this situation using a certain variable value that marks the condition "missing value". For example a value will be set to -999999.9 to indicate a missing value.
Since ODBC functions usually provide two arguments for specification of a table's column value, i.e. the value itself and the additional length argument, the latter is used for indicating a missing value:

- `SQL_NULL_DATA`: If the length specification has the value of the constant `SQL_NULL_DATA`, the contents of the variable intended to hold the table's column value shall be ignored. This way you either receive the information about missing data (in case of SQL SELECT) or you tell the driver that a value is missing (in case of SQL UPDATE or INSERT).

### 5.1.3 Other Notes Regarding the Contents of Arguments

- The use of ASCII 0 characters (`CHAR(0)`) within CHARACTER data has to be omitted, because ASCII 0 is used to indicate the end of a string.
- If nothing else is stipulated, it is permitted, to specify a null value (0) to pass a null pointer. In that case a possibly length argument will be ignored too. However the ForDBC Fortran INTERFACEs accept this only in certain situations (see appendix for INTERFACEs).
- If necessary, data are converted before return from an ODBC function. Then, the length of the data after the conversion is returned.
- In the case of character strings the terminating null is not counted.
- The driver will ignore a length specification in the case of an input argument (on input), when the data type respectively the data structure is identified as a firm length in C/C++ (e.g. these apply to integers, floating point numbers or structures of data). However, as said, on return, a length argument may indicate a missing value (if equals `SQL_NULL_DATA`).
- If an argument is too small (e.g. a CHARACTER variable), the driver tries to truncate the data to be returned. If this proceeds without loss of significant data, the driver returns the truncated data, and also returns the length of the not truncated data and indicates the status by a function value equal to `SQL_SUCCESS_WITH_INFO`.
- If a loss of significant data occurs, nothing will be returned by the argument. Also, no length is returned. The ODBC function will return `SQL_ERROR` (errors constants - see section "Return Values of the ODBC API Functions").

## 5.2 Data Types

Since the data types of the data source are sometimes different from those of the compiler language specification (e.g. a SQL data type MONEY may exist, but is not existing in C or in Fortran), a conversion is necessary. Hence, the driver maps specific SQL data types of the data source on to ODBC data types (these are defined in the ODBC SQL grammar). Information about these types can be queried by means of the ODBC API-functions `SQLGetTypeInfo`, `SQLColAttributes`, `SQLDescribeCol`, and `SQLDescribeParam`.

An ODBC data type (based on C data types) correlates to either SQL data type, e.g. the ODBC data type `SQL_C_FLOAT` corresponds to the SQL type FLOAT. The driver assumes that a data type of a table's column corresponds to either C data type of a variable. If the C data type of the variable mismatches with the expected one, then the correct C data type may be given by the *TargetType* argument of the ODBC functions `SQLBindCol`, `SQLGetData` or the `SQLBindParameter`. The driver performs the conversion of the C data type into the SQL type of the data source and vice versa.

## 5.3 Identification of Environment, of Connections and of Statements

The driver manager allocates memory for a single ODBC environment, for each data source connection and for each SQL statement. Each time memory has been allocated, a handle for its identification is returned.

The environment identification number - the **environment handle** - identifies an internal memory area that holds global information. It contains among others the valid and the active connection identification numbers (connection handles). It is of type `HENV` (ODBC v1.x/v2.x) or `SQLHANDLE` (ODBC v3.x), respectively (both map to the same INTEGER kind and are, therefore, identical). An application owns one environment identification number at the most, and this environment handle is required before connecting to the data source.

The memory containing information about a ODBC connection is identified by a connection identification number - the **connection handle**. It is of type `HDBC` (ODBC v1.x/v2.x) or `SQLHANDLE` (ODBC v3.x) respectively, and has to be created before the connection to the data source. An application may have several connection handles (i.e. several connections to various data sources), but just one single environment handle per application.

Memory for SQL statements is identified by an statement identification number - **statement handle**. For either SQL statement a statement handle has to be created before execution of the statement. It is of type `HSTMT` (ODBC v1.x/v2.x) or `SQLHANDLE` (ODBC v3.x), respectively. Any statement handle is associated with some specific connection handle.

## 5.4 The Return Values of the ODBC API Functions

The success, the status respectively the warning or the failure of a ODBC API function is returned by its function value. The following constants are common return values. The constants are defined in the module `qtODBCDefs`.

SQL_SUCCESS

SQL_INVALID_HANDLE

SQL_SUCCESS_WITH_INFO

SQL_STILL_EXECUTING

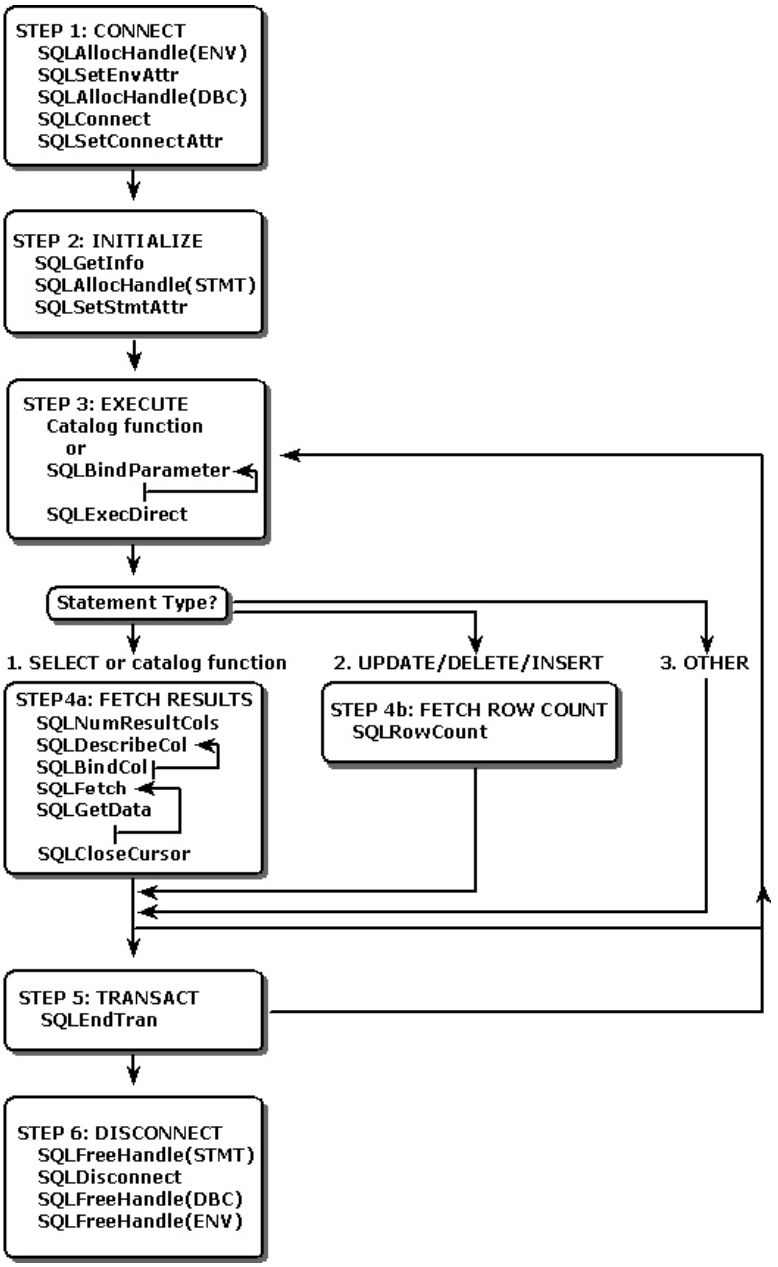
SQL_NO_DATA_FOUND

SQL_NEED_DATA

SQL_ERROR

If SQL_SUCCESS_WITH_INFO or SQL_ERROR are returned, the ODBC functions SQLError (ODBC v1.x/v2.x) and SQLGetDiagRec (ODBC v3.x) supply additional information about the error.

## 5.5 Data Source Access - Basic ODBC Application Structure




*Illus. 7: ODBC application structure (ODBC v3.x), according to [ODBC08]*

In order to access a data source, the following steps are necessary:

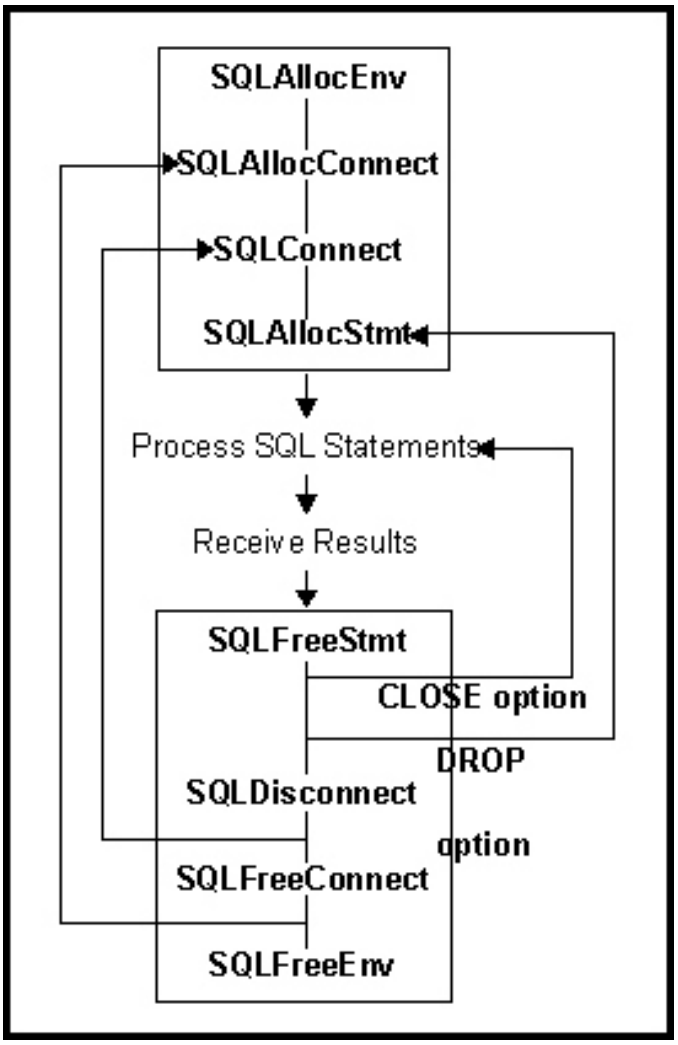
1. Establish the connection to the data source by creating an ODBC environment and connecting to the data source.

2. Execute SQL statements:
The SQL command is placed in plain text in a CHARACTER variable (a string) and passed to the appropriate ODBC function.
If this causes a result set to be generated (for example when executing a SELECT command), a SQL cursor has to be set up. This is usually accomplished automatically when a table's columns are bound to local variables. This column binding allows to fetch records successively of the result set.
In case of an error the driver queries the error information and it is possible to take action to cater the situation (for example to issue a ROLLBACK).

3. Every transaction must end with a COMMIT or ROLLBACK provided that the database connection has not been opened in AUTOCOMMIT mode.

4. When the interaction with the data source shall come to an end, the connection itself must be terminatedl.

The diagram above shows the ODBC API functions for allocating the environment, the connection to the data source (connect), for the execution of SQL statements (process) and for the termination of the connection (disconnect). The following diagram shows the structure using functions of ODBC v1.x/v2.x only.




*Illus. 8: ODBC application structure (ODBC v1.x/v2.x)*

### 5.5.1 The Initialization of the ODBC Environment

The first step of an ODBC application is the initialization of the ODBC environment by creating the environment identification number - the **environment handle**. After variables have been declared

```
INTEGER (KIND=SQLHANDLE) :: env = SQL_NULL_HANDLE
rtc = SQLAllocHandle(SQL_HANDLE_ENV,   &
                     SQL_NULL_HANDLE, env )
```

the ODBC environment is created. SQL_HANDLE_ENV is a constant that controls which type of handle is to be created by

SQLAllocHandle. If successful (rtc = SQL_SUCCESS), the function returns the environment handle in the argument env.

Note: Only a single ODBC environment handle should be open in an application at any one time.

## 5.5.2 Connecting to a Data Source

After the ODBC environment have been initialized, a connection to a data source can be made. The declaration necessary for the connection identification number - the **connection handle** - follows:

```
INTEGER (SQLHANDLE) :: dbc = SQL_NULL_HANDLE
```

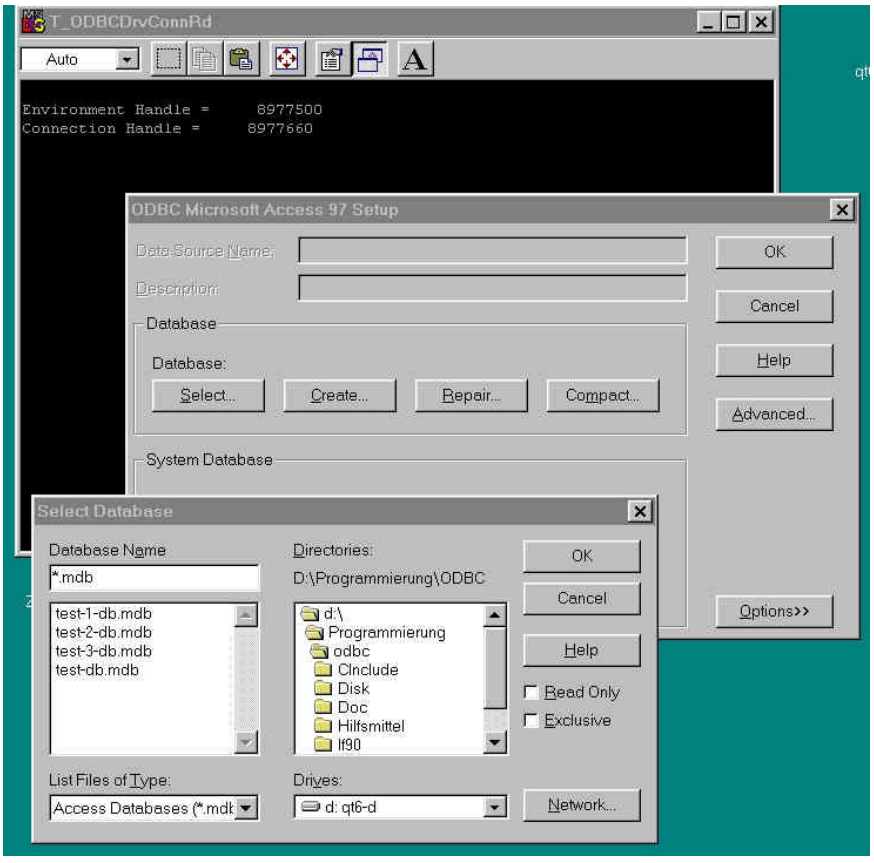
The connection is made using the function SQLAllocConnect which will need as its first argument the environment handle created before.

```
rtc = SQLAllocHandle( SQL_HANDLE_DBC, env, dbc )
```

If no error occurred (rtc = SQL_SUCCESS), the function returns the connection handle in the last argument (dbc).

The connection to the database finally follows calling the function SQLConnect. It requires the connection handle obtained before and the specification of the data source name, a user name (or user id) and a password (authentication). E.g.:

```
rtc = SQLConnect(dbc,  &
                 'Currencies'//CHAR(0), SQL_NTS, &
                 'dba'//CHAR(0), SQL_NTS, &
                 'sql'//CHAR(0), SQL_NTS )
```




*Illus.9: Selecting the data source during runtime - cf. sample program T_ODBCDrvConnRd.f90.*

The data source in the above example is named “Currencies”, the user identification (Login ID) is “dba” and the password is “sql”. All strings are null terminated (`CHAR(0)` appended) and instead of specifying the string length, `SQL_NTS` is provided as an argument (`SQL_NTS` indicates a null terminated string).

When `SQLConnect` is processed, the driver manager searches for the data source name (DSN) in the registry. When it finds the DSN, it obtains information about the database driver, the path and name of the database file and various options for loading both the driver DLL and the database file. If this is successful, a connection to the data source has been established.

There are further possibilities to connect to a data source. In particular those which allow to supply the path of the database file, the driver etc. (see `SQLDriverConnect` and sample program T_ODBCDrvConnRd.f90).

### 5.5.3 The Execution of SQL Commands

All kinds of SQL commands can be executed that are supported by the database driver. The syntax being used should comply to the standard definitions of ODBC (SQL grammar). The SQL command is converted internally by the driver into the native database syntax.

It is distinguished between

- a single execution of a command (**direct execution**)

and

- multiple or repeated execution of the same command (**prepared execution**).

Direct execution is performed by the function **`SQLExecDirect`**. The command is executed once. The result of that execution is called result set and its extent is usually not known before execution (e.g. SELECT).

The prepared execution by means of **`SQLPrepare`** and of the succeeding **`SQLExecute`** will be used in the case, when a command has to be executed repeatedly (e.g. INSERT, UPDATE).

In general, a prepared command runs faster than a direct one, because for each SQL command an “access plan” has to be set up internally.

Before the execution of a SQL command, memory must be allocated internally which is identified by a statement identification number - the **statement handle**. For example the statement identification number is of type SQLHANDLE and can be declared as follows:

```
INTEGER (KIND=HSTMT) :: stmt = SQL_NULL_HANDLE
rtc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, stmt )
```

The statement handle is returned by the function `SQLAllocHandle` which requires the connection handle (`dbc`) created before.

Before the execution of a SQL command, attributes, values and command parameters can be set (see section “Parameter Binding”).

Finally, in the case of direct execution, the SQL command is performed by means of `SQLExecDirect`. E.g.:

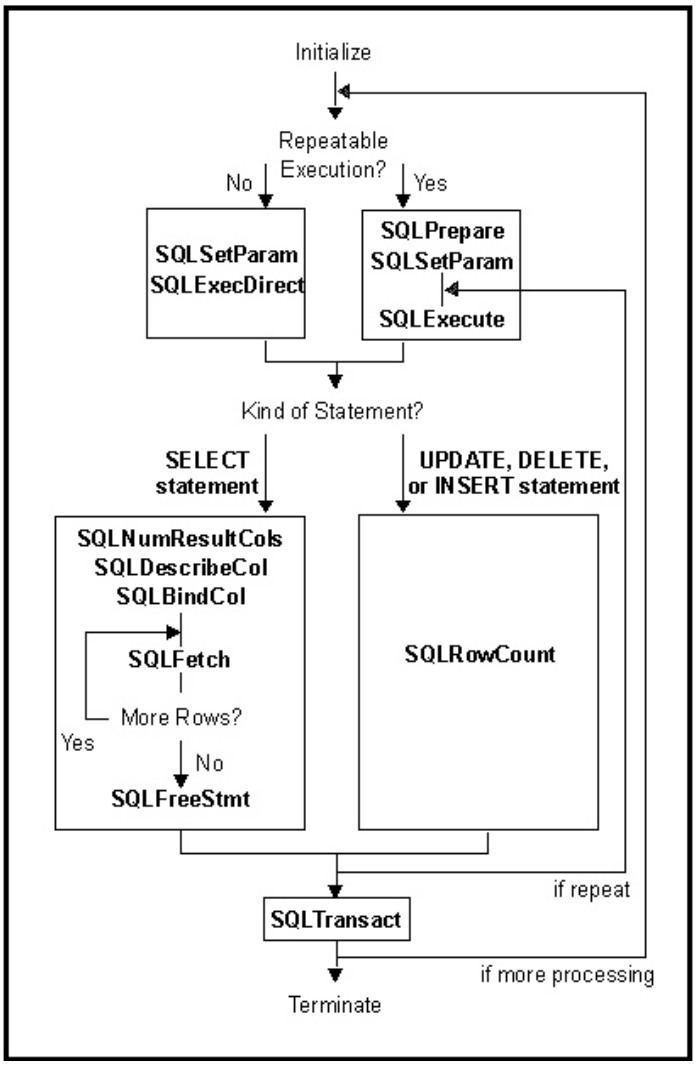
```
rtc = SQLExecDirect( stmt,   &
     "DELETE FROM Currencies WHERE Currency = 'US$'" &
     //CHAR(0), SQL_NTSL )
```

Explanation: The statement handle (stmt) created before is used to execute the SQL statement “DELETE FROM…”. The SQL command causes all entries of table "Currencies" to be deleted where the Currency equals 'US$'. The SQL statement is null terminated. A string length argument SQL_NTSL is supplied which causes the length of the statement being determined internally. This SQL command does not create a result set and does not need to be prepared, thus a direct execution is appropriate.

In the case of prepared execution, the function `SQLPrepare` is used analogously. I.e. the function obtains the same parameters as `SQLExecDirect`. However the command is not executed directly thereafter. Usually after calling `SQLPrepare`, parameters of the SQL command (for example a table's columns) are bound to local variables (see section "Parameter Binding"). Calling `SQLExecute` finally executes the prepared command. E.g.:

```
rtc = SQLExecute( stmt )
```

The following diagram (with ODBC v2 functions) shows a simple flow of a program invoking ODBC functions to run some SQL commands.




*Illus. 10: Program structure to execute SQL commands via ODBC*

We should notice that commands can be executed only once by `SQLExecDirect`, and several times after preparation by means of `SQLPrepare` and `SQLExecute`. `SQLTransact` is used to perform a COMMIT or a ROLLBACK.

### 5.5.4 Parameter Binding

A SQL command can contain dummy variables for parameter values (parameter markers). E.g.:

```
INSERT INTO addressbook (forename, surname, phone)
VALUES (?, ?, ?)
```

The driver recognizes in these dummy parameters that they have to be replaced by values during runtime. Dummy parameters are used in a prepared statement (`SQLPrepare`). At repeated execution (`SQLExecute`) those dummy parameters are replaced by actual values (in the example above those parameter markers are replaced by entries into the addressbook).

Before a parameter value can be input, a dummy variable (i.e. a Fortran variable), or more precisely a memory location must be assigned for it by means of the function `SQLBindParameter`. This is called "parameter binding". `SQLBindParameter` additionally specifies the data type of the dummy variable, the precision, the length (in bytes) and if applicable, its decimal range, and so associates the table's column (the parameter) with the dummy variable. Thereafter, the parameter can be set by assigning the desired value to the dummy variable and executing the statement (`SQLExecute`). E.g.:

```
CHARACTER(30) szFName
INTEGER (KIND=SQLUINTEGER) :: ColSize = 30
INTEGER (KIND=SQLUINTEGER) :: iDecDigits = 0
INTEGER (KIND=SQLINTEGER) :: cbFName, iBufLen = 0
.
rtc = SQLPrepare(stmt,    &
       "INSERT INTO addressbook (forename, surname,  &
              phone) VALUES (?, ?, ?)"//CHAR(0),     &
              SQL_NTS )
rtc = SQLBindParameter( stmt, 1, SQL_PARAM_INPUT,    &
                        SQL_C_CHAR, SQL_CHAR,        &
                        ColSize, iDecDigits,         &
                        szFName, iBufLen, cbFName )
...
szFName= 'John'//CHAR(0);  cbFName = SQLNTSL;
rtc = SQLExecute( stmt )
! inserts the record ("'John',...")
```

Explanation: The SQL statement is prepared and a statement handle (`stmt`) is obtained. With this, the first parameter (second argument equals 1) is bound. The parameter is intended to be input (`SQL_PARAM_INPUT`). The parameter is of type `SQL_CHAR` and the bound dummy variable `szFName` is of type `SQL_C_CHAR`. The size of the column (the parameter marker) is `ColSize`. Before executing the statement (`SQLExecute`) the value has to be put into `szFName` and its actual length has to be specified in `cbFName`. Since the parameter is input, the specification of the size of the variable `szFName` can be omitted (`iBufLen = 0`).

There is no need that the data type of the dummy variable coincides with the type of the table's column. For example, one can use the converting function of the driver (if provided), in order to convert a value in the table stored as an integer (`SQL_INTEGER`) into a value of character type (`SQL_C_CHAR`).

The assignment of a dummy variable to the ODBC/SQL input parameter remain active until it is released by a call of the function `SQLFreeHandle`. During runtime the assignment of a dummy variable to the ODBC/SQL parameter can be changed freely by another call of `SQLBindParameter`.

### 5.5.5 Transactions

ODBC and SQL know two COMMIT modes:

- The **auto commit mode** (`SQL_AUTO_COMMIT`) performs a transaction (COMMIT, ROLLBACK) automatically after the execution of any SQL statement.
- In **manual commit mode** the programmer is responsible to issue a COMMIT or ROLLBACK. The transaction is executed by calling `SQLTransact` or `SQLEndTran` respectively, which then may include one or several SQL commands being applied at one time.

If a driver supports `SQL_AUTO_COMMIT` (or `SQL_ATTR_AUTOCOMMIT`), this is the default transaction mode. Otherwise, the manual commit mode is the default. By means of the function `SQLSetConnectOption` or `SQLSetConnectAttr` respectively, it is possible to change the mode.

It might be important to know, that after a transaction the internal SQL cursor and the internal "access plans" might be lost. To obtain for information call the function `SQLGetInfo` with `SQL_CURSOR_COMMIT_BEHAVIOR` and `SQL_CURSOR_ROLLBACK_BEHAVIOR`.

### 5.5.6 Retrieving Result Sets

SQL commands can be subdivided into those

- which generate and return result sets (e.g., SELECT)

and those

- which don't. But they perform changes on the data source. For example: DELETE, UPDATE, INSERT, GRANT and REVOKE alter a database.

If a DELETE, UPDATE, or INSERT have been successful, this can be checked by either the return code of the executed function or by calling the function `SQLRowCount`.

If a result set is generated, its contents depend on the SQL command being issued. E.g.: a "SELECT * FROM adressbook" returns a result set that contains all records of that table. It might be that both the number of columns of that table and their types are unknown. Then, there are ODBC functions to obtain this information.
In most cases, the programmer knows how the result set will look like. To obtain the result set, either call `SQLBindCol` (ODBC v1.0) or `SQLBindParameter` (since ODBC v2.0), respectively, to bind Fortran variables to columns of the result set. This works as described in the chapter "Parameter Binding".
`SQLBindCol` and `SQLBindParameter` require to specify

- the data type (conformant to C) into which the result is to be converted (if it has to)
- an output buffer of sufficient size (this usually is a local variable)
- the length of the output buffer, provided that the variable being used does not have a pre-defined fixed length (for example INTEGER, REAL have a fixed length)
- a variable (or memory location) in which the length value (in bytes) can be returned.

Example:

```
CHARACTER(21) wName
INTEGER (SQLINTEGER) :: LENwName = 21
INTEGER (SQLINTEGER) :: cbwName
.
rtc = SQLExecDirect( stmt,    &
     "SELECT currencyname FROM currencies"//CHAR(0), &
     SQL_NTSL )
rtc = SQLBindCol( stmt, 1, SQL_C_CHAR, wName,  &
                  LENwName, cbwName )
```

Explanation: The first column of the SELECT command (second argument of `SQLBindCol` equals 1) gets linked to the memory location of the variable `wName` which is of type `SQL_C_CHAR`. Its buffer length is `LENwName`. If the SELECT command is executed successfully (calling `SQLFetch`), the result will be stored in `wName` and its length in `cbwName`. Since ODBC 2.0, the function `SQLBindParameter` can alternatively be used.

```
rtc = SQLBindParameter( stmt, 1, SQL_PARAM_OUTPUT,   &
                SQL_C_CHAR, SQL_CHAR, LENwName-1, 0, &
                wName, LENwName, cbwName )
```

If the bound column value equals NULL (is unset), the value `SQL_NULL_DATA` ("missing value") is returned in the length argument (`cbwName`).

If the result characteristics of a SQL statement are unknown, then the function

- `SQLNumResultCols` supplies the number of columns in the result set

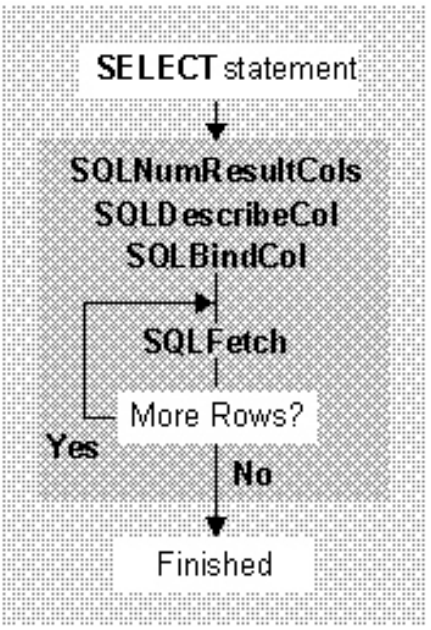
and

- `SQLColAttributes` (ODBC v1.x/2.x), `SQLColAttribute` (ODBC v3.x), and `SQLDescribeCol` return information about the columns of the result set.

These functions can be called after a SQL statement has been prepared or executed.

As soon as the binding between the columns of the result set and the variables of your program has been created (via `SQLBindCol` or `SQLBindParameter` ,respectively), the function `SQLFetch` can be called to step through the result set record by record and to obtain the column values.

The following diagram shows the course of collecting the results.




*Illus. 11: Retrieving results*

Example:

```
rtc = SQLExecDirect( stmt,   &
     "SELECT currencyname FROM currencies"//CHAR(0),  &
     SQL_NTSL )
rtc = SQLBindCol(  stmt, 1, SQL_C_CHAR,   &
                   wName, LENwName, cbwName )
DO
   rtc = SQLFetch( stmt )
   IF ( rtc == SQL_NO_DATA_FOUND ) EXIT
   PRINT*, 'Fetch: ', wName(1:cbwName)
END DO
```

Explanation: The SELECTcommand shall collect all currency names in the first and here unique column "currencyname" of the table "currencies". After the command has been executed, the variable `wName` is bound to the column "currencyname". In the DO loop the `SQLFetch` function causes that the value of the column "currencyname" in the result set and its length are placed in the variables `wName` and `cbwName`, respectively. This is repeated until `SQLFetch` returns `SQL_NO_DATA_FOUND`.

If a column value equals NULL (which means that it is unset), then no value is transferred to the bound variable (i.e. the variable does not change its value). The length specification (`cbwName`) however contains the value `SQL_NULL_DATA`.

Internally, the driver uses a cursor which is incremented when `SQLFetch` is called.

ODBC offers another function to obtain a result set which might be more appropriate if bulk of data shall be obtained (which might be much faster than repeated calls of `SQLFetch`): `SQLFetchScroll`.

Important note: When optimizing, some compilers may change the execution behavior of loops, in particular how the fetch loop is executed (cf. the example above with the DO loop). Since the bound variables change their values due to the ODBC driver and NOT by explicite assignment (as common in Fortran), the optimizer might relocate parts of the fetch loop outside of it because it recognizes erroneously that the bound variables are not changed within the loop (but in fact, they are changed!). Thus, TURN OFF THE OPTIMIZER to make sure that this effect does not take place.

### 5.5.7 Information about Status and Errors

ODBC defines return codes and a protocol for error handling. The latter specifies the way how the components (e.g. driver, driver manager) of an ODBC connection generate error messages and how the function `SQLError` (ODBC v1.x/2nd x) or `SQLGetDiagRec` (ODBC v3.x), respectively return these. The error protocol includes

- the SQL state
- a driver specific error code (the "native error")
- an error message

A return value indicates whether an ODBC-function was executed successfully, whether it was partly successful (and a warning is to be taken into account) or if it failed. Return values can be:

- `SQL_SUCCESS`: The function was executed successfully and completely. Further information is not available.
- `SQL_SUCCESS_WITH_INFO`: The function was executed successfully though a non-fatal error occurred. Further information can be obtained by means of the functions `SQLError` or `SQLGetDiagRec`.
- `SQL_NO_DATA_FOUND`: The entire result set has been collected and no more data is available, or a result set hasn't existed at all.
- `SQL_ERROR`: The function failed. `SQLError` and `SQLGetDiagRec` provide further information.
- `SQL_INVALID_HANDLE`: An invalid handle was specified (either faulty environment, connection, or statement handle). `SQLError` and `SQLGetDiagRec` do not provide further information.
- `SQL_STILL_EXECUTING`: A function is running and not completed yet.
- `SQL_NEED_DATA`: While a command is being performed, the driver requests more data (for example, a value for a parameter is missing).

Dependent on the return value, it is your program's task to react properly and to manage the fault situation. Sometimes, it is necessary in the error case to repeat the call of `SQLError` or `SQLGetDiagRec`, respectively, to fetch all error messages. If thereafter another ODBC function is called, the pending error information might be lost.

Further information and specifications regarding to ODBC error messages can be found in [ODBC E].

### 5.5.8 Cancelling of Asynchronous Functions

Functions running asynchronously can be cancelled by a call of the function `SQLCancel` . However when the cancellation happens is dependent on database driver. Thereafter, the same asynchronous function can be called again. If `SQL_STILL_EXECUTING` is returned, the cancellation was not successful yet. If the cancellation was successful, SQL_ERROR and SQLSTATE S1008 (= operation cancelled) will be returned.

### 5.5.9 Terminating a Connection

To release the resources (e.g. memory) that were created in an ODBC application, the elder (and deprecated) functions `SQLFreeStmt`, `SQLFreeConnect` and `SQLFreeEnv`, or better, `SQLFreeHandle` (since ODBC v3.0) have to be called.

`SQLFreeStmt` (ODBC v1/v2) releases the resources of a statement identification number (statement handle). The function has four options:

- `SQL_CLOSE`: closes a cursor - provided that this one existed and rejects remaining results.
- `SQL_DROP`: includes the functionality of `SQL_CLOSE` and moreover releases all resources associated with the statement handle.
- `SQL_UNBIND`: releases all output buffers that were bound by `SQLBindCol` for the specified statement handle
- `SQL_RESET_PARAMS`: releases all parameter buffers that were bound by `SQLBindParameter` for the specified statement handle.

`SQLFreeHandle( SQL_HANDLE_STMT, ... )` (ODBC v3) gathers the diverse functions of `SQLFreeStmt` in a single call (the usage of `SQLFreeStmt` is obsolete then).

After release of the statement handle(s), a connection can be disconnected by the function

- `SQLDisconnect`.

Then follows the call of the function

- `SQLFreeConnect` (ODBC v1/v2) or the newer function `SQLFreeHandle( SQL_HANDLE_DBC, ... )` (ODBC v3), respectively, which will release the resources of the connection identified by the connection handle.

At last, the call of the function

- `SQLFreeEnv` (ODBC v1/v2) or `SQLFreeHandle( SQL_HANDLE_ENV, ... )` (ODBC v3) release the ODBC environment identified by the environment handle, respectively.

## 5.6 Particularities concerning Excel Data Source Access

Special considerations should be taken into account when using Excel worksheets as data sources:

- The column names are given by those names found in the first row of a worksheet.
- Rows cannot be deleted.
- Contents of single cells can be deleted, with the exception of cells containing formulas. The latter cannot be modified.
- Indexing cannot be carried out.
- Multiple access by several users is not possible (the Excel ODBC driver does not support multiple access).
- Data that are encoded within Excel, cannot be read.

There may be more limitations, but more information cannot be given, due to the lack of sufficient documentation. See the sample program T_ODBCExcel.f90.

# 6. Installation of ForDBC

ForDBC is either delivered on a CD-ROM, or via email, or by download in compressed form (ZIP format), or in a self-extracting archive (.exe).

### Installation from CD-ROM

In the CD-ROM root directory is complete ForDBC directory which shall be copied onto your harddisk. ForDBC is ready to be used, then.

### Installation using a compressed or self-extracting Archive

If you have received ForDBC in compressed form (ZIP) or as a self-extracting archive, simply unpack it into a directory of your choice. ForDBC can be used immediately.

## 6.1 ForDBC Software and Test Programs

To use ForDBC quickly and efficiently, get acquainted with the contents of the ForDBC directory and check out the test programs.

### 6.1.1 ForDBC Modules

ForDBC consists of compiled Fortran 90 modules, for example

```
fordbc.mod
qtckinds.mod
qtodbc_compiler.mod
qtodbccorekinds.mod
qtodbcdefs.mod
qtodbcinterfaces.mod
qtodbckinds.mod
```

which can be found in the compiler-spezific directories

...\ForDBC\Bindings\CVF

or

...\ForDBC\Bindings\IVF\32Bit

or

...\ForDBC\Bindings\IVF\64Bit

respectively ("..." abbreviates the superior directory that you have chosen to create the ForDBC directory in). The acronyms CVF and IVF stand for:
CVF = Compaq Visual Fortran
IVF = Intel Visual Fortran

For IVF two variants are supplied, one for 32-bit applications, the other for 64-bit applications.

**Your compiler must have access to these module files while compiling your ForDBC based application.** Both CVF and IVF allow to specify the module path (see chapter "Notes specific to Compilers").

### 6.1.2 ForDBC Sample Programs

The sample projects in the CVF workspace

...\ForDBC\Examples\CVF\ForDBCExamples.dsw

and in the Visual Studio solution, respectively,

...\ForDBC\Examples\IVF\IVF_VS08\ForDBCExamples.sln

demonstrate the settings needed for the Fortran compiler (CVF: "Settings", IVF: "Properties").

These projects contain test programs which use the modules mentioned above. These test programs are very helpful to learn how to program an ODBC application in Fortran.

| | |
|---|---|
| T_ODBCDrivers.f90 | lists ODBC drivers installed on your PC |
| T_ODBCDataSources.f90 | lists data sources on your computer |
| T_ODBCTestAccessInfo.f90 | provides information about the data source test-db.mdb |
| T_ODBCAccessGetRows.f90 | gets the number of rows in table "Tabelle" of data source test-db.mdb |
| T_ODBCAccessRd.f90 | reads data source test-db.mdb |
| T_ODBCExcelGetRows.f90 | gets the number of rows in sheet"Table_1" of data source ODBCTestExcel.xls |
| T_ODBCDrvConnRd.f90 | reads MS/Access and MS/Excel files (test-db.mdb and TestODBCDrvConnRd.xls, to be selected during runtime) |
| T_ODBCTestAccessWr.f90 | writes to data source test-db.mdb |
| T_ODBCExcelRd.f90 | reads data source ODBCTestExcel.xls (does also display information about the table, the columns, the names etc.) |
| T_ODBCExcelWr.f90 | writes to file TestODBCExcelWr.xls (the file has to be choosen at runtime) |

These test programs are contained in the directory

...\ForDBC\Examples

It is recommended to use the debugger when working with these samples. This definitely helps to understand the process and to see the relation to the "theory" presented before. The list above is sorted by a didactical point of view, such that you work through it from top to down. If you do not intend to access Excel via ODBC, it is sufficient to understand the examples that use the Access database.
Later, using relational databases of other vendors, you will see, that your programming will be quite similar to what you have learned in the examples. Most difficulties arise when trying to connect to a database. Here it is helpful to read the comments in the sample programs, especially in T_ODBCAccessGetRows.f90. If you encounter problems with column- or parameter binding, catalog information (see `SQLColumns`) will help to find out, which tables, column types etc. you have to handle with (see T_ODBCAccessWr.f90).

### 6.1.3 ODBC32.LIB and compiler specific ForDBC Library

The ForDBC test programs have to be compiled and to be linked either with the **ODBC32.DLL** or with the import library **ODBC32.LIB**, respectively (this depends on the Fortran compiler & linker you are using) and with an additional compiler specific ForDBC library.

The ODBC32.LIB is not part of the ForDBC delivery, but is a systems library supplied by Microsoft. Then, it should be sufficient just to specify its file name to the linker (without path). If you want to add the ODBC32.LIB explicitely to your CVF or IVF project, you should find it in the directory

C:\Program Files\Microsoft Visual Studio\VC98\LIB

Owners of Visual Studio 2008 and IVF will probably find it in

C:\Program Files(x86)\Microsoft SDKs\Windows\v7.0A\Lib

C:\Program Files(x86)\Microsoft SDKs\Windows\v7.0A\Lib\x64

If not, the file finding service of the Windows File Explorer can be of help to search for the file ODBC32.LIB on your PC.

### 6.1.4 Creating the Test Programs (.exe) in the Development Environment

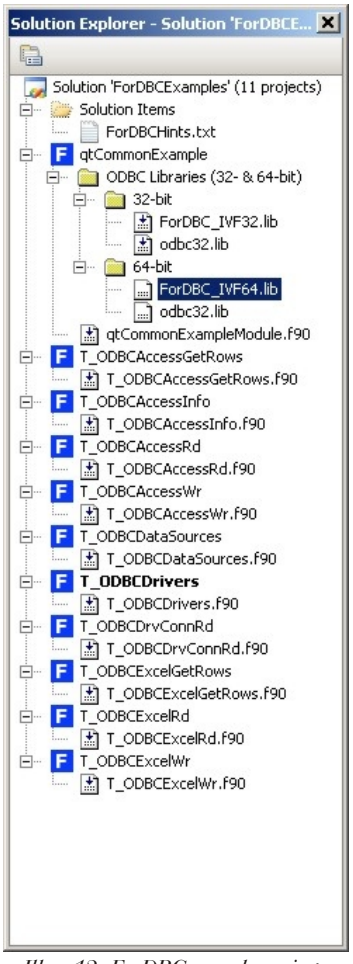
If you compile and link the test programs in Developer Studio (CVF) oder Visual Studio, you will find a file named

`ForDBCHints.txt`

in the directory

...\ForDBC\Examples

containing instructions on the settings for the compiler being used.




*Illus. 12: ForDBC sample projects in Visual Studio*

#### With IVF

For IVF, please load into Visual Studio the file

`ForDBCExamples.sln`

(for Visual Studio 2008), which is located in the subdirectory

...\Examples\IVF\IVF_VS08

Then, perform "Build Solution" to build all of the projects in the "Solution".

#### With CVF

For CVF a workspace

`ForDBCExamples.dsw`

is supplied in the subdirectory

...\Examples\CVF

which bundles all sample projects (.dsp files). A "Batch Build" in Developer Studio generates all of the test programs.

Details regarding compiler and linker settings are supplied in the chapter "Notes specific to Compilers".

### Data Sources

Some of the ForDBC test programs do work correctly only if having set up the test data source names (DSNs) before. Please read the chapter "Set-up of Data Sources for Testing".

## 6.1.5 Addendum

The ForDBC directory also contains a file named

```
Addendum.txt
```

which provide the most recent information not being convered here.

# 6.2 Set-up of Data Sources for Testing

Most of the test ODBC applications (.exe) that you have created by now are only operational when the data sources they use have been set-up properly. The test programs use the following files:

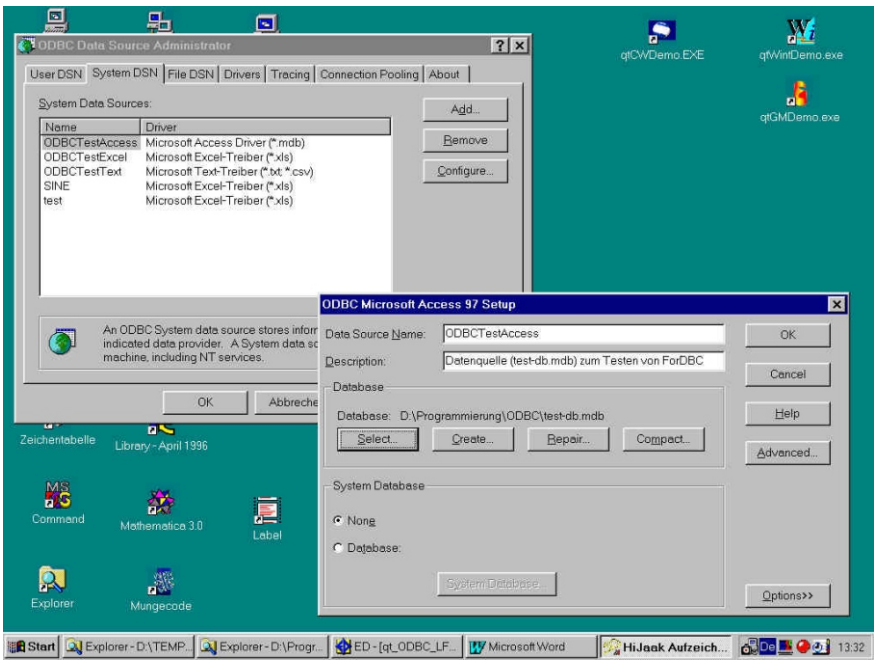
ODBCTest.xls [Excel 95 / 7.0 Worksheet]

test-db.mdb [MS Access Database]

To create the data sources, start the ODBC administrator program (see chapter "Definition and Configuration of Data Sources under Windows") and enter the following data source names (DSNs) and select the appropriate driver:

ODBCTestExcel [for the file ODBCTest.xls with Microsoft Excel driver]

ODBCTestAccess [for the file test-db.mdb with Microsoft Access driver]




*Illus. 13: Set-up a data source by means of the ODBC administrator program.*

Then, the test programs can run successfully.
Please note that on 64-bit Windows the ODBC Administrator program is available in two variants. In this situation the data sources have to be created in both variants using the appropriate drivers.

For the other two Excel files that are supplied,

TestODBCDrvConnRd.xls

and

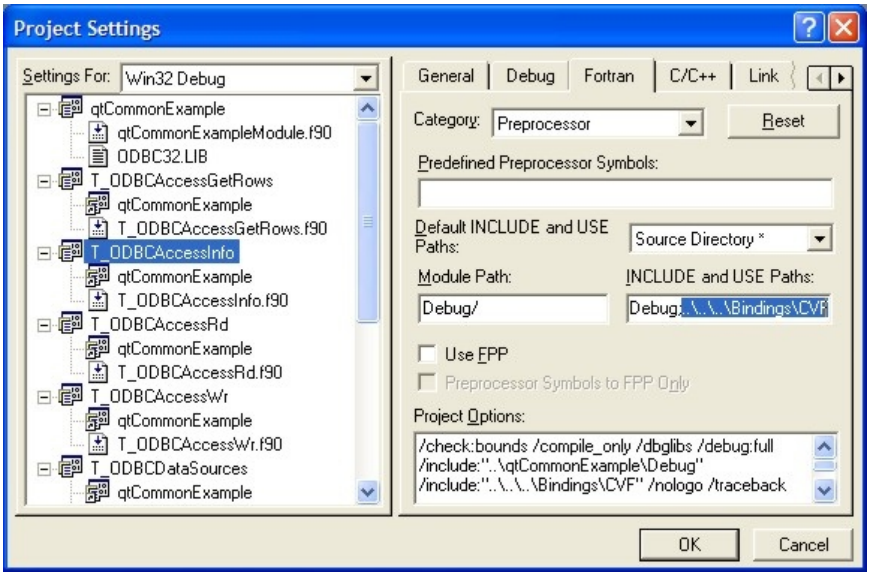
TestODBCExcelWr.xls ,

it is not necessary to set-up data sources.

## 6.3 Notes specific to Compilers

### 6.3.1 Compaq Visual Fortran

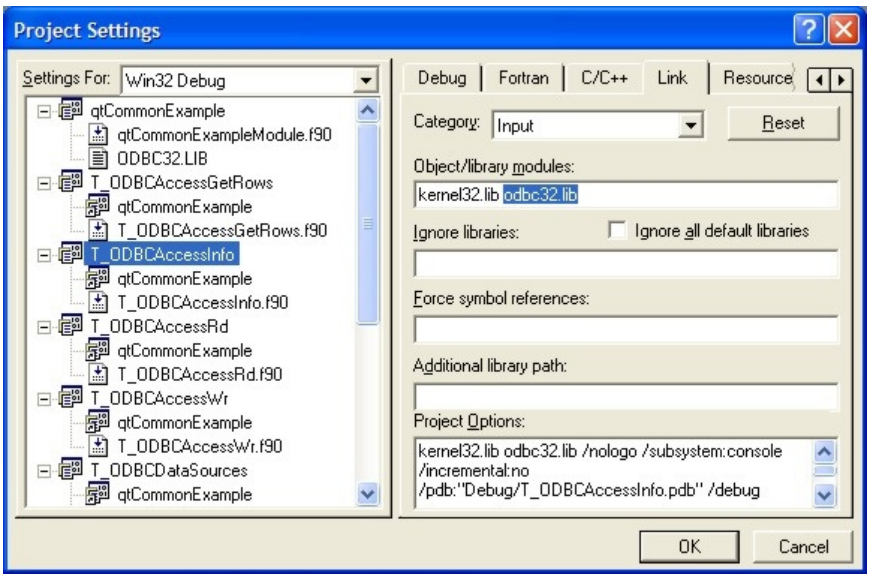
When using ForDBC, the compiler needs to know where the module files (.mod) can be found. Thus, you have to supply the module path.
In the development environment (Developer Studio) specify this in the settings of the project: dialog "Project Settings", choose "Settings For:", "All Configurations", select "Fortran" tab, "Category" "Preprocessor", enter in the edit field "Module path:" <your module path>.




*Illus. 15: Specifying the module path for CVF*

For the link step, you have to supply the import library

```
ODBC32.LIB
```




*Illus. 14: Specifying the ODBC32.LIB for CVF*

and the CVF specific ForDBC library

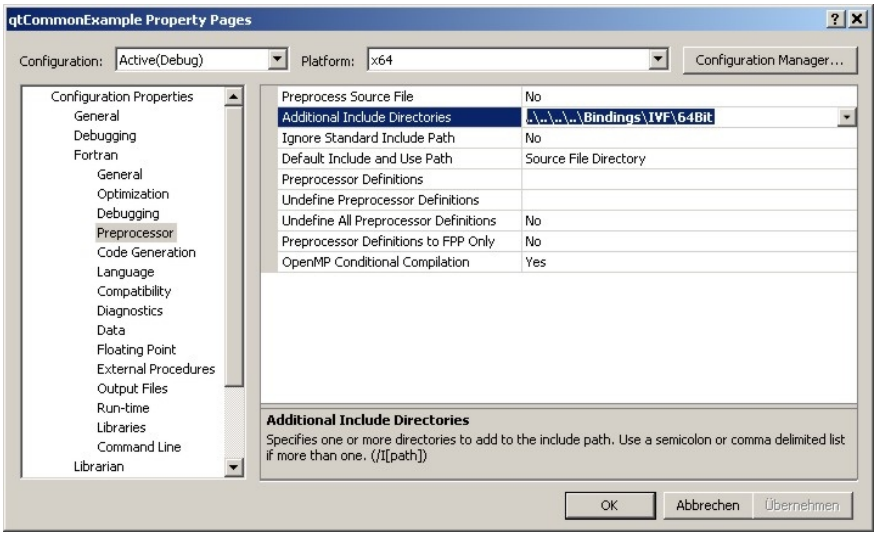
```
ForDBC_CVF.LIB
```

In the dialog "Project Settings", choose "Settings For:", "All Configurations", select "Link" tab, "Category" "Input", enter in the edit field "Object/library

modules:" kernel32.lib odbc32.lib.
You may add ForDBC_CVF.lib here too (and then also specifying its path), but it is better to add ForDBC_CVF.lib to the files of your project.

### 6.3.2 Intel Visual Fortran (IVF)

To use ForDBC the compiler needs to know where the module files (.mod) can be found. Thus, you have to supply the module path.
In the development environment (Visual Studio) open the dialog "Property Pages" of your IVF project: Choose in the list box titled "Configuration" "All Configurations", then select in the "Configuration Properties" treeview "Fortran | Preprocessor" and enter in the entry field named "Additional Include Directories" the module path where the ForDBC .mod files are located.

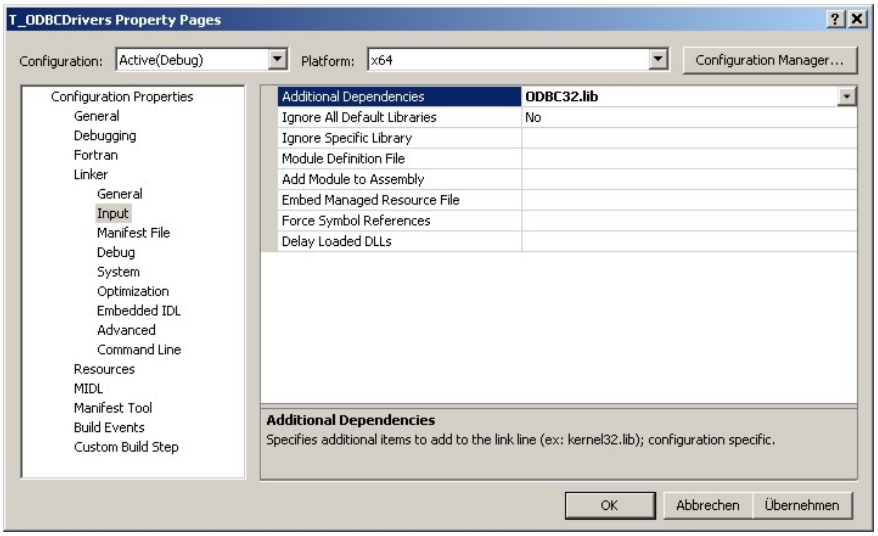



*Illus. 16: Specifying the module path for IVF*

For the link step, you have to supply the

```
ODBC32.LIB
```

Open the dialog "Property Pages" of your IVF project: Choose in the list box titled "Configuration" "All Configurations", then select in the "Configuration Properties" treeview "Linker | Input" and enter in the edit field "Additional Dependencies:" odbc32.lib.




*Illus. 17: Specifying the ODBC32.LIB in VS*

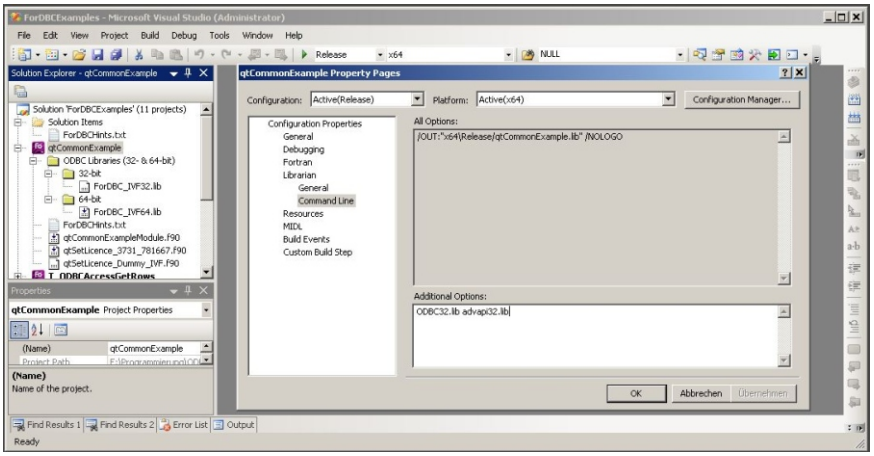
Additionally you have to add

```
ForDBC_IVF32.LIB   or   ForDBC_IVF64.LIB
```

(for 32-bit or 64-bit programs, respectively) and the systems library

```
advapi32.lib
```

to the files of your project . If you alternatively add the ODBC32.lib to your project files, make sure you select the correct one (either 32-bit or 64-bit). Then your project can only be used to create either version (but not both 32-bit and 64-bit which would be possible when specifying ODBC32.lib in the linker input field).
In the sample programs, you will find both systems libraries (odbc32.lib and advapi32.lib) being specified as "Additional Options" of the Librarian's "Command Line" for the sample library project "qtCommonExample".




*Illus. 18: odbc32.lib and advapi32.lib are specified as additional options of the Librarian's command line (project "qtCommonExample")*

### 6.3.2.1 Initialization and Licensing of ForDBC

Since version 3.25 of ForDBC, it has to be initialized by calling the procedure `ForDBC_Initialize` and passing the licence number obtained (format: L0889-###### for 32-bit programs, L3731-##### for 64-bit programs; with # = 0, 1, 2,..., 8 or 9). E.g.:

```
! for 32-32-bit programs
iRet = ForDBC_Initialize('L0889-123456')
```

If the licence is found and is valid, the function returns 0 (iRet = 0; otherwise a value /= 0 is returned). If you don't own a ForDBC licence, you can use it in demonstration mode:

```
! in demo mode
iRet = ForDBC_Initialize('evaluation')
```

In `ForDBC_Initialize` the routine `qtSetLicence_ForDBC` (in case of 32-bit) or `qtSetLicence_ForDBC64` (in case of 64-bit), respectively, is called. This routine is supplied as a "dummy" for testing (demo mode):

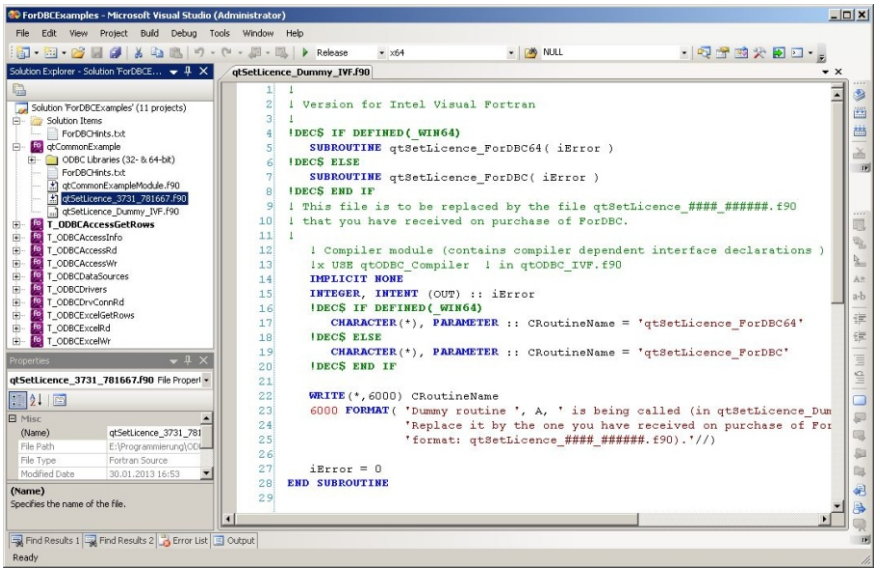
```
qtSetLicence_Dummy_IVF.f90
```

If you have acquired a ForDBC licence, you have to replace that by the licence routine you have obtained.

```
qtSetLicence_####_######.f90
(# = a digit 0 ... 9)
```

This means you have to replace the file qtSetLicence_Dummy_IVF.f90 by

qtSetLicence_####_######.f90 in your projects, and you have to specify your licence number to `ForDBC_Initialize`.




*Illus. 19: The licence routine in qtSetLicence_####_######.f90 replaces the dummy licence, which was then excluded from "Build".*

# 7. ForDBC Functions - Overview

The following table lists those functions being supplied by ForDBC. It also mentions the ODBC level (cf. chapter "ODBC Conformance Levels"). A complete listing of the ForDBC INTERFACE definitions is found in appendix A.

| Function name | Short description | ODBC Level |
|---|---|---|
| SQLAllocConnect | Allocate memory for connection | (C) |
| SQLAllocEnv | Allocate environment | (C) |
| SQLAllocHandle | Allocate handle | (3) |
| SQLAllocStmt | Allocate memory for statement | (C) |
| SQLBindCol | Bind column | (C) |
| SQLBindCol*xxxx* | Bind column (*xxxx* = Char, I1, I2, I4, LP, R4 und DP) | (C) |
| SQLBindParameter | Bind a buffer to a parameter marker in an SQL statement | (1) |
| SQLBindParameter*xxxx* | Bind a buffer to a parameter marker in an SQL statement (*xxxx* = Char, I1, I2, I4, LP, R4 und DP) | (1) |
| SQLBrowseConnect | Connect using „browsing" methods | (1) |
| SQLBulkOperations | Performs bulk insertions and bulk bookmark operations | (3) |
| SQLCancel | Cancel a processing | (2) |
| SQLCloseCursor | Close cursor | (3) |
| SQLColAttribute | Return descriptor information for a column in a result set | (3) |
| SQLColAttributeChar | Return descriptor information (CHARACTER type) for a column in a result set | (3) |
| SQLColAttributes | Return descriptor information for a column in a result set | (C) |
| SQLColumnPrivileges | Get column privileges | (1) |
| SQLColumns*xxxx* | Return a list of column names (*xxxx* = Char und LP) | (1) |
| SQLConnect | Connect to datasource | (C) |
| SQLCopyDesc | Copy descriptor information | (3) |
| SQLDataSources | List data sources | (2) |
| SQLDescribeCol | Return the result descriptor of a column | (C) |
| SQLDescribeParam | Return the description of a parameter marker | (2) |
| SQLDisconnect | Disconnect | (C) |
| SQLDriverConnect | Connect and return driver information | (1) |
| SQLDrivers | Return driver information | (2) |
| SQLEndTran | End transaction | (3) |
| SQLError | Return error information | (C) |
| SQLExecDirect | Execute SQL statement directly | (C) |
| SQLExecute | Execute prepared SQL statement | (C) |
| SQLExtendedFetch | Fetch rowset | (2) |
| SQLFetch | Fetch row from the result set | (C) |
| SQLFetchScroll | Fetches the specified rowset of data | (3) |
| SQLForeignKeys | Return list of foreign keys | (1) |
| SQLFreeConnect | Free connection memory | (C) |
| SQLFreeEnv | Free environment memory | (C) |
| SQLFreeHandle | Free handle | (3) |
| SQLFreeStmt | Free statement | (C) |
| SQLGetConnectAttr | Get connection attribute settings (to buffer) | (3) |
| SQLGetConnectAttrChar | Get connection attribute settings (to CHARACTER buffer) | (3) |
| SQLGetConnectOption | Get the current settings of a connection option | (1) |
| SQLGetConnectOption*xxxx* | Get the current settings of a connection option (*xxxx* = Char und I4) | (1) |
| SQLGetCursorName | Get cursor name | (C) |
| SQLGetData | Get result data for a single unbound column in the current row | (1) |
| SQLGetData*xxxx* | Get result data for a single unbound column in the current row (*xxxx* = Char, I2, I4, R4 und DP) | (1) |
| SQLGetDescField | Get descriptor field settings | (3) |
| SQLGetDescRec | Get settings for decsriptor record fields | (3) |
| SQLGetDiagField | Get value of a field of a record of the diagnostic data structure | (3) |
| SQLGetDiagRec | Get values of a diagnostic record | (3) |
| SQLGetEnvAttr*xxxx* | Get environment attribute settings (*xxxx* = Char und I4) | (3) |
| SQLGetFunctions | Check if function supported | (1) |
| SQLGetInfo | Get general driver information | (1) |
| SQLGetInfo*xxxx* | Get general driver information (*xxxx* = Char, I2 und I4) | (1) |
| SQLGetStmtAttr | Get environment attribute settings (to any buffer) | (3) |
| SQLGetStmtAttrChar | Get environment attribute settings (to CHARACTER buffer) | (3) |
| SQLGetStmtOption | Set current statement option settings | (1) |
| SQLGetStmtOption*xxxx* | Set current statement option settings (*xxxx* = Char und I4) | (1) |
| SQLGetTypeInfo | Get information about supported data types | (1) |
| SQLMoreResults | Check for more results | (2) |
| SQLNativeSql | Return statement as translated by the driver | (2) |
| SQLNumParams | Return the number of parameters in an SQL statement | (2) |

| | | |
|---|---|---|
| SQLNumResultCols | Return the number of columns in a result set | (C) |
| SQLParamOptions | Set parameters | (1) |
| SQLParamData | Supply parameter data | (1) |
| SQLParamData*xxxx* | Supply parameter data (*xxxx* = Char, I1, I2, I4, R4 und DP) | (1) |
| SQLPrepare | Prepare SQL string for execution | (C) |
| SQLPrimaryKeys | Get primary keys of a table | (1) |
| SQLProcedureColumns | Returns input and output parameters and columns of the result set for specified procedures | (1) |
| SQLProcedures | Returns list of procedure names | (1) |
| SQLPutData | Send data for a parameter or column to the driver | (1) |
| SQLPutData*xxxx* | Send data for a parameter or column to the driver (*xxxx* = Char, I2, I4, R4 und DP) | (1) |
| SQLRowCount | Return the number of rows | (C) |
| SQLSetConnectAttr | Set connection attribute | (3) |
| SQLSetConnectAttr*xxxx* | Set connection attribute (*xxxx* = Char und I4) | (3) |
| SQLSetConnectOption | Set connection option | (1) |
| SQLSetCursorName | Set cursor name | (C) |
| SQLSetDescField | Set descriptor field | (3) |
| SQLSetDescFieldChar | Set descriptor field | (3) |
| SQLSetDescRec | Set descriptor fields in a record | (3) |
| SQLSetEnvAttr | Set environment attribute | (3) |
| SQLSetEnvAttrChar | Set environment attribute (if CHARACTER type attribute) | (3) |
| SQLSetPos | Set cursor position | (2) |
| SQLSetStmtAttr | Set statement attributes | (3) |
| SQLSetStmtAttr*xxxx* | Set statement attributes (*xxxx* = Char und I4) | (3) |
| SQLSetScrollOptions | Set options for controlling the cursor behaviour | (2) |
| SQLSetStmtOption | Set statement option | (1) |
| SQLSpecialColumns | Get special columns | (1) |
| SQLStatistics | Retrieve table statistics | (1) |
| SQLTablePrivileges | Return a list of tables and their privileges | (1) |
| SQLTables | Return a list of table names | (1) |
| SQLTablesLP | Return a list of table names (LP arguments) | (1) |
| SQLTransact | Commit transaction | (C) |

ODBC Level: C = core, 1 = level 1, 2 = level 2, 3 = level 3

# 8. References / Literature

References to [ODBC..] refer to:


[ODBC08] Online-Help Microsoft Visual Studio 2008
Win32 anc COM Development\Data Access and Storage
\Microsoft Open Database Connectivity (ODBC)
\Microsoft Open Database Connectivity (ODBC)\ODBC Programmer's Reference\Developing Applications and Drivers\Basic ODBC Application Steps

[ODBC96] Microsoft Developer Network, Library 1996: Product Documentation\SDKs\Open Database Connectivity\Programmer's Reference

[ODBC98] Microsoft Developer Network, Library Visual Studio 6.0, 1998: Plattorm-SDK\Database- and Messaging-Services\Microsoft Data Access SDK\ SDKs\Open Database Connectivity (ODBC)\ODBC Programmer's Reference

[ODBC-C] [ODBC96] Part 6 Appendixes\Appendix C

[ODBC-E] [ODBC96] Part 2 Developing Applications\Chapter 8 Retrieving Status and Error Information\ODBC Error Messages

[ODBC-I] [ODBC96] Part 2 Developing Applications\Chapter 10 Constructing an ODBC Application\Installing and Configuring ODBC Software

[ODBC-R] [ODBC96] Part 2 Developing Applications\Chapter 7 Retrieving Results\ODBC Extensions for Results

[SQL] Wolfgang Misgeld: SQL - Einführung und Anwendung, Hanser Verlag, ISBN 3-446-18260-8

# Appendix A - ForDBC Functions

## qtODBCInterfaces

```
00001 !=================================================================
00002 ! (C) Copyright Joerg Kuthe, Germany, 1999 - 2018
00003 !     All rights reserved. http://www.qtsoftware.de
00004 !———————————————————————————
00005 !
00006 ! qtODBCInterfaces for FTN95, CVF, IVF, ...
00007 !
00008 ! DVF/CVF
00009 ! ———————
00010 ! compile: DF qtODBCInterfaces.F90 -c -win -compile_only -nologo -libs:dll /warn:nofileopt -dll
00011 !
00012 ! IVF
00013 ! ——
00014 ! compile: IFORT qtODBCInterfaces.F90 /nologo /Od /libs:static /threads /c
00015 !
00016 ! LF95
00017 ! ——
00018 ! compile: LF95 qtODBCInterfaces.f90 -nwrap -c -win -mod d:.mod&obj -ml msvc
00019 !          mit "d:.mod&obj" als dem Modulpfad
00020 !
00021 MODULE qtODBCInterfaces
00022
00023    INTERFACE SQLAllocConnect
00024       FUNCTION SQLAllocConnect(env, dbc)
00025          USE qtODBCKinds
00026          INTEGER (SQLRETURN) :: SQLAllocConnect
00028          INTEGER (SQLHENV) :: env
00029          INTEGER (SQLHDBC) :: dbc
00031       END FUNCTION SQLAllocConnect
00032    END INTERFACE
00033
00034    INTERFACE SQLAllocEnv
00035       FUNCTION SQLAllocEnv( env )
00036          USE qtODBCKinds
00037          INTEGER (SQLRETURN) :: SQLAllocEnv
00038          INTEGER (SQLHENV) :: env
00041       END FUNCTION SQLAllocEnv
00042    END INTERFACE
00043
00044    INTERFACE SQLAllocHandle
00045       FUNCTION SQLAllocHandle( HandleType, InputHandle, OutputHandlePtr )
00046          USE qtODBCKinds
00047          INTEGER (SQLRETURN) :: SQLAllocHandle
00048          INTEGER (SQLSMALLINT) :: HandleType
00049          INTEGER (SQLHANDLE) :: InputHandle
00050          INTEGER (SQLHANDLE) :: OutputHandlePtr
00053       END FUNCTION SQLAllocHandle
00054    END INTERFACE
00055
00056    INTERFACE SQLAllocStmt
00057       FUNCTION SQLAllocStmt( dbc, phstmt )
00058          USE qtODBCKinds
00059          INTEGER (SQLRETURN) :: SQLAllocStmt
00060          INTEGER (SQLHDBC) :: dbc
00061          INTEGER (SQLHSTMT) :: phstmt
00064       END FUNCTION SQLAllocStmt
00065    END INTERFACE
00066
00067
00068    INTERFACE SQLBindCol
00069       !   SQLRETURN SQLBindCol(
00070       !        SQLHSTMT       StatementHandle,
00071       !        SQLUSMALLINT   ColumnNumber,
00072       !        SQLSMALLINT    TargetType,
00073       !        SQLPOINTER     TargetValuePtr,
00074       !        SQLLEN         BufferLength,
00075       !        SQLLEN *       StrLen_or_Ind);
00076
00077       FUNCTION SQLBindColChar( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
00078       ! bind CHAR column
00079          USE qtODBCKinds
00080          INTEGER (SQLRETURN) :: SQLBindColChar
00081          INTEGER (SQLHSTMT) :: stmt
00082          INTEGER (SQLUSMALLINT) :: icol
00083          INTEGER (SQLSMALLINT) :: fCType
00084          CHARACTER(*) rgbValue
00085          INTEGER (SQLLEN) :: cbValueMax
00086          INTEGER (SQLLEN) :: pcbValue
```

```
00089       END FUNCTION SQLBindColChar
00090
00091       FUNCTION SQLBindColI1( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
00092       ! bind INTEGER*1 column
00093          USE qtODBCKinds
00094          INTEGER (SQLRETURN) :: SQLBindColI1
00095          INTEGER (SQLHSTMT) :: stmt
00096          INTEGER (SQLUSMALLINT) :: icol
00097          INTEGER (SQLSMALLINT) :: fCType
00098          INTEGER(BYTE) rgbValue
00099          INTEGER (SQLLEN) :: cbValueMax
00100         INTEGER (SQLLEN) :: pcbValue
00103       END FUNCTION SQLBindColI1
00104
00105       FUNCTION SQLBindColI2( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
00106       ! bind INTEGER(SHORT) column
00107          USE qtODBCKinds
00108          INTEGER (SQLRETURN) :: SQLBindColI2
00109          INTEGER (SQLHSTMT) :: stmt
00110          INTEGER (SQLUSMALLINT) :: icol
00111          INTEGER (SQLSMALLINT) :: fCType
00112          INTEGER(SHORT) rgbValue
00113          INTEGER (SQLLEN) :: cbValueMax
00114          INTEGER (SQLLEN) :: pcbValue
00117       END FUNCTION SQLBindColI2
00118
00119       FUNCTION SQLBindColI4( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
00120       ! bind INTEGER(LONG) column
00121          USE qtODBCKinds
00122          INTEGER (SQLRETURN) :: SQLBindColI4
00123          INTEGER (SQLHSTMT) :: stmt
00124          INTEGER (SQLUSMALLINT) :: icol
00125          INTEGER (SQLSMALLINT) :: fCType
00126          INTEGER(LONG) rgbValue
00127          INTEGER (SQLLEN) :: cbValueMax
00128          INTEGER (SQLLEN) :: pcbValue
00131       END FUNCTION SQLBindColI4
00132
00133       FUNCTION SQLBindColR4( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
00134       ! bind REAL(FLOAT) column
00135          USE qtODBCKinds
00136          INTEGER (SQLRETURN) :: SQLBindColR4
00137          INTEGER (SQLHSTMT) :: stmt
00138          INTEGER (SQLUSMALLINT) :: icol
00139          INTEGER (SQLSMALLINT) :: fCType
00140          REAL(FLOAT) rgbValue
00141          INTEGER (SQLLEN) :: cbValueMax
00142          INTEGER (SQLLEN) :: pcbValue
00145       END FUNCTION SQLBindColR4
00146
00147       FUNCTION SQLBindColDP( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
00148       ! bind DOUBLE PRECISION column
00149          USE qtODBCKinds
00150          INTEGER (SQLRETURN) :: SQLBindColDP
00151          INTEGER (SQLHSTMT) :: stmt
00152          INTEGER (SQLUSMALLINT) :: icol
00153          INTEGER (SQLSMALLINT) :: fCType
00154          DOUBLE PRECISION rgbValue
00155          INTEGER (SQLLEN) :: cbValueMax
00156          INTEGER (SQLLEN) :: pcbValue
00159       END FUNCTION SQLBindColDP
00160
00161    END INTERFACE SQLBindCol
00162
00163    INTERFACE
00164       FUNCTION SQLBindColLP( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )   ! added 15.10.2000
00165          ! bind column via pointer (use LOC() function to get variable's address)
00166          USE qtODBCKinds
00167          INTEGER (SQLRETURN) :: SQLBindColLP
00168          INTEGER (SQLHSTMT) :: stmt
00169          INTEGER (SQLUSMALLINT) :: icol
00170          INTEGER (SQLSMALLINT) :: fCType
00171          INTEGER (SQLPOINTER) :: rgbValue
00172          INTEGER (SQLLEN) :: cbValueMax
00173          INTEGER (SQLLEN) :: pcbValue
00176       END FUNCTION SQLBindColLP
00177    END INTERFACE
00178
00179
00180    INTERFACE SQLBindParameter
00181       !   SQLRETURN SQLBindParameter(
00182       !        SQLHSTMT     StatementHandle,
00183       !        SQLUSMALLINT     ParameterNumber,
00184       !        SQLSMALLINT     InputOutputType,
00185       !        SQLSMALLINT     ValueType,
00186       !        SQLSMALLINT     ParameterType,
00187       !        SQLULEN        ColumnSize,
00188       !        SQLSMALLINT     DecimalDigits,
00189       !        SQLPOINTER     ParameterValuePtr,
00190       !        SQLINTEGER     BufferLength,
00191       !        SQLLEN *     StrLen_or_IndPtr);
00192
```

```
00193       FUNCTION SQLBindParameterChar( stmt, ipar,    &
00194                                      fParamType, fCType, fSqlType, cbColDef,            &
00195                                      ibScale, rgbValue, cbValueMax, pcbValue )
00196          ! rgbValue is a CHARACTER buffer
00197          USE qtODBCKinds
00198          INTEGER (SQLRETURN) :: SQLBindParameterChar
00199          INTEGER (SQLHSTMT) :: stmt
00200          INTEGER (SQLUSMALLINT) :: ipar
00201          CHARACTER (LEN=*) :: rgbValue
00202          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00203          !2011-08-08# INTEGER (SQLUINTEGER) :: cbColDef
00204          !2011-08-08# INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00205          INTEGER (SQLULEN) :: cbColDef
00206          INTEGER (SQLLEN) :: cbValueMax, pcbValue
00209       END FUNCTION SQLBindParameterChar
00210
00211       FUNCTION SQLBindParameterI1( stmt, ipar,    &
00212                                    fParamType, fCType, fSqlType, cbColDef,            &
00213                                    ibScale, rgbValue, cbValueMax, pcbValue )
00214          ! rgbValue is an INTEGER*1 value
00215          USE qtODBCKinds
00216          INTEGER (SQLRETURN) :: SQLBindParameterI1
00217          INTEGER (SQLHSTMT) :: stmt
00218          INTEGER (SQLUSMALLINT) :: ipar
00219          INTEGER(BYTE) rgbValue
00220          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00221          INTEGER (SQLUINTEGER) :: cbColDef
00222          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00225       END FUNCTION SQLBindParameterI1
00226
00227       FUNCTION SQLBindParameterI2( stmt, ipar,    &
00228                                    fParamType, fCType, fSqlType, cbColDef,            &
00229                                    ibScale, rgbValue, cbValueMax, pcbValue )
00230          ! rgbValue is an INTEGER(SHORT) value
00231          USE qtODBCKinds
00232          INTEGER (SQLRETURN) :: SQLBindParameterI2
00233          INTEGER (SQLHSTMT) :: stmt
00234          INTEGER (SQLUSMALLINT) :: ipar
00235          INTEGER(SHORT) rgbValue
00236          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00237          !2011-08-08# INTEGER (SQLUINTEGER) :: cbColDef
00238          !2011-08-08# INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00239          INTEGER (SQLULEN) :: cbColDef
00240          INTEGER (SQLLEN) :: cbValueMax, pcbValue
00243       END FUNCTION SQLBindParameterI2
00244
00245       FUNCTION SQLBindParameterI4( stmt, ipar,    &
00246                                    fParamType, fCType, fSqlType, cbColDef,            &
00247                                    ibScale, rgbValue, cbValueMax, pcbValue )
00248          ! rgbValue is an INTEGER(LONG) value
00249          USE qtODBCKinds
00250          INTEGER (SQLRETURN) :: SQLBindParameterI4
00251          INTEGER (SQLHSTMT) :: stmt
00252          INTEGER (SQLUSMALLINT) :: ipar
00253          INTEGER(LONG) rgbValue
00254          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00255          !2011-08-08# INTEGER (SQLUINTEGER) :: cbColDef
00256          !2011-08-08# INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00257          INTEGER (SQLULEN) :: cbColDef
00258          INTEGER (SQLLEN) :: cbValueMax, pcbValue
00261       END FUNCTION SQLBindParameterI4
00262
00263       FUNCTION SQLBindParameterR4( stmt, ipar,    &
00264                                    fParamType, fCType, fSqlType, cbColDef,            &
00265                                    ibScale, rgbValue, cbValueMax, pcbValue )
00266          ! rgbValue is a REAL(FLOAT) value
00267          USE qtODBCKinds
00268          INTEGER (SQLRETURN) :: SQLBindParameterR4
00269          INTEGER (SQLHSTMT) :: stmt
00270          INTEGER (SQLUSMALLINT) :: ipar
00271          REAL(FLOAT) rgbValue
00272          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00273          !2011-08-08# INTEGER (SQLUINTEGER) :: cbColDef
00274          !2011-08-08# INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00275          INTEGER (SQLULEN) :: cbColDef
00276          INTEGER (SQLLEN) :: cbValueMax, pcbValue
00279       END FUNCTION SQLBindParameterR4
00280
00281       FUNCTION SQLBindParameterDP( stmt, ipar,    &
00282                                    fParamType, fCType, fSqlType, cbColDef,            &
00283                                    ibScale, rgbValue, cbValueMax, pcbValue )
00284          ! rgbValue is an DOUBLE PRECISION value
00285          USE qtODBCKinds
00286          INTEGER (SQLRETURN) :: SQLBindParameterDP
00287          INTEGER (SQLHSTMT) :: stmt
00288          INTEGER (SQLUSMALLINT) :: ipar
00289          DOUBLE PRECISION rgbValue
00290          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00291          !2011-08-08# INTEGER (SQLUINTEGER) :: cbColDef
00292          !2011-08-08# INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00293          INTEGER (SQLULEN) :: cbColDef
00294          INTEGER (SQLLEN) :: cbValueMax, pcbValue
```

```
00297       END FUNCTION SQLBindParameterDP
00298
00299    END INTERFACE SQLBindParameter
00300
00301    INTERFACE   ! added 19.10.2000
00302       FUNCTION SQLBindParameterLP( stmt, ipar,    &
00303                                    fParamType, fCType, fSqlType, cbColDef,            &
00304                                    ibScale, rgbValue, cbValueMax, pcbValue )
00305          ! rgbValue is a pointer (use LOC()
00306          USE qtODBCKinds
00307          INTEGER (SQLRETURN) :: SQLBindParameterLP
00308          INTEGER (SQLHSTMT) :: stmt
00309          INTEGER (SQLUSMALLINT) :: ipar
00310          INTEGER (SQLPOINTER) :: rgbValue
00311          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00312          !2011-08-08# INTEGER (SQLUINTEGER) :: cbColDef
00313          !2011-08-08# INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00314          INTEGER (SQLULEN) :: cbColDef
00315          INTEGER (SQLLEN) :: cbValueMax, pcbValue
00318       END FUNCTION SQLBindParameterLP
00319    END INTERFACE
00320
00321    INTERFACE SQLBrowseConnect
00322       FUNCTION SQLBrowseConnect( dbc, InConnectionString, cbInConnStr,    &
00323                                  OutConnectionString, cbOutConnStr, pbOutConnStrLength )
00324          USE qtODBCKinds
00325          INTEGER (SQLRETURN) :: SQLBrowseConnect
00326          INTEGER (SQLHDBC) :: dbc
00327          CHARACTER(*) InConnectionString, OutConnectionString
00328          INTEGER (SQLSMALLINT) :: cbInConnStr, cbOutConnStr, pbOutConnStrLength
00331       END FUNCTION SQLBrowseConnect
00332    END INTERFACE
00333
00334
00335    INTERFACE SQLBulkOperations
00336       FUNCTION SQLBulkOperations( Stmt, Operation )
00337          USE qtODBCKinds
00338          INTEGER (SQLRETURN) :: SQLBulkOperations
00339          INTEGER (SQLHSTMT) :: Stmt
00340          INTEGER (SQLUSMALLINT) :: Operation
00342       END FUNCTION SQLBulkOperations
00343    END INTERFACE
00344
00345    INTERFACE SQLCancel
00346       FUNCTION SQLCancel( stmt )
00347          USE qtODBCKinds
00348          INTEGER (SQLRETURN) :: SQLCancel
00349          INTEGER (SQLHSTMT) :: stmt
00351       END FUNCTION SQLCancel
00352    END INTERFACE
00353
00354    INTERFACE SQLCloseCursor
00355       FUNCTION SQLCloseCursor( Stmt )
00356          USE qtODBCKinds
00357          INTEGER (SQLRETURN) :: SQLCloseCursor
00358          INTEGER (SQLHSTMT) :: Stmt
00360       END FUNCTION SQLCloseCursor
00361    END INTERFACE
00362
00363    INTERFACE SQLColAttribute
00364      ! charAttribute is a CHARACTER buffer
00365       FUNCTION SQLColAttributeChar( stmt, icol, fieldId, charAttribute,      &
00366                                     lenCharAttribute, CharAttrLength, NumAttribute )
00367          USE qtODBCKinds
00368          INTEGER (SQLRETURN) :: SQLColAttributeChar
00369          INTEGER (SQLHSTMT) :: stmt
00370          INTEGER (SQLUSMALLINT) :: icol, fieldId
00371          CHARACTER (LEN=*) :: charAttribute
00372          INTEGER (SQLSMALLINT) :: lenCharAttribute, CharAttrLength
00373          !2011-08-08# INTEGER (SQLINTEGER) :: NumAttribute
00374          INTEGER (SQLLEN) :: NumAttribute
00377       END FUNCTION SQLColAttributeChar
00378
00379      ! charAttribute is a pointer
00380       FUNCTION SQLColAttribute( stmt, icol, fieldId, charAttribute,    &
00381                                 lenCharAttribute, CharAttrLength, NumAttribute )
00382          USE qtODBCKinds
00383          INTEGER (SQLRETURN) :: SQLColAttribute
00384          INTEGER (SQLHSTMT) :: stmt
00385          INTEGER (SQLUSMALLINT) :: icol, fieldId
00386          INTEGER (SQLPOINTER) :: charAttribute
00387          INTEGER (SQLSMALLINT) :: lenCharAttribute, CharAttrLength
00388          !2011-08-08# INTEGER (SQLINTEGER) :: NumAttribute
00389          INTEGER (SQLLEN) :: NumAttribute
00392       END FUNCTION SQLColAttribute
00393    END INTERFACE
00394
00395    INTERFACE SQLColAttributes
00396       FUNCTION SQLColAttributes( stmt, icol,  &
00397                                  fDescType, rgbDesc, cbDescMax, pcbDesc, pfDesc )
00398          USE qtODBCKinds
```

```
00399          INTEGER (SQLRETURN) :: SQLColAttributes
00400          INTEGER (SQLHSTMT) :: stmt
00401          INTEGER (SQLUSMALLINT) :: icol, fDescType
00402          CHARACTER (LEN=*) :: rgbDesc
00403          INTEGER (SQLSMALLINT) :: cbDescMax, pcbDesc
00404          !2011-08-08# INTEGER (SQLINTEGER) :: pfDesc
00405          INTEGER (SQLLEN ) :: pfDesc
00408       END FUNCTION SQLColAttributes
00409    END INTERFACE
00410
00411    INTERFACE SQLColumnPrivileges
00412       FUNCTION SQLColumnPrivileges( stmt,   &
00413                                     CatalogName, LenCatName,   &
00414                                     SchemaName, LenSchemaName, &
00415                                     TableName, LenTableName,  &
00416                                     ColumnName, LenColName )
00417          USE qtODBCKinds
00418          INTEGER (SQLRETURN) :: SQLColumnPrivileges
00419          INTEGER (SQLHSTMT) :: stmt
00420          CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName, ColumnName
00421          INTEGER (SQLSMALLINT) :: LenCatName, LenSchemaName, LenTableName, LenColName
00424       END FUNCTION SQLColumnPrivileges
00425    END INTERFACE
00426
00427    INTERFACE SQLColumns
00428       !      SQLRETURN SQLColumns(
00429       !        SQLHSTMT     StatementHandle,
00430       !        SQLCHAR *     CatalogName,
00431       !        SQLSMALLINT     NameLength1,
00432       !        SQLCHAR *     SchemaName,
00433       !        SQLSMALLINT     NameLength2,
00434       !        SQLCHAR *     TableName,
00435       !        SQLSMALLINT     NameLength3,
00436       !        SQLCHAR *     ColumnName,
00437       !        SQLSMALLINT     NameLength4);
00438       FUNCTION SQLColumnsChar( stmt,  &
00439                                szTableQualifier, cbTableQualifier,       &
00440                                szTableOwner, cbTableOwner,               &
00441                                szTableName, cbTableName,                 &
00442                                szColumnName, cbColumnName )      ! changed 14.10.2000: szColumnName, cbColum
00442-1                                 nName )    ! changed 14.10.2000: SQLColumns -> SQLColumnsChar
00443          USE qtODBCKinds
00444          INTEGER (SQLRETURN) :: SQLColumnsChar
00445          INTEGER (SQLHSTMT) :: stmt
00446          CHARACTER (LEN=*) :: szTableQualifier, szTableOwner,  &
00447                               szTableName, szColumnName
00448          INTEGER (SQLSMALLINT) :: cbTableQualifier, cbTableOwner, cbTableName, cbColumnName
00452       END FUNCTION SQLColumnsChar
00453
00454       ! 14.10.2000: added SQLColumnsLP (all arguments being transferred as values, use LOC() to pass a refer
00454-1          ence)
00455       FUNCTION SQLColumnsLP( stmt,  &
00456                              szTableQualifier, cbTableQualifier,       &
00457                              szTableOwner, cbTableOwner,               &
00458                              szTableName, cbTableName,                 &
00459                              szColumnName, cbColumnName )
00460          USE qtODBCKinds
00461          INTEGER (SQLRETURN) :: SQLColumnsLP
00462          INTEGER (SQLHSTMT) :: stmt
00463          INTEGER (SQLPOINTER) :: szTableQualifier, szTableOwner,  &
00464                                  szTableName, szColumnName
00465          INTEGER (SQLSMALLINT) :: cbTableQualifier, cbTableOwner, cbTableName, cbColumnName
00467       END FUNCTION SQLColumnsLP
00468    END INTERFACE SQLColumns
00469
00470    INTERFACE SQLConnect
00471       FUNCTION SQLConnect( dbc, szDSN, cbDSN, szUID, cbUID, szAuthStr, cbAuthStr )
00472          USE qtODBCKinds
00473          INTEGER (SQLRETURN) ::SQLConnect
00474          INTEGER (SQLHDBC) :: dbc
00475          CHARACTER(*) szDSN, szUID, szAuthStr
00476          INTEGER (SQLSMALLINT) :: cbDSN, cbUID, cbAuthStr
00479       END FUNCTION SQLConnect
00480    END INTERFACE
00481
00482    INTERFACE SQLCopyDesc
00483       FUNCTION SQLCopyDesc( SourceDescHandle, TargetDescHandle )
00484          USE qtODBCKinds
00485          INTEGER (SQLRETURN) :: SQLCopyDesc
00486          INTEGER (SQLHDESC) :: SourceDescHandle, TargetDescHandle
00488       END FUNCTION SQLCopyDesc
00489    END INTERFACE
00490
00491    INTERFACE SQLDataSources
00492       FUNCTION SQLDataSources( env, fDirection,   &
00493                                szDSN, cbDSNMax, pcbDSN,   &
00494                                szDescription, cbDescriptionMax, pcbDescription )
00495          USE qtODBCKinds
00496          INTEGER (SQLRETURN) :: SQLDataSources
00497          INTEGER (SQLHENV) :: env
00498          INTEGER (SQLUSMALLINT) :: fDirection
00499          CHARACTER (LEN=*) :: szDSN, szDescription
```

```
00500          INTEGER (SQLSMALLINT) :: cbDSNMax, pcbDSN, cbDescriptionMax, pcbDescription
00504       END FUNCTION SQLDataSources
00505    END INTERFACE
00506
00507    INTERFACE SQLDescribeCol
00508       FUNCTION SQLDescribeCol( stmt, icol,    &
00509                                szColName, cbColNameMax, pcbColName,   &
00510                                pfSqlType, pcbColDef, pibScale, pfNullable )
00511          USE qtODBCKinds
00512          INTEGER (SQLRETURN) :: SQLDescribeCol
00513          INTEGER (SQLHSTMT) :: stmt
00514          INTEGER (SQLUSMALLINT) :: icol
00515          CHARACTER (LEN=*) :: szColName
00516          INTEGER (SQLSMALLINT) :: cbColNameMax, pcbColName, pfSqlType, pibScale, pfNullable
00517          !2011-08-08# INTEGER (SQLUINTEGER) :: pcbColDef
00518          INTEGER (SQLULEN) :: pcbColDef
00523       END FUNCTION SQLDescribeCol
00524    END INTERFACE
00525
00526    INTERFACE SQLDescribeParam
00527       FUNCTION SQLDescribeParam( stmt, ipar, pfSqlType,   &
00528                                  pcbColDef, pibScale, pfNullable )
00529          USE qtODBCKinds
00530          INTEGER (SQLRETURN) :: SQLDescribeParam
00531          INTEGER (SQLHSTMT) :: stmt
00532          INTEGER (SQLUSMALLINT) :: ipar
00533          INTEGER (SQLSMALLINT) :: pfSqlType, pibScale, pfNullable
00534          !2011-08-08# INTEGER (SQLUINTEGER) :: pcbColDef
00535          INTEGER (SQLULEN) :: pcbColDef
00539       END FUNCTION SQLDescribeParam
00540    END INTERFACE
00541
00542    INTERFACE SQLDisconnect
00543       FUNCTION SQLDisconnect( dbc )
00544          USE qtODBCKinds
00545          INTEGER (SQLRETURN) :: SQLDisconnect
00546          INTEGER (SQLHDBC) :: dbc
00548       END FUNCTION SQLDisconnect
00549    END INTERFACE
00550
00551    INTERFACE   ! SQLDriverConnect; DVF5 -> ERROR (could not find generic interface specific function...!)
00552       FUNCTION SQLDriverConnect( dbc, wnd,  &
00553                                  szConnStrIn, cbConnStrIn,  &
00554                                  szConnStrOut, cbConnStrOutMax, pcbConnStrOut, &
00555                                  fDriverCompletion)
00556          USE qtODBCKinds
00557          INTEGER (SQLRETURN) :: SQLDriverConnect
00558          INTEGER (SQLHDBC) :: dbc
00559          INTEGER (SQLHWND) :: wnd
00560          CHARACTER (LEN=*) :: szConnStrIn, szConnStrOut
00561          INTEGER (SQLSMALLINT) :: cbConnStrIn, cbConnStrOutMax, pcbConnStrOut
00562          INTEGER (SQLUSMALLINT) :: fDriverCompletion
00566       END FUNCTION SQLDriverConnect
00567    END INTERFACE
00568
00569    INTERFACE SQLDrivers
00570       FUNCTION SQLDrivers( env, fDirection,   &
00571                            szDrvDesc, cbDrvDescMax, pcbDrvDesc,   &
00572                            szDrvAttr, cbDrvAttrMax, pcbDrvAttr )
00573          USE qtODBCKinds
00574          INTEGER (SQLRETURN) :: SQLDrivers
00575          INTEGER (SQLHENV) :: env
00576          INTEGER (SQLUSMALLINT) :: fDirection
00577          CHARACTER (LEN=*) :: szDrvDesc, szDrvAttr
00578          INTEGER (SQLSMALLINT) :: cbDrvDescMax, pcbDrvDesc, cbDrvAttrMax, pcbDrvAttr
00582       END FUNCTION SQLDrivers
00583    END INTERFACE
00584
00585    INTERFACE SQLEndTran
00586      FUNCTION SQLEndTran( HandleType, hndl, CompletionType )
00587          USE qtODBCKinds
00588          INTEGER (SQLRETURN) :: SQLEndTran
00589          INTEGER (SQLSMALLINT) :: HandleType
00590          INTEGER (SQLHANDLE) :: hndl
00591          INTEGER (SQLSMALLINT) :: CompletionType
00593       END FUNCTION SQLEndTran
00594    END INTERFACE
00595
00596    INTERFACE SQLError
00597       FUNCTION SQLError( env, dbc, stmt, szSqlState, pfNativeError,  &
00598                          szErrorMsg, cbErrorMsgMax, pcbErrorMsg )
00599          USE qtODBCKinds
00600          INTEGER (SQLRETURN) :: SQLError
00601          INTEGER (SQLHENV) :: env
00602          INTEGER (SQLHDBC) :: dbc
00603          INTEGER (SQLHSTMT) :: stmt
00604          CHARACTER(*) szSqlState, szErrorMsg
00605          INTEGER (SQLINTEGER) :: pfNativeError
00606          INTEGER (SQLSMALLINT) :: cbErrorMsgMax, pcbErrorMsg
00609      END FUNCTION SQLError
00610    END INTERFACE
00611
```

```
00612    INTERFACE SQLExecDirect
00613       FUNCTION SQLExecDirect( stmt, szSqlStr, cbSqlStr )
00614          USE qtODBCKinds
00615          INTEGER (SQLRETURN) :: SQLExecDirect
00616          INTEGER (SQLHSTMT) :: stmt
00617          CHARACTER(*) szSqlStr
00618          INTEGER (SQLINTEGER) :: cbSqlStr
00621       END FUNCTION SQLExecDirect
00622    END INTERFACE
00623
00624    INTERFACE SQLExecute
00625       FUNCTION SQLExecute( stmt )
00626          USE qtODBCKinds
00627          INTEGER (SQLRETURN) :: SQLExecute
00628          INTEGER (SQLHSTMT) :: stmt
00630       END FUNCTION SQLExecute
00631    END INTERFACE
00632
00633    INTERFACE SQLExtendedFetch
00634       FUNCTION SQLExtendedFetch( stmt, fFetchType, irow, pcrow, rgfRowStatus )
00635          USE qtODBCKinds
00636          INTEGER (RETCODE) :: SQLExtendedFetch
00638          INTEGER (HSTMT) :: stmt
00639          INTEGER (UWORD) :: fFetchType, rgfRowStatus
00640          !2011-08-08# INTEGER (SDWORD) :: irow
00641          !2011-08-08# INTEGER (UDWORD) :: pcrow
00642          INTEGER (SQLLEN) :: irow
00643          INTEGER (SQLULEN) :: pcrow
00645       END FUNCTION SQLExtendedFetch
00646    END INTERFACE
00647
00648    INTERFACE SQLFetch
00649       FUNCTION SQLFetch( stmt )
00650          USE qtODBCKinds
00651          INTEGER (SQLRETURN) :: SQLFetch
00652          INTEGER (SQLHSTMT) :: stmt
00654       END FUNCTION SQLFetch
00655    END INTERFACE
00656
00657    INTERFACE SQLFetchScroll
00658       FUNCTION SQLFetchScroll( stmt, FetchOrientation, FetchOffset )
00659          USE qtODBCKinds
00660          INTEGER (SQLRETURN) :: SQLFetchScroll
00662          INTEGER (SQLHSTMT) :: stmt
00663          INTEGER (SQLSMALLINT) :: FetchOrientation
00664          !2011-08-08# INTEGER (SQLINTEGER) :: FetchOffset
00665          INTEGER (SQLLEN) :: FetchOffset
00666       END FUNCTION SQLFetchScroll
00667    END INTERFACE
00668
00669    INTERFACE SQLForeignKeys
00670       FUNCTION SQLForeignKeys( stmt, PKCatalogName, PKCatNameLength,    &
00671                                      PKSchemaName, PKSchemaNameLength,  &
00672                                      PKTableName, PKTableNameLength,    &
00673                                      FKCatalogName, FKCatalogNameLength,&
00674                                      FKSchemaName, FKSchemaNameLength,  &
00675                                      FKTableName, FKTableNameLength)
00676          USE qtODBCKinds
00677          INTEGER (SQLRETURN) :: SQLForeignKeys
00679          INTEGER (SQLHSTMT) :: stmt
00680          CHARACTER (LEN=*) :: PKCatalogName, PKSchemaName, PKTableName,  &
00681                               FKCatalogName, FKSchemaName, FKTableName
00684          INTEGER (SQLSMALLINT) :: PKCatNameLength, PKSchemaNameLength, PKTableNameLength, &
00685                                   FKCatalogNameLength, FKSchemaNameLength, FKTableNameLength
00686       END FUNCTION SQLForeignKeys
00687    END INTERFACE
00688
00689    INTERFACE SQLFreeConnect
00690       FUNCTION SQLFreeConnect( dbc )
00691          USE qtODBCKinds
00692          INTEGER (SQLRETURN) :: SQLFreeConnect
00693          INTEGER (SQLHDBC) :: dbc
00695       END FUNCTION SQLFreeConnect
00696    END INTERFACE
00697
00698    INTERFACE SQLFreeEnv
00699       FUNCTION SQLFreeEnv( env )
00700          USE qtODBCKinds
00701          INTEGER (SQLRETURN) :: SQLFreeEnv
00702          INTEGER (SQLHENV) :: env
00704       END FUNCTION SQLFreeEnv
00705    END INTERFACE
00706
00707    INTERFACE SQLFreeHandle
00708       FUNCTION SQLFreeHandle( HndType, Hnd )
00709          USE qtODBCKinds
00710          INTEGER (SQLRETURN) :: SQLFreeHandle
00711          INTEGER (SQLSMALLINT) :: HndType
00712          INTEGER (SQLHANDLE) :: Hnd
00714       END FUNCTION SQLFreeHandle
00715    END INTERFACE
```

```
00716
00717    INTERFACE SQLFreeStmt
00718       FUNCTION SQLFreeStmt( stmt, fOption )
00719          USE qtODBCKinds
00720          INTEGER (SQLRETURN) :: SQLFreeStmt
00721          INTEGER (SQLHSTMT) :: stmt
00722          INTEGER (SQLUSMALLINT) :: fOption
00724       END FUNCTION SQLFreeStmt
00725    END INTERFACE
00726
00727    INTERFACE SQLGetConnectAttrChar
00728    ! ValuePtr is a CHARACTER buffer
00729       FUNCTION SQLGetConnectAttrChar( dbc, Attrib, ValuePtr, LenValuePtr, ValuePtrLength)
00730          USE qtODBCKinds
00732          INTEGER (SQLRETURN) :: SQLGetConnectAttrChar
00733          INTEGER (SQLHDBC) :: dbc
00734          INTEGER (SQLINTEGER) :: Attrib, LenValuePtr, ValuePtrLength
00735          CHARACTER (LEN=*) :: ValuePtr
00737       END FUNCTION SQLGetConnectAttrChar
00738    END INTERFACE
00739
00740    INTERFACE SQLGetConnectAttr
00741    ! ValuePtr is a pointer to a buffer
00742       FUNCTION SQLGetConnectAttr( dbc, Attrib, ValuePtr, LenValuePtr, ValuePtrLength)
00743          USE qtODBCKinds
00745          INTEGER (SQLRETURN) :: SQLGetConnectAttr
00746          INTEGER (SQLHDBC) :: dbc
00747          INTEGER (SQLINTEGER) :: Attrib, LenValuePtr, ValuePtrLength
00748          INTEGER (SQLPOINTER) :: ValuePtr
00750       END FUNCTION SQLGetConnectAttr
00751    END INTERFACE
00752
00753    INTERFACE SQLGetConnectOption
00754       FUNCTION SQLGetConnectOptionChar( dbc, fOption, pvParam )
00755       ! pvParam is a CHARACTER buffer
00756          USE qtODBCKinds
00757          INTEGER (SQLRETURN) :: SQLGetConnectOptionChar
00758          INTEGER (SQLHDBC) :: dbc
00759          INTEGER (SQLUSMALLINT) :: fOption
00760          CHARACTER (LEN=*) :: pvParam
00763       END FUNCTION SQLGetConnectOptionChar
00764
00765       FUNCTION SQLGetConnectOptionI4( dbc, fOption, pvParam )
00766       ! pvParam is an INTEGER(LONG) value
00767          USE qtODBCKinds
00768          INTEGER (SQLRETURN) :: SQLGetConnectOptionI4
00769          INTEGER (SQLHDBC) :: dbc
00770          INTEGER (SQLUSMALLINT) :: fOption
00771          INTEGER(LONG) pvParam
00774       END FUNCTION SQLGetConnectOptionI4
00775    END INTERFACE SQLGetConnectOption
00776
00777    INTERFACE SQLGetCursorName
00778       FUNCTION SQLGetCursorName( stmt, szCursor, cbCursorMax, pcbCursor )
00779          USE qtODBCKinds
00780          INTEGER (SQLRETURN) :: SQLGetCursorName
00781          INTEGER (SQLHSTMT) :: stmt
00782          CHARACTER (LEN=*) :: szCursor
00783          INTEGER (SQLSMALLINT) :: cbCursorMax, pcbCursor
00786       END FUNCTION SQLGetCursorName
00787    END INTERFACE
00788
00789
00790    INTERFACE SQLGetData
00791       !   SQLRETURN SQLGetData(
00792       !     SQLHSTMT     StatementHandle,
00793       !     SQLUSMALLINT     ColumnNumber,
00794       !     SQLSMALLINT     TargetType,
00795       !     SQLPOINTER     TargetValuePtr,
00796       !     SQLINTEGER     BufferLength,
00797       !     SQLLEN *     StrLen_or_IndPtr);
00798       !
00799       FUNCTION SQLGetDataChar( stmt, icol, fCType,   &
00800                                rgbValue, cbValueMax, pcbValue )
00801       ! rgbValue is a CHARACTER buffer
00802          USE qtODBCKinds
00803          INTEGER (SQLRETURN) :: SQLGetDataChar
00804          INTEGER (SQLHSTMT) :: stmt
00805          INTEGER (SQLUSMALLINT) :: icol
00806          CHARACTER (LEN=*) :: rgbValue
00807          INTEGER (SQLSMALLINT) :: fCType
00808          !2011-08-08# INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00809          INTEGER (SQLLEN) :: cbValueMax, pcbValue
00812       END FUNCTION SQLGetDataChar
00813
00814       FUNCTION SQLGetDataI2( stmt, icol, fCType,   &
00815                              rgbValue, cbValueMax, pcbValue )
00816       ! rgbValue is an INTEGER(SHORT) value
00817          USE qtODBCKinds
00818          INTEGER (SQLRETURN) :: SQLGetDataI2
00819          INTEGER (SQLHSTMT) :: stmt
```

```
00820          INTEGER (SQLUSMALLINT) :: icol
00821          INTEGER(SHORT) rgbValue
00822          INTEGER (SQLSMALLINT) :: fCType
00823          !2011-08-08# INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00824          INTEGER (SQLLEN) :: cbValueMax, pcbValue
00827       END FUNCTION SQLGetDataI2
00828
00829       FUNCTION SQLGetDataI4( stmt, icol, fCType,   &
00830                              rgbValue, cbValueMax, pcbValue )
00831       ! rgbValue is an INTEGER(LONG) value
00832          USE qtODBCKinds
00833          INTEGER (SQLRETURN) :: SQLGetDataI4
00834          INTEGER (SQLHSTMT) :: stmt
00835          INTEGER (SQLUSMALLINT) :: icol
00836          INTEGER(LONG) rgbValue
00837          INTEGER (SQLSMALLINT) :: fCType
00838          !2011-08-08# INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00839          INTEGER (SQLLEN) :: cbValueMax, pcbValue
00842       END FUNCTION SQLGetDataI4
00843
00844       FUNCTION SQLGetDataR4( stmt, icol, fCType,   &
00845                              rgbValue, cbValueMax, pcbValue )
00846       ! rgbValue is a REAL(FLOAT) value
00847          USE qtODBCKinds
00848          INTEGER (SQLRETURN) :: SQLGetDataR4
00849          INTEGER (SQLHSTMT) :: stmt
00850          INTEGER (SQLUSMALLINT) :: icol
00851          REAL(FLOAT) rgbValue
00852          INTEGER (SQLSMALLINT) :: fCType
00853          !2011-08-08# INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00854          INTEGER (SQLLEN) :: cbValueMax, pcbValue
00857       END FUNCTION SQLGetDataR4
00858
00859       FUNCTION SQLGetDataDP( stmt, icol, fCType,   &
00860                              rgbValue, cbValueMax, pcbValue )
00861       ! rgbValue is a DOUBLE PRECISION value
00862          USE qtODBCKinds
00863          INTEGER (SQLRETURN) :: SQLGetDataDP
00864          INTEGER (SQLHSTMT) :: stmt
00865          INTEGER (SQLUSMALLINT) :: icol
00866          DOUBLE PRECISION rgbValue
00867          INTEGER (SQLSMALLINT) :: fCType
00868          !2011-08-08# INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00869          INTEGER (SQLLEN) :: cbValueMax, pcbValue
00872       END FUNCTION SQLGetDataDP
00873
00874    END INTERFACE SQLGetData
00875
00876
00877    INTERFACE SQLGetDescField
00878       ! ValuePtr is a CHARACTER buffer
00879       FUNCTION SQLGetDescFieldChar( DescriptorHandle, RecNumber, FieldIdentifier,  &
00880                                     ValuePtr, LenValuePtr, ValuePtrLen )
00881          USE qtODBCKinds
00883          INTEGER (SQLRETURN) :: SQLGetDescFieldChar
00884          INTEGER (SQLHDESC) :: DescriptorHandle
00885          INTEGER (SQLSMALLINT) :: RecNumber, FieldIdentifier
00886          CHARACTER (LEN=*) :: ValuePtr
00887          INTEGER (SQLINTEGER) :: LenValuePtr, ValuePtrLen
00889       END FUNCTION SQLGetDescFieldChar
00890
00891       ! ValuePtr is a pointer
00892       FUNCTION SQLGetDescField( DescriptorHandle, RecNumber, FieldIdentifier,  &
00893                                 ValuePtr, LenValuePtr, ValuePtrLen )
00894          USE qtODBCKinds
00896          INTEGER (SQLRETURN) :: SQLGetDescField
00897          INTEGER (SQLHDESC) :: DescriptorHandle
00898          INTEGER (SQLSMALLINT) :: RecNumber, FieldIdentifier
00899          INTEGER (SQLPOINTER) :: ValuePtr
00900          INTEGER (SQLINTEGER) :: LenValuePtr, ValuePtrLen
00902       END FUNCTION SQLGetDescField
00903    END INTERFACE SQLGetDescField
00904
00905    INTERFACE SQLGetDescRec
00906       FUNCTION SQLGetDescRec( DescriptorHandle, RecNumber, DescName,  &
00907                               LenDescName, DescNameLength, TypePtr, SubTypePtr,   &
00908                               LengthPtr, PrecisionPtr, ScalePtr, NullablePtr )
00909          USE qtODBCKinds
00911          INTEGER (SQLRETURN) :: SQLGetDescRec
00912          INTEGER (SQLHDESC) :: DescriptorHandle
00913          INTEGER (SQLSMALLINT) :: RecNumber, LenDescName, DescNameLength,  &
00914                                   TypePtr, SubTypePtr, PrecisionPtr, ScalePtr, NullablePtr
00915          !2011-08-08# INTEGER (SQLINTEGER) :: LengthPtr
00916          INTEGER (SQLLEN) :: LengthPtr
00917          CHARACTER (LEN=*) :: DescName
00920       END FUNCTION SQLGetDescRec
00921    END INTERFACE
00922
00923    INTERFACE SQLGetDiagField
00924       FUNCTION SQLGetDiagField( HandleType, Hndl, RecNumber, DiagIdentifier,  &
00925                                 DiagInfoPtr, LenDiagInfo, DiagInfoLen )
00926          USE qtODBCKinds
```

```
00928          INTEGER (SQLRETURN) :: SQLGetDiagField
00929          INTEGER (SQLSMALLINT) :: HandleType, RecNumber, DiagIdentifier,   &
00930                                   LenDiagInfo, DiagInfoLen
00931          INTEGER (SQLHANDLE) :: Hndl
00932          INTEGER (SQLPOINTER) ::  DiagInfoPtr
00934       END FUNCTION SQLGetDiagField
00935    END INTERFACE
00936
00937    INTERFACE SQLGetDiagRec
00938       FUNCTION SQLGetDiagRec( HandleType, Hndl, RecNumber, Sqlstate,  &
00939                               NativeError, MessageText, LenMsgText, MsgTextLen )
00940          USE qtODBCKinds
00942          INTEGER (SQLRETURN) :: SQLGetDiagRec
00943          INTEGER (SQLSMALLINT) :: HandleType, RecNumber, LenMsgText, MsgTextLen
00944          INTEGER (SQLHANDLE) :: Hndl
00945          CHARACTER (LEN=*) :: Sqlstate, MessageText
00946          INTEGER (SQLINTEGER) :: NativeError
00948       END FUNCTION SQLGetDiagRec
00949    END INTERFACE
00950
00951    INTERFACE SQLGetEnvAttr
00952       ! Value is a CHARACTER buffer
00953       FUNCTION SQLGetEnvAttrChar( env, Attribute, Value, LenValue, ValueLength )
00954          USE qtODBCKinds
00956          INTEGER (SQLRETURN) :: SQLGetEnvAttrChar
00957          INTEGER (SQLHENV) :: env
00958          INTEGER (SQLINTEGER) :: Attribute, LenValue, ValueLength
00959          CHARACTER (LEN=*) :: Value
00961       END FUNCTION SQLGetEnvAttrChar
00962
00963       ! Value is an INTEGER
00964       FUNCTION SQLGetEnvAttrI4( env, Attribute, Value, LenValue, ValueLength )
00965          USE qtODBCKinds
00967          INTEGER (SQLRETURN) :: SQLGetEnvAttrI4
00968          INTEGER (SQLHENV) :: env
00969          INTEGER (SQLINTEGER) :: Attribute, LenValue, ValueLength
00970          INTEGER (SQLINTEGER) :: Value
00972       END FUNCTION SQLGetEnvAttrI4
00973    END INTERFACE SQLGetEnvAttr
00974
00975    INTERFACE SQLGetFunctions
00976       FUNCTION SQLGetFunctions( dbc, fFunction, pfExists )
00977          USE qtODBCKinds
00978          INTEGER (SQLRETURN) :: SQLGetFunctions
00979          INTEGER (SQLHDBC) :: dbc
00980          INTEGER (SQLUSMALLINT) :: fFunction, pfExists
00983       END FUNCTION SQLGetFunctions
00984    END INTERFACE
00985
00986    INTERFACE SQLGetInfo
00987       FUNCTION SQLGetInfoChar( dbc, fInfoType, rgbInfoValue,     &
00988                                cbInfoValueMax, pcbInfoValue )
00989          ! rgbInfoValue is a CHARACTER buffer
00990          USE qtODBCKinds
00991          INTEGER (SQLRETURN) :: SQLGetInfoChar
00992          INTEGER (SQLHDBC) :: dbc
00993          INTEGER (SQLUSMALLINT) :: fInfoType
00994          CHARACTER (LEN=*) :: rgbInfoValue
00995          INTEGER (SQLSMALLINT) :: cbInfoValueMax, pcbInfoValue
00998       END FUNCTION SQLGetInfoChar
00999
01000       FUNCTION SQLGetInfoI2( dbc, fInfoType, rgbInfoValue,    &
01001                              cbInfoValueMax, pcbInfoValue )
01002          ! rgbInfoValue is of type INTEGER(SHORT)
01003          USE qtODBCKinds
01004          INTEGER (SQLRETURN) :: SQLGetInfoI2
01005          INTEGER (SQLHDBC) :: dbc
01006          INTEGER (SQLUSMALLINT) :: fInfoType
01007          INTEGER(SHORT) rgbInfoValue
01008          INTEGER (SQLSMALLINT) :: cbInfoValueMax, pcbInfoValue
01011       END FUNCTION SQLGetInfoI2
01012
01013       FUNCTION SQLGetInfoI4( dbc, fInfoType, rgbInfoValue,    &
01014                              cbInfoValueMax, pcbInfoValue )
01015          ! rgbInfoValue is of type INTEGER(LONG)
01016          USE qtODBCKinds
01017          INTEGER (SQLRETURN) :: SQLGetInfoI4
01018          INTEGER (SQLHDBC) :: dbc
01019          INTEGER (SQLUSMALLINT) :: fInfoType
01020          INTEGER(LONG) rgbInfoValue
01021          INTEGER (SQLSMALLINT) :: cbInfoValueMax, pcbInfoValue
01024       END FUNCTION SQLGetInfoI4
01025    END INTERFACE SQLGetInfo
01026
01027    INTERFACE SQLGetStmtAttr
01028       ! Value is a CHARACTER buffer
01029       FUNCTION SQLGetStmtAttrChar( stmt, Attribute, Value, LenValue, ValueLength )
01030          USE qtODBCKinds
01032          INTEGER (SQLRETURN) :: SQLGetStmtAttrChar
01033          INTEGER (SQLHSTMT) :: stmt
01034          INTEGER (SQLINTEGER) :: Attribute, LenValue, ValueLength
01035          CHARACTER (LEN=*) :: Value
```

```
01037       END FUNCTION SQLGetStmtAttrChar
01038
01039       ! Value is a pointer to a buffer
01040       FUNCTION SQLGetStmtAttr( stmt, Attribute, ValuePtr, LenValue, ValueLength )
01041          USE qtODBCKinds
01043          INTEGER (SQLRETURN) :: SQLGetStmtAttr
01044          INTEGER (SQLHSTMT) :: stmt
01045          INTEGER (SQLINTEGER) :: Attribute, LenValue, ValueLength
01046          INTEGER (SQLPOINTER) :: ValuePtr
01048       END FUNCTION SQLGetStmtAttr
01049    END INTERFACE SQLGetStmtAttr
01050
01051    INTERFACE SQLGetStmtOption
01052       FUNCTION SQLGetStmtOptionChar( stmt, fOption, pvParam )
01053       ! pvParam is a CHARACTER buffer
01054          USE qtODBCKinds
01055          INTEGER (SQLRETURN) :: SQLGetStmtOptionChar
01056          INTEGER (SQLHSTMT) :: stmt
01057          INTEGER (SQLUSMALLINT) :: fOption
01058          CHARACTER (LEN=*) :: pvParam
01061       END FUNCTION SQLGetStmtOptionChar
01062
01063       FUNCTION SQLGetStmtOptionI4( stmt, fOption, pvParam )
01064       ! pvParam is an INTEGER(LONG) value
01065          USE qtODBCKinds
01066          INTEGER (SQLRETURN) :: SQLGetStmtOptionI4
01067          INTEGER (SQLHSTMT) :: stmt
01068          INTEGER (SQLUSMALLINT) :: fOption
01069          INTEGER(LONG) pvParam
01072       END FUNCTION SQLGetStmtOptionI4
01073    END INTERFACE SQLGetStmtOption
01074
01075    INTERFACE SQLGetTypeInfo
01076       FUNCTION SQLGetTypeInfo( stmt, fSqlType )
01077          USE qtODBCKinds
01078          INTEGER (SQLRETURN) :: SQLGetTypeInfo
01079          INTEGER (SQLHSTMT) :: stmt
01080          INTEGER (SQLSMALLINT) :: fSqlType
01082       END FUNCTION SQLGetTypeInfo
01083    END INTERFACE
01084
01085    INTERFACE SQLMoreResults
01086       FUNCTION SQLMoreResults( stmt )
01087          USE qtODBCKinds
01088          INTEGER (SQLRETURN) :: SQLMoreResults
01089          INTEGER (SQLHSTMT) :: stmt
01091       END FUNCTION SQLMoreResults
01092    END INTERFACE
01093
01094    INTERFACE SQLNativeSql
01095       FUNCTION SQLNativeSql( dbc, szSqlStrIn, cbSqlStrIn,  &
01096                              szSqlStr, cbSqlStrMax, pcbSqlStr )
01097          USE qtODBCKinds
01098          INTEGER (SQLRETURN) :: SQLNativeSql
01099          INTEGER (SQLHDBC) :: dbc
01100          CHARACTER (LEN=*) :: szSqlStrIn, szSqlStr
01101          INTEGER (SQLINTEGER) :: cbSqlStrIn, cbSqlStrMax, pcbSqlStr
01104       END FUNCTION SQLNativeSql
01105    END INTERFACE
01106
01107    INTERFACE SQLNumParams
01108       FUNCTION SQLNumParams( stmt, pcpar )
01109          USE qtODBCKinds
01110          INTEGER (SQLRETURN) :: SQLNumParams
01111          INTEGER (SQLHSTMT) :: stmt
01112          INTEGER (SQLSMALLINT) :: pcpar
01115       END FUNCTION SQLNumParams
01116    END INTERFACE
01117
01118    INTERFACE SQLNumResultCols
01119       FUNCTION SQLNumResultCols( stmt, pccol )
01120          USE qtODBCKinds
01121          INTEGER (SQLRETURN) :: SQLNumResultCols
01122          INTEGER (SQLHSTMT) :: stmt
01123          INTEGER (SQLSMALLINT) :: pccol
01126       END FUNCTION SQLNumResultCols
01127    END INTERFACE
01128
01129    INTERFACE SQLParamData
01130       FUNCTION SQLParamDataChar( stmt, prgbValue )
01131       ! prgbValue is a CHARACTER buffer
01132          USE qtODBCKinds
01133          INTEGER (SQLRETURN) :: SQLParamDataChar
01134          INTEGER (SQLHSTMT) :: stmt
01135          CHARACTER (LEN=*) :: prgbValue
01138       END FUNCTION SQLParamDataChar
01139
01140       FUNCTION SQLParamDataI2( stmt, prgbValue )
01141       ! prgbValue is an INTEGER(SHORT) value
01142          USE qtODBCKinds
01143          INTEGER (SQLRETURN) :: SQLParamDataI2
```

```
01144          INTEGER (SQLHSTMT) :: stmt
01145          INTEGER(SHORT) prgbValue
01148       END FUNCTION SQLParamDataI2
01149
01150       FUNCTION SQLParamDataI4( stmt, prgbValue )
01151       ! prgbValue is an INTEGER(LONG) value
01152          USE qtODBCKinds
01153          INTEGER (SQLRETURN) :: SQLParamDataI4
01154          INTEGER (SQLHSTMT) :: stmt
01155          INTEGER(LONG) prgbValue
01158       END FUNCTION SQLParamDataI4
01159
01160       FUNCTION SQLParamDataR4( stmt, prgbValue )
01161       ! prgbValue is an REAL(FLOAT) value
01162          USE qtODBCKinds
01163          INTEGER (SQLRETURN) :: SQLParamDataR4
01164          INTEGER (SQLHSTMT) :: stmt
01165          REAL(FLOAT) prgbValue
01168       END FUNCTION SQLParamDataR4
01169
01170       FUNCTION SQLParamDataDP( stmt, prgbValue )
01171       ! prgbValue is an DOUBLE PRECISION value
01172          USE qtODBCKinds
01173          INTEGER (SQLRETURN) :: SQLParamDataDP
01174          INTEGER (SQLHSTMT) :: stmt
01175          DOUBLE PRECISION prgbValue
01178       END FUNCTION SQLParamDataDP
01179    END INTERFACE SQLParamData
01180
01181    INTERFACE SQLParamOptions
01182       FUNCTION SQLParamOptions( stmt, crow, pirow )
01183          USE qtODBCKinds
01185          INTEGER (RETCODE) :: SQLParamOptions
01186          INTEGER (HSTMT) :: stmt
01187          !2011-08-08# INTEGER (UDWORD) :: crow, pirow
01188          INTEGER (SQLULEN ) :: crow, pirow
01190       END FUNCTION SQLParamOptions
01191    END INTERFACE
01192
01193    INTERFACE SQLPrepare
01194       FUNCTION SQLPrepare( stmt, szSqlStr, cbSqlStr )
01195          USE qtODBCKinds
01196          INTEGER (SQLRETURN) :: SQLPrepare
01197          INTEGER (SQLHSTMT) :: stmt
01198          CHARACTER (LEN=*) :: szSqlStr
01199          INTEGER (SQLINTEGER) :: cbSqlStr
01202       END FUNCTION SQLPrepare
01203    END INTERFACE
01204
01205    INTERFACE SQLPrimaryKeys
01206       FUNCTION SQLPrimaryKeys( stmt, CatalogName, CatNameLength,   &
01207                                      SchemaName, SchemaNameLength, &
01208                                      TableName, TableNameLength )
01209          USE qtODBCKinds
01211          INTEGER (SQLRETURN) :: SQLPrimaryKeys
01212          INTEGER (SQLHSTMT) :: stmt
01213          CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName
01214          INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength, TableNameLength
01216       END FUNCTION SQLPrimaryKeys
01217    END INTERFACE
01218
01219    INTERFACE SQLProcedureColumns
01220       FUNCTION SQLProcedureColumns( stmt, CatalogName, CatNameLength,   &
01221                                           SchemaName, SchemaNameLength, &
01222                                           ProcName, ProcNameLength,     &
01223                                           ColumnName, ColNameLength )
01224          USE qtODBCKinds
01226          INTEGER (SQLRETURN) :: SQLProcedureColumns
01227          INTEGER (SQLHSTMT) :: stmt
01228          CHARACTER (LEN=*) :: CatalogName, SchemaName, ProcName, ColumnName
01229          INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength,   &
01230                                   ProcNameLength, ColNameLength
01232       END FUNCTION SQLProcedureColumns
01233    END INTERFACE
01234
01235    INTERFACE SQLProcedures
01236       FUNCTION SQLProcedures( stmt, CatalogName, CatNameLength,   &
01237                                     SchemaName, SchemaNameLength, &
01238                                     ProcName, ProcNameLength )
01239          USE qtODBCKinds
01241          INTEGER (SQLRETURN) :: SQLProcedures
01242          INTEGER (SQLHSTMT) :: stmt
01243          CHARACTER (LEN=*) :: CatalogName, SchemaName, ProcName
01244          INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength, ProcNameLength
01246       END FUNCTION SQLProcedures
01247    END INTERFACE
01248
01249    INTERFACE SQLPutData
01250       FUNCTION SQLPutDataChar( stmt, rgbValue, cbValue )
01251       ! rgbValue is a CHARACTER buffer
01252          USE qtODBCKinds
```

```
01253          INTEGER (SQLRETURN) :: SQLPutDataChar
01254          INTEGER (SQLHSTMT) :: stmt
01255          CHARACTER (LEN=*) :: rgbValue
01256          !2011-08-08# INTEGER (SQLINTEGER) :: cbValue
01257          INTEGER (SQLLEN ) :: cbValue
01260       END FUNCTION SQLPutDataChar
01261
01262       FUNCTION SQLPutDataI2( stmt, rgbValue, cbValue )
01263       ! rgbValue is an INTEGER(SHORT) value
01264          USE qtODBCKinds
01265          INTEGER (SQLRETURN) :: SQLPutDataI2
01266          INTEGER (SQLHSTMT) :: stmt
01267          INTEGER(SHORT) rgbValue
01268          !2011-08-08# INTEGER (SQLINTEGER) :: cbValue
01269          INTEGER (SQLLEN ) :: cbValue
01272       END FUNCTION SQLPutDataI2
01273
01274       FUNCTION SQLPutDataI4( stmt, rgbValue, cbValue )
01275       ! rgbValue is an INTEGER(LONG) value
01276          USE qtODBCKinds
01277          INTEGER (SQLRETURN) :: SQLPutDataI4
01278          INTEGER (SQLHSTMT) :: stmt
01279          INTEGER(LONG) rgbValue
01280          !2011-08-08# INTEGER (SQLINTEGER) :: cbValue
01281          INTEGER (SQLLEN ) :: cbValue
01284       END FUNCTION SQLPutDataI4
01285
01286       FUNCTION SQLPutDataR4( stmt, rgbValue, cbValue )
01287       ! rgbValue is an REAL(FLOAT) value
01288          USE qtODBCKinds
01289          INTEGER (SQLRETURN) :: SQLPutDataR4
01290          INTEGER (SQLHSTMT) :: stmt
01291          REAL(FLOAT) rgbValue
01292          !2011-08-08# INTEGER (SQLINTEGER) :: cbValue
01293          INTEGER (SQLLEN ) :: cbValue
01296       END FUNCTION SQLPutDataR4
01297
01298       FUNCTION SQLPutDataDP( stmt, rgbValue, cbValue )
01299       ! rgbValue is an DOUBLE PRECISION value
01300          USE qtODBCKinds
01301          INTEGER (SQLRETURN) :: SQLPutDataDP
01302          INTEGER (SQLHSTMT) :: stmt
01303          DOUBLE PRECISION rgbValue
01304          !2011-08-08# INTEGER (SQLINTEGER) :: cbValue
01305          INTEGER (SQLLEN ) :: cbValue
01308       END FUNCTION SQLPutDataDP
01309    END INTERFACE SQLPutData
01310
01311    INTERFACE SQLRowCount
01312       FUNCTION SQLRowCount( stmt, pcrow )
01313          USE qtODBCKinds
01314          INTEGER (SQLRETURN) :: SQLRowCount
01315          INTEGER (SQLHSTMT) :: stmt
01316          !2011-08-08# INTEGER (SQLINTEGER) :: pcrow
01317          INTEGER (SQLLEN) :: pcrow
01320       END FUNCTION SQLRowCount
01321    END INTERFACE
01322
01323    INTERFACE SQLSetConnectAttr
01324       FUNCTION SQLSetConnectAttrLP( dbc, Attribute, ValuePtr, StringLength )
01325          USE qtODBCKinds
01326          INTEGER (SQLRETURN) :: SQLSetConnectAttrLP
01327          INTEGER (SQLHDBC) :: dbc
01328          INTEGER (SQLINTEGER) :: Attribute
01329          INTEGER (SQLPOINTER) :: ValuePtr
01330          INTEGER (SQLINTEGER) :: StringLength
01332       END FUNCTION SQLSetConnectAttrLP
01333
01334       FUNCTION SQLSetConnectAttrChar( dbc, Attribute, ValuePtr, StringLength )
01335       ! ValuePtr is a zero terminated string
01336          USE qtODBCKinds
01337          INTEGER (SQLRETURN) :: SQLSetConnectAttrChar
01338          INTEGER (SQLHDBC) :: dbc
01339          INTEGER (SQLINTEGER) :: Attribute
01340          CHARACTER (LEN=*) :: ValuePtr
01341          INTEGER (SQLINTEGER) :: StringLength
01344       END FUNCTION SQLSetConnectAttrChar
01345    END INTERFACE
01346
01347    INTERFACE SQLSetConnectOption
01348       FUNCTION SQLSetConnectOption( dbc, fOption, vParam )
01349          USE qtODBCKinds
01350          INTEGER (SQLRETURN) :: SQLSetConnectOption
01351          INTEGER (SQLHDBC) :: dbc
01352          INTEGER (SQLUSMALLINT) :: fOption
01353          !2011-08-08# INTEGER (SQLUINTEGER) :: vParam
01354          INTEGER (SQLULEN ) :: vParam
01356       END FUNCTION SQLSetConnectOption
01357    END INTERFACE
01358
01359    INTERFACE SQLSetCursorName
01360       FUNCTION SQLSetCursorName( stmt, szCursor, cbCursor )
```

```
01361          USE qtODBCKinds
01362          INTEGER (SQLRETURN) :: SQLSetCursorName
01363          INTEGER (SQLHSTMT) :: stmt
01364          CHARACTER (LEN=*) :: szCursor
01365          INTEGER (SQLSMALLINT) :: cbCursor
01368       END FUNCTION SQLSetCursorName
01369    END INTERFACE
01370
01371    INTERFACE SQLSetDescField
01372       ! ValuePtr is a CHARACTER buffer
01373       FUNCTION SQLSetDescFieldChar( DescriptorHandle, RecNumber, FieldIdentifier,  &
01374                                     ValuePtr, LenValuePtr )
01375          USE qtODBCKinds
01377          INTEGER (SQLRETURN) :: SQLSetDescFieldChar
01378          INTEGER (SQLHDESC) :: DescriptorHandle
01379          INTEGER (SQLSMALLINT) :: RecNumber, FieldIdentifier
01380          CHARACTER (LEN=*) :: ValuePtr
01382          INTEGER (SQLINTEGER) :: LenValuePtr
01383       END FUNCTION SQLSetDescFieldChar
01384
01385       ! ValuePtr is a pointer
01386       FUNCTION SQLSetDescField( DescriptorHandle, RecNumber, FieldIdentifier,  &
01387                                 ValuePtr, LenValuePtr )
01388          USE qtODBCKinds
01390          INTEGER (SQLRETURN) :: SQLSetDescField
01391          INTEGER (SQLHDESC) :: DescriptorHandle
01392          INTEGER (SQLSMALLINT) :: RecNumber, FieldIdentifier
01393          INTEGER (SQLPOINTER) :: ValuePtr
01394          INTEGER (SQLINTEGER) :: LenValuePtr
01395       END FUNCTION SQLSetDescField
01396    END INTERFACE SQLSetDescField
01397
01398    INTERFACE SQLSetDescRec
01399       FUNCTION SQLSetDescRec( DescriptorHandle, RecNumber, ValType, SubType,   &
01400                               fldLength, PrecVal, ScaleVal, DataPtr,       &
01401                               StringLength, Indicator )
01402          USE qtODBCKinds
01404          INTEGER (SQLRETURN) :: SQLSetDescRec
01405          INTEGER (SQLHDESC) :: DescriptorHandle
01406          INTEGER (SQLSMALLINT) :: RecNumber, ValType, SubType, PrecVal, ScaleVal, NullablePtr
01407          !2011-08-08# INTEGER (SQLINTEGER) :: fldLength, StringLength, Indicator
01408          INTEGER (SQLINTEGER) :: fldLength
01409          INTEGER (SQLLEN) :: StringLength, Indicator
01410          INTEGER (SQLPOINTER) :: DataPtr
01412       END FUNCTION SQLSetDescRec
01413    END INTERFACE
01414
01415    INTERFACE SQLSetEnvAttr
01416       FUNCTION SQLSetEnvAttrI4( env, Attribute, ValuePtr, StringLength )      ! corr. 12.10.2000:
SQLSetEnvAttr -> SQLSetEnvAttrI4
01416-1          ttr -> SQLSetEnvAttrI4
01417          ! ValuePtr is a 32-bit unsigned integer value
01418          USE qtODBCKinds
01419          INTEGER (SQLRETURN) :: SQLSetEnvAttrI4
01420          INTEGER (SQLHENV) :: env
01421          INTEGER (SQLINTEGER) :: Attribute
01422          INTEGER (SQLPOINTER) :: ValuePtr
01423          INTEGER (SQLINTEGER) :: StringLength
01425       END FUNCTION SQLSetEnvAttrI4
01426
01427       FUNCTION SQLSetEnvAttrChar( env, Attribute, ValuePtr, StringLength )
01428       ! ValuePtr is a zero terminated string
01429          USE qtODBCKinds
01430          INTEGER (SQLRETURN) :: SQLSetEnvAttrChar
01431          INTEGER (SQLHENV) :: env
01432          INTEGER (SQLINTEGER) :: Attribute
01433          CHARACTER (LEN=*) :: ValuePtr
01434          INTEGER (SQLINTEGER) :: StringLength
01437       END FUNCTION SQLSetEnvAttrChar
01438    END INTERFACE SQLSetEnvAttr
01439
01440    INTERFACE
01441       FUNCTION SQLSetPos( stmt, irow, fOption, fLock )
01442          USE qtODBCKinds
01443          INTEGER (SQLRETURN) :: SQLSetPos
01444          INTEGER (SQLHSTMT) :: stmt
01445          !2011-08-08# INTEGER (SQLUSMALLINT) :: irow, fOption, fLock
01446          INTEGER (SQLSETPOSIROW) :: irow
01447          INTEGER (SQLUSMALLINT) :: fOption, fLock
01449       END FUNCTION SQLSetPos
01450    END INTERFACE
01451
01452    INTERFACE SQLSetScrollOptions
01453       FUNCTION SQLSetScrollOptions( stmt, fConcurrency, crowKeyset, crowRowset )
01454          USE qtODBCKinds
01455          INTEGER (SQLRETURN) :: SQLSetScrollOptions
01456          INTEGER (SQLHSTMT) :: stmt
01457          INTEGER (SQLUSMALLINT) :: fConcurrency, crowRowset
01458          !2011-08-08# INTEGER (SQLINTEGER) :: crowKeyset
01459          INTEGER (SQLLEN) :: crowKeyset
01461       END FUNCTION SQLSetScrollOptions
01462    END INTERFACE
```

```
01463
01464    INTERFACE SQLSetStmtAttrChar
01465       FUNCTION SQLSetStmtAttrChar( stmt, Attribute, Value, LenValue )
01466       ! Value is a CHARACTER buffer
01467          USE qtODBCKinds
01469          INTEGER (SQLRETURN) :: SQLSetStmtAttrChar
01470          INTEGER (SQLHSTMT) :: stmt
01471          INTEGER (SQLINTEGER) :: Attribute, LenValue
01472          CHARACTER (LEN=*) :: Value
01474       END FUNCTION SQLSetStmtAttrChar
01475    END INTERFACE
01476
01477    INTERFACE SQLSetStmtAttrI4
01478       FUNCTION SQLSetStmtAttrI4( stmt, Attribute, Value, LenValue )
01479          ! Value is an INTEGER(LONG)
01480          USE qtODBCKinds
01482          INTEGER (SQLRETURN) :: SQLSetStmtAttrI4
01483          INTEGER (SQLHSTMT) :: stmt
01484          INTEGER (SQLINTEGER) :: Attribute, LenValue
01485          INTEGER(LONG) Value
01487       END FUNCTION SQLSetStmtAttrI4
01488    END INTERFACE
01489
01490    INTERFACE SQLSetStmtAttr
01491       FUNCTION SQLSetStmtAttr( stmt, Attribute, ValuePtr, LenValue )
01492          ! Value is a pointer to a buffer
01493          USE qtODBCKinds
01495          INTEGER (SQLRETURN) :: SQLSetStmtAttr
01496          INTEGER (SQLHSTMT) :: stmt
01497          INTEGER (SQLINTEGER) :: Attribute, LenValue
01498          INTEGER (SQLPOINTER) :: ValuePtr
01499       END FUNCTION SQLSetStmtAttr
01500    END INTERFACE
01501
01502    INTERFACE SQLSetStmtOption
01503       FUNCTION SQLSetStmtOption( stmt, fOption, vParam )
01504          USE qtODBCKinds
01505          INTEGER (SQLRETURN) :: SQLSetStmtOption
01506          INTEGER (SQLHSTMT) :: stmt
01507          INTEGER (SQLUSMALLINT) :: fOption
01508          !2011-08-08 INTEGER (SQLUINTEGER) :: vParam
01509          INTEGER (SQLULEN) :: vParam
01511       END FUNCTION SQLSetStmtOption
01512    END INTERFACE
01513
01514    INTERFACE SQLSpecialColumns
01515       FUNCTION SQLSpecialColumns( stmt, IdentifierType,  &
01516                                   CatalogName, CatNameLength,    &
01517                                   SchemaName, SchemaNameLength,  &
01518                                   TableName, TableNameLength,    &
01519                                   Scope, Nullable)
01520          USE qtODBCKinds
01521          INTEGER (SQLRETURN) :: SQLSpecialColumns
01523          INTEGER (SQLHSTMT) :: stmt
01524          CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName
01526          INTEGER (SQLSMALLINT) :: IdentifierType, CatNameLength, SchemaNameLength,  &
01527                                   TableNameLength, Scope, Nullable
01528       END FUNCTION SQLSpecialColumns
01529    END INTERFACE
01530
01531    INTERFACE SQLStatistics
01532       FUNCTION SQLStatistics( stmt, CatalogName, CatNameLength,    &
01533                                  SchemaName, SchemaNameLength,  &
01534                                  TableName, TableNameLength,    &
01535                                  Unique, Reserved )
01536          USE qtODBCKinds
01537          INTEGER (SQLRETURN) :: SQLStatistics
01539          INTEGER (SQLHSTMT) :: stmt
01540          CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName
01542          INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength, TableNameLength
01543          INTEGER (SQLUSMALLINT) :: Unique, Reserved
01544       END FUNCTION SQLStatistics
01545    END INTERFACE
01546
01547    INTERFACE SQLTablePrivileges
01548       FUNCTION SQLTablePrivileges( stmt, CatalogName, CatNameLength,    &
01549                                       SchemaName, SchemaNameLength,  &
01550                                       TableName, TableNameLength )
01551          USE qtODBCKinds
01552          INTEGER (SQLRETURN) :: SQLTablePrivileges
01554          INTEGER (SQLHSTMT) :: stmt
01555          CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName
01557          INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength, TableNameLength
01558       END FUNCTION SQLTablePrivileges
01559    END INTERFACE
01560
01561   INTERFACE SQLTables
01562       FUNCTION SQLTables( stmt, szTableQualifier, cbTableQualifier,          &
01563                                 szTableOwner, cbTableOwner,                  &
01564                                 szTableName, cbTableName, szTableType, cbTableType )
01565          USE qtODBCKinds
```

```
01566          INTEGER (SQLRETURN) :: SQLTables
01567          INTEGER (SQLHSTMT) :: stmt
01568          CHARACTER (LEN=*) :: szTableQualifier, szTableOwner,  &
01569                               szTableName, szTableType
01570          INTEGER (SQLSMALLINT) :: cbTableQualifier, cbTableOwner,    &
01571                                   cbTableName, cbTableType
01575       END FUNCTION SQLTables
01576
01577       ! added 14.10.2000, case: all pointer variables to be treated as Values (use LOC() function to specify
01577-1           a pointer to a variable)
01578       FUNCTION SQLTablesLP( stmt, szTableQualifier, cbTableQualifier,          &
01579                                   szTableOwner, cbTableOwner,                  &
01580                                   szTableName, cbTableName, szTableType, cbTableType )
01581          USE qtODBCKinds
01582          INTEGER (SQLRETURN) :: SQLTablesLP
01583          INTEGER (SQLHSTMT) :: stmt
01584          INTEGER (LP) :: szTableQualifier, szTableOwner,  &
01585                          szTableName, szTableType
01586          INTEGER (SQLSMALLINT) :: cbTableQualifier, cbTableOwner,    &
01587                                   cbTableName, cbTableType
01589       END FUNCTION SQLTablesLP
01590    END INTERFACE SQLTables
01591
01592    INTERFACE SQLTransact
01593       FUNCTION SQLTransact( env, dbc, fType )
01594          USE qtODBCKinds
01595          INTEGER (SQLRETURN) :: SQLTransact
01596          INTEGER (SQLHENV) :: env
01597          INTEGER (SQLHDBC) :: dbc
01598          INTEGER (SQLUSMALLINT) :: fType
01600       END FUNCTION SQLTransact
01601    END INTERFACE
01602
01603 END MODULE qtODBCInterfaces
01604 !================================================================
```

# Index

## T



## U



## V



## W



## Z