Jörg Kuthe

ForDBC

Fortran Database Connectivity

Revision date: 8th of February 2008

QT software GmbH
Konstanzer Str. 10
D-10707 Berlin
Germany

Telefon 030/9290087-0
Telefax 030/9290087-2

http://www.qtsoftware.de
eMail info@qtsoftware.de

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". 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 followingly, 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

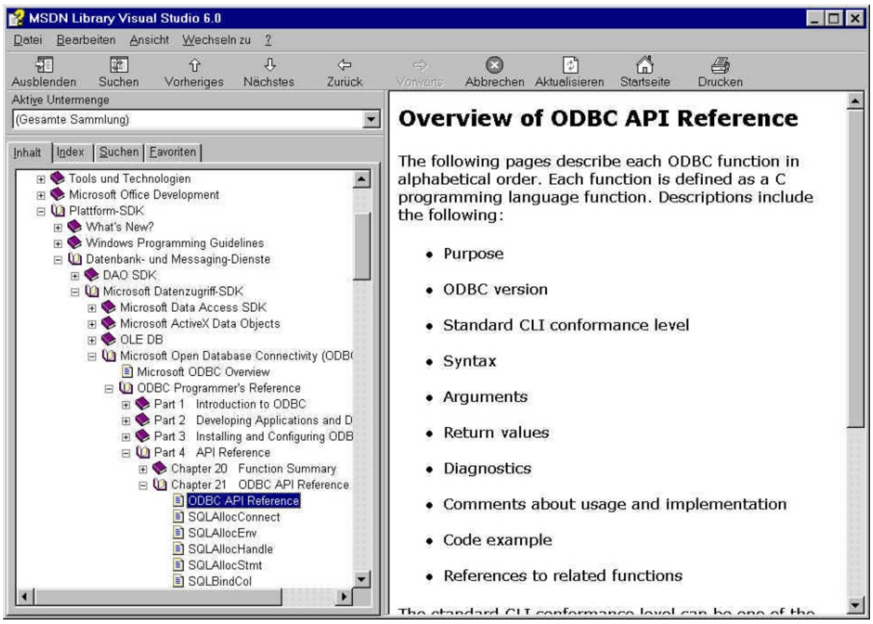
The author and QT software GmbH acknowledge the rights of the owners at the brand names, trademarks and products named in this document:
Excel is a trademark of Microsoft Corporation, U.S.A..

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 witha 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.


MSDN Library Visual Studio 6.0
Datei
Bearbeiten
Ansicht
Wechseln zu
?
Ausblenden
Suchen
Vorheriges
Nächstes
Zurück
Vorwärts
Abbrechen
Aktualisieren
Startseite
Drucken
Aktive Untermenge
(Gesamte Sammlung)
Inhalt
Index
Suchen
Favoriten
Tools und Technologien
Microsoft Office Development
Plattform-SDK
What's New?
Windows Programming Guidelines
Datenbank- und Messaging-Dienste
DAO SDK
Microsoft Datenzugriff-SDK
Microsoft Data Access SDK
Microsoft ActiveX Data Objects
OLE DB
Microsoft Open Database Connectivity (ODB
Microsoft ODBC Overview
ODBC Programmer's Reference
Part 1 Introduction to ODBC
Part 2 Developing Applications and D
Part 3 Installing and Configuring ODB
Part 4 API Reference
Chapter 20 Function Summary
Chapter 21 ODBC API Reference
ODBC API Reference
SQLAllocConnect
SQLAllocEnv
SQLAllocHandle
SQLAllocStmt
SQLBindCol

Overview of ODBC API Reference

The following pages describe each ODBC function in alphabetical order. Each function is defined as a C programming language function. Descriptions include the following:

- Purpose
- ODBC version
- Standard CLI conformance level
- Syntax
- Arguments
- Return values
- Diagnostics
- Comments about usage and implementation
- Code example
- References to related functions

The standard CLI conformance level can be one of the

Illus. 1: The description of ODBC API in MSDN [ODBC98].

This description can also be found on the Internet:

⇨ http://msdn.microsoft.com/library/

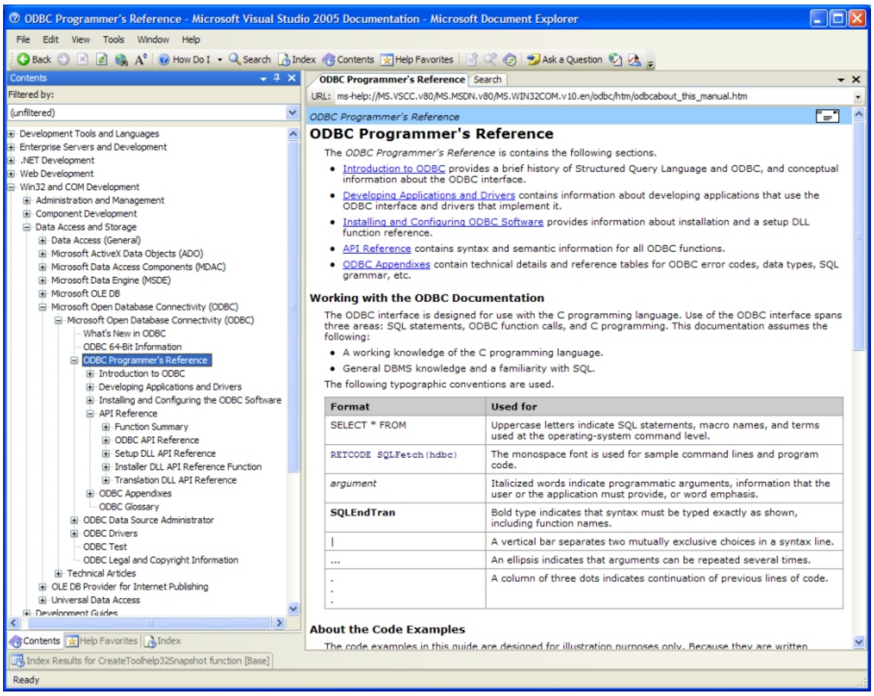
The main objectives of this introduction into ODBC programming with Fortran 90 respectively Fortran 95 (abbreviated Fortran9x in the further documentation) is

- 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.


ODBC Programmer's Reference - Microsoft Visual Studio 2005 Documentation - Microsoft Document Explorer
File Edit View Tools Window Help
Back
How Do I
Search
Index
Contents
Help Favorites
Ask a Question
Contents
Filtered by:
(unfiltered)
Development Tools and Languages
Enterprise Servers and Development
.NET Development
Web Development
Win32 and COM Development
Administration and Management
Component Development
Data Access and Storage
Data Access (General)
Microsoft ActiveX Data Objects (ADO)
Microsoft Data Access Components (MDAC)
Microsoft Data Engine (MSDE)
Microsoft OLE DB
Microsoft Open Database Connectivity (ODBC)
Microsoft Open Database Connectivity (ODBC)
What's New in ODBC
ODBC 64-Bit Information
ODBC Programmer's Reference
Introduction to ODBC
Developing Applications and Drivers
Installing and Configuring the ODBC Software
API Reference
Function Summary
ODBC API Reference
Setup DLL API Reference
Installer DLL API Reference Function
Translation DLL API Reference
ODBC Appendixes
ODBC Glossary
ODBC Data Source Administrator
ODBC Drivers
ODBC Test
ODBC Legal and Copyright Information
Technical Articles
OLE DB Provider for Internet Publishing
Universal Data Access
Development Guides
Contents
Help Favorites
Index
Index Results for CreateToolhelp32Snapshot function [Base]
Ready
ODBC Programmer's Reference
Search
URL: ms-help://MS.VSCC.v80/MS.MSDN.v80/MS.WIN32COM.v10.en/odbc/htm/odbcabout_this_manual.htm
ODBC Programmer's Reference

ODBC Programmer's Reference

The *ODBC Programmer's Reference* is contains the following sections.

- Introduction to ODBC provides a brief history of Structured Query Language and ODBC, and conceptual information about the ODBC interface.
- Developing Applications and Drivers contains information about developing applications that use the ODBC interface and drivers that implement it.
- Installing and Configuring ODBC Software provides information about installation and a setup DLL function reference.
- API Reference contains syntax and semantic information for all ODBC functions.
- ODBC Appendixes contain technical details and reference tables for ODBC error codes, data types, SQL grammar, etc.

Working with the ODBC Documentation

The ODBC interface is designed for use with the C programming language. Use of the ODBC interface spans three areas: SQL statements, ODBC function calls, and C programming. This documentation assumes the following:

- A working knowledge of the C programming language.
- General DBMS knowledge and a familiarity with SQL.

The following typographic conventions are used.

Format	Used for
SELECT * FROM	Uppercase letters indicate SQL statements, macro names, and terms used at the operating-system command level.
`RETCODE SQLFetch(hdbc)`	The monospace font is used for sample command lines and program code.
argument	Italicized words indicate programmatic arguments, information that the user or the application must provide, or word emphasis.
SQLEndTran	Bold type indicates that syntax must be typed exactly as shown, including function names.
|	A vertical bar separates two mutually exclusive choices in a syntax line.
...	An ellipsis indicates that arguments can be repeated several times.
. . .	A column of three dots indicates continuation of previous lines of code.

About the Code Examples

The code examples in this guide are designed for illustration purposes only. Because they are written

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]. 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. for 32 bit ODBC: 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 95, 98, 2000, NT, XP, Vista 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,
- 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.

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 in the ODBC.INI file or 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 initialization file 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 these 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
```

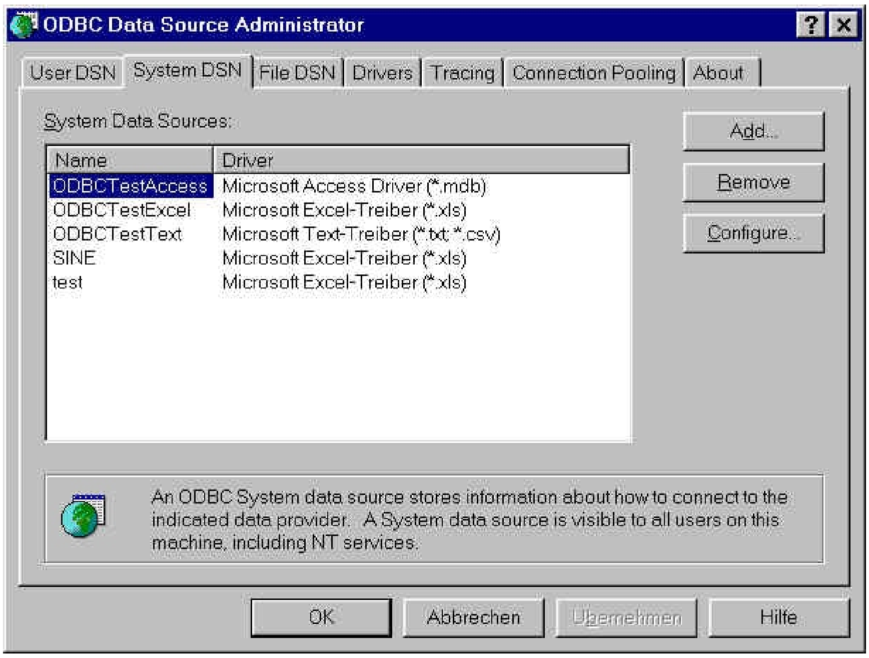
Hence, any data base for which an ODBC driver is installed can be used as a data source. In general, ODBC drivers are supplied with all well known databases, e.g. from Oracle, Sybase (SQL Anywhere and others), Informix, IBM (DB2) or Microsoft (MS/SQL). In addition, you can find ODBC drivers for MS/Access,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

Under Windows you start the ODBC administrator program (e.g. for 32 bit ODBC: 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").

ODBC (32bit)



System Data Sources:

Name	Driver
ODBCTestAccess	Microsoft Access Driver (*.mdb)
ODBCTestExcel	Microsoft Excel-Treiber (*.xls)
ODBCTestText	Microsoft Text-Treiber (*.txt; *.csv)
SINE	Microsoft Excel-Treiber (*.xls)
test	Microsoft Excel-Treiber (*.xls)

An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users on this machine, including NT services.

Illus. 3: View of tab „System DSN“ in 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. See the sample program T_ODBCExcel.f90.

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

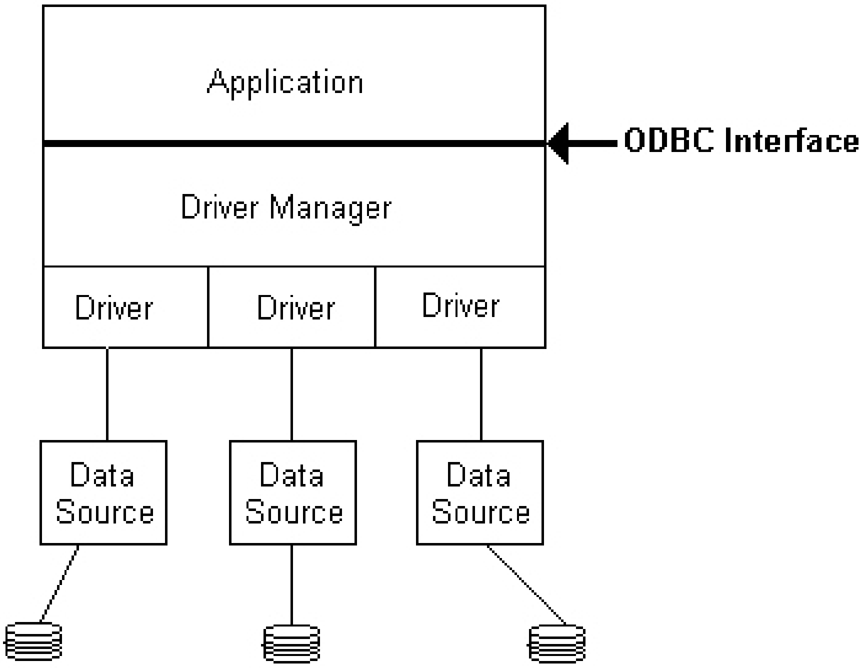


Abb. 4: Grundsätzlicher Aufbau von ODBC.

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 applications 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 creating an

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

Necessary for the communication with the data source is a

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

With both numbers (*hEnv* and *hDBC*) the application can operate to 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. 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 of 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 90/95 programmers are provided with appropriate Fortran 9x modules which are integrated by means of the USE command:

```
USE qt_ODBC
```

The source code of the module is in the file

qt_ODBC.f90.

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

qt_ODBCKinds
(see file qt_ODBCKinds.f90)

and constants (PARAMETERs) in

qt_ODBCDefs
(see file qt_ODBCDefs.f90)

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

qt_CKinds
(see file qt_CKinds.f90)

and

qt_Win32Kinds
(see file qt_Win32Kinds.f90)

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

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

which is used by the module qt_Win32Kinds to define the data type LP:

```
INTEGER (KIND=LONG) :: LP  ! long pointer
PARAMETER (LP = LONG)
```

In qt_ODBCKinds and qt_ODBC thes data types are used to define ODBC specific data types and constants . E.g.:

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

This seems confusing and doubtful since finally all types, constants, variables are mapped on 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 qt_CKinds) easily causes a complete change of the derived data types (this case occurred for example at the change of Win16 to 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 qt_ODBC
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 SQLBindColI2, SQLBindColI4, SQLBindColChar etc.. Unfortunately due to compatibility reasons it was not possible to map these functions on 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 qt_ODBC
INTEGER (SQLINTEGER) :: iAttr
INTEGER (SQLPOINTER) :: lpAttr
INTEGER (SQLINTEGER) :: Value
  .
  .
iAttr = FALSE   ! Attribute = FALSE
lpAttr= LOC(iAttr )          ! LOC() returns address
rtc = SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT, &
                        lpAttr, 4 )
.
END
```

The example shows how the variable lpAttr (which is of type SQLPOINTER) obtains the memory location of the variable iAttr 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

`qt_ODBCInterfaces`
(see file qt_ODBCInterfaces.f90)

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

`qt_ODBC_Compiler`
(see files qt_ODBC_*compiler*.f90
with *compiler* = DVF, FTN, IVF, LF90, LF95 etc.)

This means that the module name (`qt_ODBC_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 is required** (e.g. ODBC32.LIB).

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 those arguments passed when calling the ODBC API functions. In our Fortran programs we use those 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 sting 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 character strings need not 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 will be always returned with a terminating null.

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 to. For this situation there isn't any equivalent in Fortran. 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. 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 yourself 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 `SQLHENV` (ODBC v3.x), respectively. Both types base on INTEGER and are likewise 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 `SQLHDBC` (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 `SQLHSTMT` (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 `qt_SQLDefs` (see file qt_SQLDefs.f90).

```
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

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

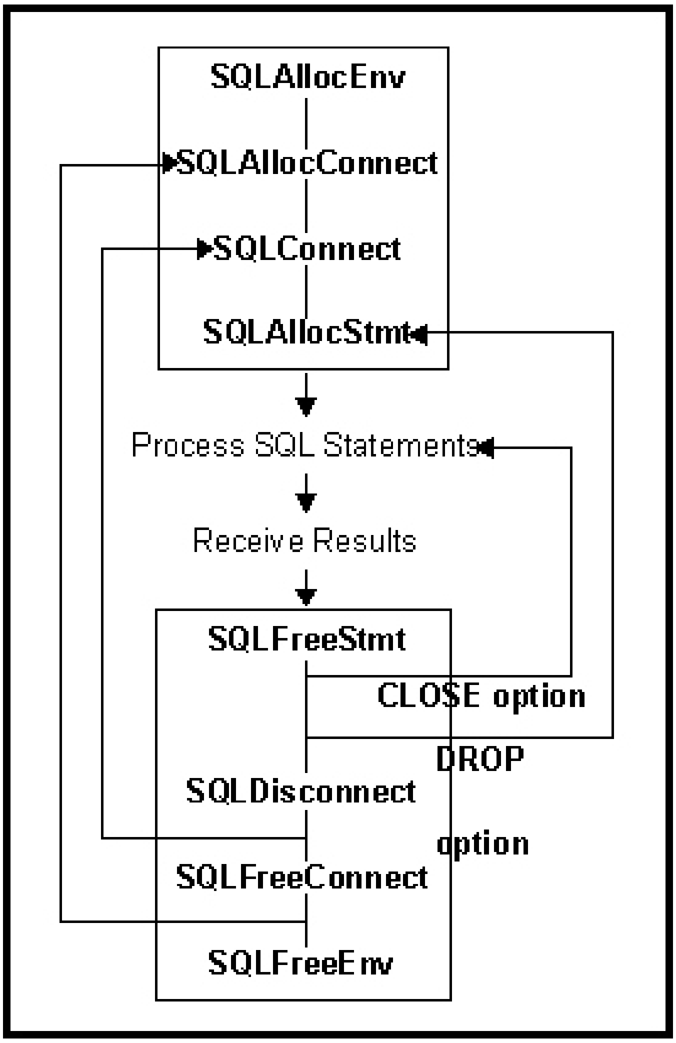
1. Establish the connection to the data source by creating an ODBC environment and connecting to the data source.

2. Execute of 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 a record 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 following diagram 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) (based on ODBC v1.x/v2.x).



Illus. 5: 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=HENV) :: env = SQL_NULL_HENV
INTEGER rtc
```

the ODBC environment is created under ODBC v1.x/v2.x as follows:

```
rtc = SQLAllocEnv( env )
```

If successful (rtc = SQL_SUCCESS), the function `SQLAllocEnv` returns the environment handle in the argument `env`. Since ODBC v3.0 a new function for the initialization is available:

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

`SQL_HANDLE_ENV` is a constant that controls which type of handle is to be created by `SQLAllocHandle`.

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 (KIND=HDBC) :: dbc = SQL_NULL_HDBC
```

or

```
INTEGER (SQLHDBC) :: dbc = SQL_NULL_HDBC
```

or

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

are equivalent. The different forms are due to changes in the last 10 years. Microsoft varied the names of the derived types a few times. This is being mentioned for the case that you see elder examples where either form has been used.

The connection is made using the function `SQLAllocConnect` (ODBC v1.x/v2.x) which will need as its first argument the formerly created environment handle.

```
rtc = SQLAllocConnect( env, dbc )
```

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

Since ODBC v3.x it is possible to use the function `SQLAllocHandle`:

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

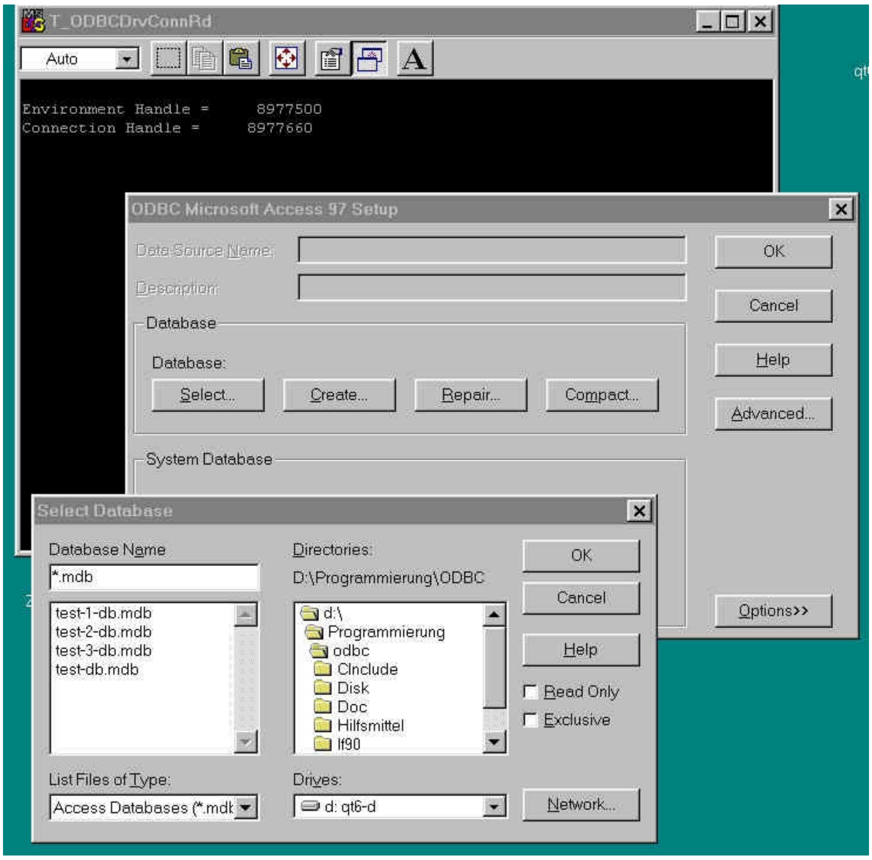
The real connection finally follows calling the function `SQLConnect`. It requires the formerly obtained connection handle and the specification of the data source name, a user name (or user id) and a password (authentication). E.g.:

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

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 either 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).



```
Environment Handle =      8977500
Connection Handle =      8977660
```

Database Name: *.mdb

test-1-db.mdb
test-2-db.mdb
test-3-db.mdb
test-db.mdb

Directories: D:\Programmierung\ODBC

d:\
Programmierung
odbc
CInclude
Disk
Doc
Hilfsmittel
If90

List Files of Type: Access Databases (*.mdb

Drives: d: qt6-d

□ Read Only

□ Exclusive

Illus.6: Selecting the data source during runtime - cf. 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 HSTMT or SQLHANDLE respectively and can be declared as follows:

```
INTEGER (KIND=HSTMT) :: stmt = SQL_NULL_HSTMT
```

The statement handle is returned by the function `SQLAllocStmt` (ODBC v1.x/v2.x). `SQLAllocStmt` required the formerly created connection handle (`dbc`).

```
rtc = SQLAllocStmt( dbc, stmt )
```

Since ODBC v3.0 one can code equivalently :

```
rtc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, stmt )
```

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.:

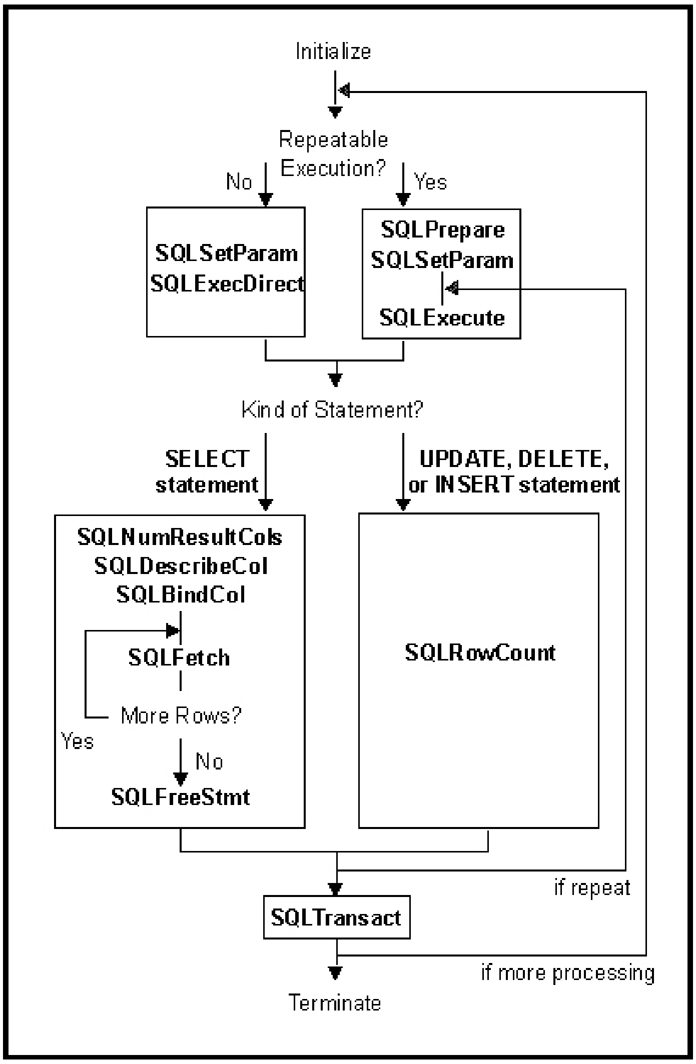
```
rtc = SQLExecDirect( stmt,   &
        "DELETE FROM Currencies WHERE CurrencyCode=
'DM'" &
        //CHAR(0), SQL_NTSL )
```

Explanation: The previously created statement handle (stmt) is used to execute the SQL statement "DELETE FROM…". The SQL command causes all entries of table "Currencies" to be deleted where the CurrencyCode equals 'DM'. The SQL statement is null terminated. As 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 recommended.

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. 7: 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 (parameters markers). E.g.:

```
INSERT INTO addressbook (name, 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) szName
INTEGER (KIND=SQLUINTEGER) :: ColSize = 30
INTEGER (KIND=SQLUINTEGER) :: iDecDigits = 0
INTEGER (KIND=SQLINTEGER) :: cbName, iBufLen = 0
.
rtc = SQLPrepare(stmt,    &
             "INSERT INTO addressbook (name, surname,
              phone) VALUES (?, ?, ?)"//CHAR(0),     &
             SQL_NTS )
rtc = SQLBindParameter( stmt, 1, SQL_PARAM_INPUT,    &
                        SQL_C_CHAR, SQL_CHAR,        &
                        ColSize, iDecDigits, szName, &
                        iBufLen, cbName )
```

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 szName 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 szName and its actual length has to be specified in cbName. Since the parameter is input, the specification of the size of the variable szName 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 SQLFreeStmt using of the options SQL_RESET_PARAMS and SQL_DROP (ODBC v1.x/v2.x). Since ODBC v3 you may perfer to use SQLFreeHandle.

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 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 function 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 (conformable to C) into which the result is to be converted (if it has to)
- a 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

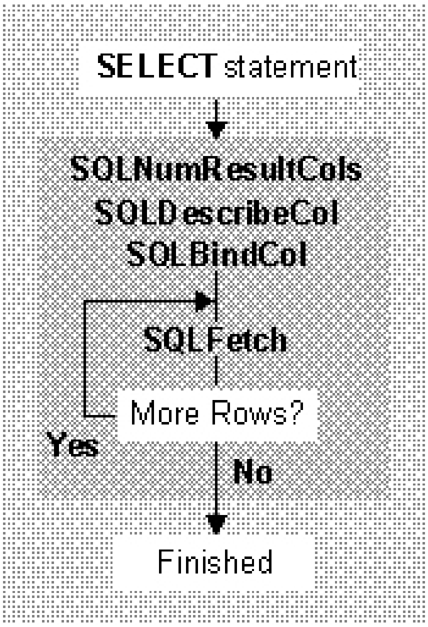
and

- 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. 8: 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 WHILE (.TRUE.)
   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 WHILE 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 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 result set which might be more appropriate if bulf 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 WHILE 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 the fetch loop outside 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 function SQLError or SQLGetDiagRec.
- SQL_NO_DATA_FOUND: The entire result set has been collected and no more data aren't 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 functions SQLFreeStmt, SQLFreeConnect and SQLFreeEnv, or SQLFreeHandle. (since ODBC v3.0) have to be called.

SQLFreeStmt 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.

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

- SQLDisconnect.

Then follows the call of the function

- SQLFreeConnect which releases the resources of the connection identified by the connection handle.

At last, the call of the function

- SQLFreeEnv releases the ODBC environment identified by the environment handle.

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.

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). In the CD-ROM root directory is an installation program which performs partly automatically. Even if you made use of it, you should read the instructions of the manual installation though, in order to get informed about what was installed, and in order to check the installation. If you received ForDBC in compressed form (ZIP), you only need to unpack into a directory of your choice.

6.1 Automated Installation

A batch file is supplied to install ForDBC:

```
install.bat
```

Start this from the DOS box:

`install [Q]  [Z]  [C]` press <Enter/Return>key

The parameters have the following meaning:

[Q]: Drive letter of your CD-ROM drive, e.g. D

[Z]: Name of the target directory without drive designator, e.g. ForDBC

[C]: Compiler, either DVF, FTN95, IVF, LF90, or LF95

The compiler names are abbreviated: DVF = Compaq Visual Fortran (formerly DigitalVisual Fortran), FTN95 = Salford FTN95, IVF = Intel Visual Fortran, LF90 = Lahey Fortran 90, LF95 = Lahey Fortran 95.

Example:

```
install  D  FORDBC  FTN95
```

The installation assumes that the current harddisk drive (e.g. C:) is the one on which you install ForDBC.

If these automated installation completes without errors, then please carry out the installation of the demo databases (see the next but one chapter).

6.2 Manual Installation

If you install ForDBC manually, or if you want to check the installation, the following describes what is to be installed.

ForDBC consists of the Fortran 9x modules
qt_Ckinds.f90
qt_Win32Kinds.f90
qt_Win32Types.f90
qt_Win32Constants.f90
qt_ODBCKinds.f90
qt_ODBCDefs.f90
qt_ODBC.f90
qt_ODBCInterfaces.f90
which are found in the root directory of ForDBC (either on the CD-ROM or in compressed ZIP file).

Each compiler specific directory contains a module named

qt_ODBC_*compiler*.f90 with *compiler* = DVF, FTN, IVF, LF90, or LF95

Copy the compiler specific module of your choice and the modules listed before to a directory of your choice on your harddisk, for example one being named ForDBC.

These modules have to be compiled then (in order as listed). You may want to use a batch file named

compile_Modules.bat

which is available in the compiler specific directory for any compiler mentioned abouve with the exception of Intel Visual Fortran.

For IVF (Intel Visual Fortran), please load into Visual Studio (the IDE) the file IVF.sln that can be found in the subdirectory IVF. Then start the "Batch Build" dialog by selecting in the menu "Build | Batch Build" and press "Build" to create all projects of the "solution".

After successful compilation the built objects (.obj) and module files (.mod) are in a directory that you should add to the module paths of your compiler to ease the process of building ODBC applications.

ForDBC provides 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_ODBCDataSources.f90	lists data sources on your computer
T_ODBCDrivers.f90	lists ODBC drivers installed on your PC
T_ODBCDrvConnRd.f90	reads MS/Access and MS/Excel files (test-db.mdb and TestODBCDrvConnRd.xls, to be selected during runtime)
T_ODBCTestAccessInfo.f90	provides information about the data source test-db.mdb
T_ODBCTestAccessRd.f90	reads data source test-db.mdb
T_ODBCTestAccessWr.f90	writes to data source test-db.mdb
T_ODBCTestExcelRd.f90	reads data source ODBCTestExcel.xls (does also display information about the table, the columns, the names etc.)

T_ODBCTestExcelWr.f90	writes to file TestODBCExcelWr.xls (the file has to be choosen at runtime)

These test programs should also be copied into the directory formerly created (for example ForDBC). Then compile them and link them with the **ODBC32.DLL** or with the import library **ODBC32.LIB**, respectively (this depends on the Fortran compiler & linker you are using) and, if necessary with an additional compiler specific interface library. For all compilers - with the exception of IVF - a batch file named

compile&link_Testprograms.bat

is provided. You may have to adapt this to your particular installation.
If you want to compile and link the test programs within a development environment, you find at the beginning of each file instructions.

The installation also contains a file named

Addendum.txt

which provide the most recent information not being convered here..

6.3 Set-up of Data Sources for Testing

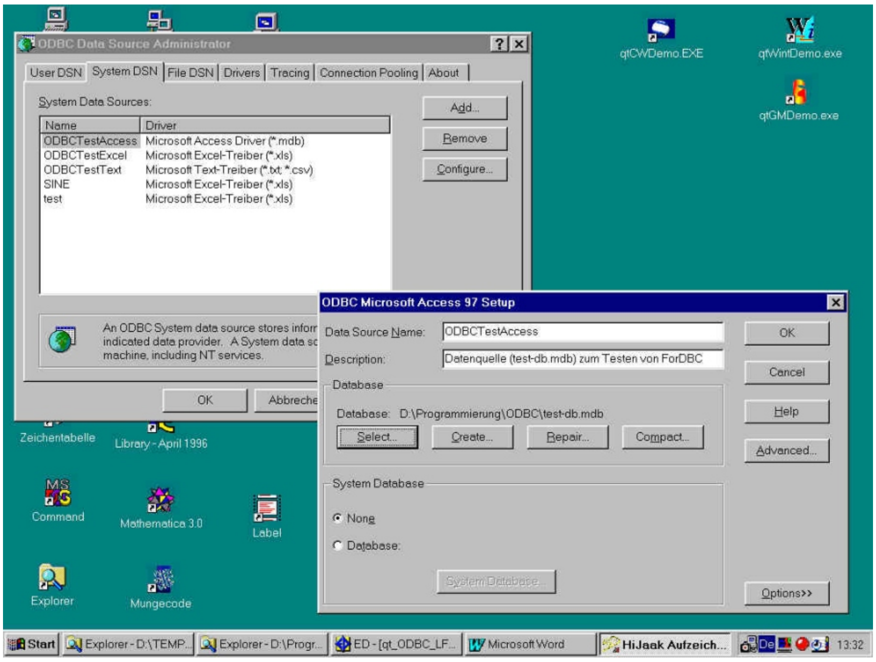
Most of the test ODBC applications (.exe) that you have created by now are only operational when the data sources they use have been set-up properly. The test programs use the following files:

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 (DSN) 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]

Then, those test programs can run successfully.



ODBC Data Source Administrator

System Data Sources:

Name	Driver
ODBCTestAccess	Microsoft Access Driver (*.mdb)
ODBCTestExcel	Microsoft Excel-Treiber (*.xls)
ODBCTestText	Microsoft Text-Treiber (*.txt; *.csv)
SINE	Microsoft Excel-Treiber (*.xls)
test	Microsoft Excel-Treiber (*.xls)

An ODBC System data source stores infor
indicated data provider. A System data sc
machine, including NT services.

ODBC Microsoft Access 97 Setup

Data Source Name: ODBCTestAccess

Description: Datenquelle (test-db.mdb) zum Testen von ForDBC

Database: D:\Programmierung\ODBC\test-db.mdb

System Database: ● None ○ Database:

Illus. 9: Set-up a data source by means of the ODBC administrator program.

For the other two Excel files that are supplied,

TestODBCDrvConnRd.xls

and

TestODBCExcelWr.xls

it is not necessary to set-up data sources.

6.4 Notes specific to Compilers

6.4.1 CompaqVisual Fortran

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

For the link step, you have to supply the import library ODBC32.LIB: 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.
Alternatively, you may want to add odbc32.lib to the files of your project.

■ 6.4.2 Intel Visual Fortran

If you want to use ForDBC in the development environment (Visual Studio), the compiler needs to know where the module files (.mod) can be found. Thus, you have to supply the module path. 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.

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" enter in the edit field "Additional Dependencies:" odbc32.lib.
Alternatively, you may want to add odbc32.lib to the files of your project.

■ 6.4.3 Lahey LF95

To generate an ODBC application, you need the modules mentioned above and the ODBC Library (ODBC32.LIB) and also an interface library which is supplied with ForDBC:

qtODBCLF95.LIB

Provided the module path is correctly set such that LF95 finds the ForDBC module files (.mod), an ODBC program is compiled an linked as shown here:

```
lf95 T_ODBCAccessRd.f90 -lib qtODBCLF95.lib ODBC32.lib
-winconsole -ml msvc
```

■ 6.4.4 Salford/Silverfrost FTN95

Unfortunately FTN95 does not provide means to program definite INTERFACES for the import of those ODBC API functions (cf. qt_ODBC_FTN.f90). Due to this there might arise problems when passing parameters of 2-byte INTEGER type variables. These problems can mostly solved by using 4-byte INTEGER variables which are initialized with 0.

When compiling, the FTN95 needs to know where the modules (.mod) are located. Thus, specify the module path using the /mod_path option.

Linking using SLINK requires to supply either the ODBC32.DLL or the import library ODBC32.LIB.

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, 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, 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, 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:

[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

File qt_ODBCInterfaces.f90

```
! ========================================
! qt_ODBCInterfaces for LF95, FTN95, DVF...
! ——————————————————————-
! (C) Jörg Kuthe, QT software, 1999-2007.
! ——————————————————————-
! Kontakt: email: jk@qtsoftware.de   http://www.qtsoftware.de

! DVF/CVF
! ———-
! compile: DF qt_ODBCInterfaces.F90 -c -win -compile_only -nologo -libs:dll /warn:nofileopt -dll

! LF95
! ——
! compile: LF95 qt_ODBCInterfaces.f90 -nwrap -c -win -mod d:.mod&obj -ml msvc
!          mit “d:.mod&obj” als dem Modulpfad

MODULE qt_ODBCInterfaces
   USE qt_ODBCKinds

   INTERFACE SQLAllocConnect
      FUNCTION SQLAllocConnect(env, dbc)
         USE qt_ODBCKinds
         INTEGER (SQLRETURN) :: SQLAllocConnect
         INTEGER (SQLHENV) :: env
         INTEGER (SQLHDBC) :: dbc
      END FUNCTION SQLAllocConnect
   END INTERFACE

   INTERFACE SQLAllocEnv
      FUNCTION SQLAllocEnv( env )
         USE qt_ODBCKinds
         INTEGER (SQLRETURN) :: SQLAllocEnv
         INTEGER (SQLHENV) :: env
      END FUNCTION SQLAllocEnv
   END INTERFACE

   INTERFACE SQLAllocHandle
      FUNCTION SQLAllocHandle( HandleType, InputHandle, OutputHandlePtr )
         USE qt_ODBCKinds
         INTEGER (SQLRETURN) :: SQLAllocHandle
         INTEGER (SQLSMALLINT) :: HandleType
         INTEGER (SQLHANDLE) :: InputHandle
         INTEGER (SQLHANDLE) :: OutputHandlePtr
      END FUNCTION SQLAllocHandle
   END INTERFACE

   INTERFACE SQLAllocStmt
      FUNCTION SQLAllocStmt( dbc, phstmt )
         USE qt_ODBCKinds
         INTEGER (SQLRETURN) :: SQLAllocStmt
         INTEGER (SQLHDBC) :: dbc
         INTEGER (SQLHSTMT) :: phstmt
      END FUNCTION SQLAllocStmt
   END INTERFACE

   INTERFACE SQLBindCol

      FUNCTION SQLBindColChar( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
      ! bind CHAR column
         USE qt_ODBCKinds
         INTEGER (SQLRETURN) :: SQLBindColChar
         INTEGER (SQLHSTMT) :: stmt
         INTEGER (SQLUSMALLINT) :: icol
         INTEGER (SQLSMALLINT) :: fCType
         CHARACTER*(*) rgbValue
         INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
      END FUNCTION SQLBindColChar

      FUNCTION SQLBindColI1( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
      ! bind INTEGER*1 column
         USE qt_ODBCKinds
         INTEGER (SQLRETURN) :: SQLBindColI1
         INTEGER (SQLHSTMT) :: stmt
         INTEGER (SQLUSMALLINT) :: icol
         INTEGER (SQLSMALLINT) :: fCType
         INTEGER*1 rgbValue
         INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
      END FUNCTION SQLBindColI1
```

```fortran

   FUNCTION SQLBindColI2( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
   ! bind INTEGER*2 column
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBindColI2
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol
      INTEGER (SQLSMALLINT) :: fCType
      INTEGER*2 rgbValue
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLBindColI2

   FUNCTION SQLBindColI4( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
   ! bind INTEGER*4 column
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBindColI4
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol
      INTEGER (SQLSMALLINT) :: fCType
      INTEGER*4 rgbValue
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLBindColI4

   FUNCTION SQLBindColR4( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
   ! bind REAL*4 column
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBindColR4
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol
      INTEGER (SQLSMALLINT) :: fCType
      REAL*4 rgbValue
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLBindColR4

   FUNCTION SQLBindColDP( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
   ! bind DOUBLE PRECISION column
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBindColDP
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol
      INTEGER (SQLSMALLINT) :: fCType
      DOUBLE PRECISION rgbValue
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLBindColDP

END INTERFACE

INTERFACE
   FUNCTION SQLBindColLP( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )   ! added 15.10.2000
   ! bind column via pointer (use LOC() function to bind variable)
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBindColLP
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol
      INTEGER (SQLSMALLINT) :: fCType
      INTEGER (LP) :: rgbValue
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLBindColLP
END INTERFACE

INTERFACE SQLBindParameter

   FUNCTION SQLBindParameterChar( stmt, ipar,    &
                                  fParamType, fCType, fSqlType, cbColDef,             &
                                  ibScale, rgbValue, cbValueMax, pcbValue )
   ! rgbValue is a CHARACTER buffer
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBindParameterChar
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: ipar
      CHARACTER (LEN=*) :: rgbValue
      INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
      INTEGER (SQLUINTEGER) :: cbColDef
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLBindParameterChar

   FUNCTION SQLBindParameterI1( stmt, ipar,    &
                                fParamType, fCType, fSqlType, cbColDef,             &
                                ibScale, rgbValue, cbValueMax, pcbValue )
   ! rgbValue is an INTEGER*1 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBindParameterI1
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: ipar
      INTEGER*1 rgbValue
      INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
      INTEGER (SQLUINTEGER) :: cbColDef
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLBindParameterI1

   FUNCTION SQLBindParameterI2( stmt, ipar,    &
                                fParamType, fCType, fSqlType, cbColDef,             &
                                ibScale, rgbValue, cbValueMax, pcbValue )
   ! rgbValue is an INTEGER*2 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBindParameterI2
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: ipar
      INTEGER*2 rgbValue
      INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
      INTEGER (SQLUINTEGER) :: cbColDef
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLBindParameterI2

   FUNCTION SQLBindParameterI4( stmt, ipar,    &
                                fParamType, fCType, fSqlType, cbColDef,            &
                                ibScale, rgbValue, cbValueMax, pcbValue )
   ! rgbValue is an INTEGER*4 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBindParameterI4
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: ipar
      INTEGER*4 rgbValue
      INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
      INTEGER (SQLUINTEGER) :: cbColDef
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLBindParameterI4

   FUNCTION SQLBindParameterR4( stmt, ipar,    &
                                fParamType, fCType, fSqlType, cbColDef,            &
                                ibScale, rgbValue, cbValueMax, pcbValue )
   ! rgbValue is a REAL*4 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBindParameterR4
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: ipar
      REAL*4 rgbValue
      INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
      INTEGER (SQLUINTEGER) :: cbColDef
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLBindParameterR4

   FUNCTION SQLBindParameterDP( stmt, ipar,    &
                                fParamType, fCType, fSqlType, cbColDef,            &
                                ibScale, rgbValue, cbValueMax, pcbValue )
   ! rgbValue is an DOUBLE PRECISION value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBindParameterDP
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: ipar
      DOUBLE PRECISION rgbValue
      INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
      INTEGER (SQLUINTEGER) :: cbColDef
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLBindParameterDP

END INTERFACE

INTERFACE   ! added 19.10.2000
   FUNCTION SQLBindParameterLP( stmt, ipar,    &
                                fParamType, fCType, fSqlType, cbColDef,            &
                                ibScale, rgbValue, cbValueMax, pcbValue )
   ! rgbValue is a pointer (use LOC()
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBindParameterLP
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: ipar
      INTEGER (LP) :: rgbValue
      INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
      INTEGER (SQLUINTEGER) :: cbColDef
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLBindParameterLP
END INTERFACE

INTERFACE SQLBrowseConnect
   FUNCTION SQLBrowseConnect( dbc, InConnectionString, cbInConnStr,    &
                              OutConnectionString, cbOutConnStr, pbOutConnStrLength )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBrowseConnect
      INTEGER (SQLHDBC) :: dbc
      CHARACTER*(*) InConnectionString, OutConnectionString
      INTEGER (SQLSMALLINT) :: cbInConnStr, cbOutConnStr, pbOutConnStrLength
   END FUNCTION SQLBrowseConnect
END INTERFACE

INTERFACE SQLBulkOperations
   FUNCTION SQLBulkOperations( Stmt, Operation )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLBulkOperations
      INTEGER (SQLHSTMT) :: Stmt
      INTEGER (SQLUSMALLINT) :: Operation
   END FUNCTION SQLBulkOperations
END INTERFACE

```

```
INTERFACE SQLCancel
   FUNCTION SQLCancel( stmt )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLCancel
      INTEGER (SQLHSTMT) :: stmt
   END FUNCTION SQLCancel
END INTERFACE

INTERFACE SQLCloseCursor
   FUNCTION SQLCloseCursor( Stmt )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLCloseCursor
      INTEGER (SQLHSTMT) :: Stmt
   END FUNCTION SQLCloseCursor
END INTERFACE

INTERFACE SQLColAttributeChar

! charAttribute is a CHARACTER buffer
   FUNCTION SQLColAttributeChar( stmt, icol, fieldId, charAttribute,      &
                                 lenCharAttribute, CharAttrLength, NumAttribute )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLColAttributeChar
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol, fieldId
      CHARACTER (LEN=*) :: charAttribute
      INTEGER (SQLSMALLINT) :: lenCharAttribute, CharAttrLength
      INTEGER (SQLINTEGER) :: NumAttribute
   END FUNCTION SQLColAttributeChar
END INTERFACE

INTERFACE SQLColAttribute
! charAttribute is a pointer
   FUNCTION SQLColAttribute( stmt, icol, fieldId, charAttribute,    &
                             lenCharAttribute, CharAttrLength, NumAttribute )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLColAttribute
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol, fieldId
      INTEGER (SQLPOINTER) :: charAttribute
      INTEGER (SQLSMALLINT) :: lenCharAttribute, CharAttrLength
      INTEGER (SQLINTEGER) :: NumAttribute
  END FUNCTION SQLColAttribute

END INTERFACE

INTERFACE SQLColAttributes
   FUNCTION SQLColAttributes( stmt, icol,  &
                              fDescType, rgbDesc, cbDescMax, pcbDesc, pfDesc )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLColAttributes
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol, fDescType
      CHARACTER (LEN=*) :: rgbDesc
      INTEGER (SQLSMALLINT) :: cbDescMax, pcbDesc
      INTEGER (SQLINTEGER) :: pfDesc
   END FUNCTION SQLColAttributes
END INTERFACE

INTERFACE SQLColumnPrivileges
   FUNCTION SQLColumnPrivileges( stmt,   &
                                 CatalogName, LenCatName,   &
                                 SchemaName, LenSchemaName, &
                                 TableName, LenTableName,  &
                                 ColumnName, LenColName )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLColumnPrivileges
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName, ColumnName
      INTEGER (SQLSMALLINT) :: LenCatName, LenSchemaName, LenTableName, LenColName
   END FUNCTION SQLColumnPrivileges
END INTERFACE

INTERFACE SQLColumns

   FUNCTION SQLColumnsChar( stmt,  &
                            szTableQualifier, cbTableQualifier,       &
                            szTableOwner, cbTableOwner,               &
                            szTableName, cbTableName,                 &
                            szColumnName, cbColumnName )      ! changed 14.10.2000: szColumnName, cbColum
                                 nName )    ! changed 14.10.2000: SQLColumns -> SQLColumnsChar
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLColumnsChar
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER (LEN=*) :: szTableQualifier, szTableOwner,  &
                           szTableName, szColumnName
      INTEGER (SQLSMALLINT) :: cbTableQualifier, cbTableOwner, cbTableName, cbColumnName
      !DEC$ ATTRIBUTES REFERENCE :: szTableName, szColumnName
   END FUNCTION SQLColumnsChar

   ! 14.10.2000: added SQLColumnsLP (all arguments being transferred as values, use LOC() to pass a refer
        ence)
   FUNCTION SQLColumnsLP( stmt,  &
                          szTableQualifier, cbTableQualifier,       &
                          szTableOwner, cbTableOwner,               &
                          szTableName, cbTableName,                 &
                          szColumnName, cbColumnName )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLColumnsLP
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (LP) :: szTableQualifier, szTableOwner,  &
                      szTableName, szColumnName
      INTEGER (SQLSMALLINT) :: cbTableQualifier, cbTableOwner, cbTableName, cbColumnName
   END FUNCTION SQLColumnsLP

END INTERFACE

INTERFACE SQLConnect
   FUNCTION SQLConnect( dbc, szDSN, cbDSN, szUID, cbUID, szAuthStr, cbAuthStr )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) ::SQLConnect
      INTEGER (SQLHDBC) :: dbc
      CHARACTER*(*) szDSN, szUID, szAuthStr
      INTEGER (SQLSMALLINT) :: cbDSN, cbUID, cbAuthStr
   END FUNCTION SQLConnect
END INTERFACE

INTERFACE SQLCopyDesc
   FUNCTION SQLCopyDesc( SourceDescHandle, TargetDescHandle )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLCopyDesc
      INTEGER (SQLHDESC) :: SourceDescHandle, TargetDescHandle
   END FUNCTION SQLCopyDesc
END INTERFACE

INTERFACE SQLDataSources
   FUNCTION SQLDataSources( env, fDirection,   &
                            szDSN, cbDSNMax, pcbDSN,   &
                            szDescription, cbDescriptionMax, pcbDescription )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLDataSources
      INTEGER (SQLHENV) :: env
      INTEGER (SQLUSMALLINT) :: fDirection
      CHARACTER (LEN=*) :: szDSN, szDescription
      INTEGER (SQLSMALLINT) :: cbDSNMax, pcbDSN, cbDescriptionMax, pcbDescription
   END FUNCTION SQLDataSources
END INTERFACE

INTERFACE SQLDescribeCol
   FUNCTION SQLDescribeCol( stmt, icol,    &
                            szColName, cbColNameMax, pcbColName,   &
                            pfSqlType, pcbColDef, pibScale, pfNullable )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLDescribeCol
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol
      CHARACTER (LEN=*) :: szColName
      INTEGER (SQLSMALLINT) :: cbColNameMax, pcbColName, pfSqlType, pibScale, pfNullable
      INTEGER (SQLUINTEGER) :: pcbColDef
   END FUNCTION SQLDescribeCol
END INTERFACE

INTERFACE SQLDescribeParam
   FUNCTION SQLDescribeParam( stmt, ipar, pfSqlType,   &
                              pcbColDef, pibScale, pfNullable )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLDescribeParam
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: ipar
      INTEGER (SQLSMALLINT) :: pfSqlType, pibScale, pfNullable
      INTEGER (SQLUINTEGER) :: pcbColDef
   END FUNCTION SQLDescribeParam
END INTERFACE

INTERFACE SQLDisconnect
   FUNCTION SQLDisconnect( dbc )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLDisconnect
      INTEGER (SQLHDBC) :: dbc
   END FUNCTION SQLDisconnect
END INTERFACE

INTERFACE   ! SQLDriverConnect; DVF5 -> ERROR (could not find generic interface specific function...!)
   FUNCTION SQLDriverConnect( dbc, wnd,  &
                              szConnStrIn, cbConnStrIn,  &
                              szConnStrOut, cbConnStrOutMax, pcbConnStrOut, &
                              fDriverCompletion)
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLDriverConnect
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLHWND) :: wnd
      CHARACTER (LEN=*) :: szConnStrIn, szConnStrOut
      INTEGER (SQLSMALLINT) :: cbConnStrIn, cbConnStrOutMax, pcbConnStrOut
      INTEGER (SQLUSMALLINT) :: fDriverCompletion
   END FUNCTION SQLDriverConnect
END INTERFACE

INTERFACE SQLDrivers
   FUNCTION SQLDrivers( env, fDirection,   &
                        szDrvDesc, cbDrvDescMax, pcbDrvDesc,   &
                        szDrvAttr, cbDrvAttrMax, pcbDrvAttr )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLDrivers
      INTEGER (SQLHENV) :: env
      INTEGER (SQLUSMALLINT) :: fDirection
      CHARACTER (LEN=*) :: szDrvDesc, szDrvAttr
      INTEGER (SQLSMALLINT) :: cbDrvDescMax, pcbDrvDesc, cbDrvAttrMax, pcbDrvAttr
   END FUNCTION SQLDrivers
END INTERFACE

INTERFACE SQLEndTran
  FUNCTION SQLEndTran( HandleType, hndl, CompletionType )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLEndTran
      INTEGER (SQLSMALLINT) :: HandleType
      INTEGER (SQLHANDLE) :: hndl
      INTEGER (SQLSMALLINT) :: CompletionType
   END FUNCTION SQLEndTran
END INTERFACE

INTERFACE SQLError
   FUNCTION SQLError( env, dbc, stmt, szSqlState, pfNativeError,  &
                      szErrorMsg, cbErrorMsgMax, pcbErrorMsg )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLError
      INTEGER (SQLHENV) :: env
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER*(*) szSqlState, szErrorMsg
      INTEGER (SQLINTEGER) :: pfNativeError
      INTEGER (SQLSMALLINT) :: cbErrorMsgMax, pcbErrorMsg
       !DEC$ ATTRIBUTES STDCALL, ALIAS : ‘_SQLError@32’ :: SQLError
  END FUNCTION SQLError
END INTERFACE

INTERFACE SQLExecDirect
   FUNCTION SQLExecDirect( stmt, szSqlStr, cbSqlStr )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLExecDirect
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER*(*) szSqlStr
      INTEGER (SQLINTEGER) :: cbSqlStr
   END FUNCTION SQLExecDirect
END INTERFACE

INTERFACE SQLExecute
   FUNCTION SQLExecute( stmt )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLExecute
      INTEGER (SQLHSTMT) :: stmt
   END FUNCTION SQLExecute
END INTERFACE

INTERFACE SQLExtendedFetch
   FUNCTION SQLExtendedFetch( stmt, fFetchType, irow, pcrow, rgfRowStatus )
      USE qt_ODBCKinds
      INTEGER (RETCODE) :: SQLExtendedFetch
      INTEGER (HSTMT) :: stmt
      INTEGER (UWORD) :: fFetchType, rgfRowStatus
      INTEGER (SDWORD) :: irow
      INTEGER (UDWORD) :: pcrow
   END FUNCTION SQLExtendedFetch
END INTERFACE

INTERFACE SQLFetch
   FUNCTION SQLFetch( stmt )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLFetch
      INTEGER (SQLHSTMT) :: stmt
   END FUNCTION SQLFetch
END INTERFACE

INTERFACE SQLFetchScroll
   FUNCTION SQLFetchScroll( stmt, FetchOrientation, FetchOffset )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLFetchScroll
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLSMALLINT) :: FetchOrientation
      INTEGER (SQLINTEGER) :: FetchOffset
   END FUNCTION SQLFetchScroll
END INTERFACE

INTERFACE SQLForeignKeys
```

```
   FUNCTION SQLForeignKeys( stmt, PKCatalogName, PKCatNameLength,    &
                                  PKSchemaName, PKSchemaNameLength,  &
                                  PKTableName, PKTableNameLength,    &
                                  FKCatalogName, FKCatalogNameLength,&
                                  FKSchemaName, FKSchemaNameLength,  &
                                  FKTableName, FKTableNameLength)
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLForeignKeys
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER (LEN=*) :: PKCatalogName, PKSchemaName, PKTableName,  &
                           FKCatalogName, FKSchemaName, FKTableName
      !DEC$ ATTRIBUTES REFERENCE :: FKCatalogName, FKSchemaName, FKTableName
      INTEGER (SQLSMALLINT) :: PKCatNameLength, PKSchemaNameLength, PKTableNameLength, &
                               FKCatalogNameLength, FKSchemaNameLength, FKTableNameLength
   END FUNCTION SQLForeignKeys
END INTERFACE

INTERFACE SQLFreeConnect
   FUNCTION SQLFreeConnect( dbc )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLFreeConnect
      INTEGER (SQLHDBC) :: dbc
   END FUNCTION SQLFreeConnect
END INTERFACE

INTERFACE SQLFreeEnv
   FUNCTION SQLFreeEnv( env )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLFreeEnv
      INTEGER (SQLHENV) :: env
   END FUNCTION SQLFreeEnv
END INTERFACE

INTERFACE SQLFreeHandle
   FUNCTION SQLFreeHandle( HndType, Hnd )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLFreeHandle
      INTEGER (SQLSMALLINT) :: HndType
      INTEGER (SQLHANDLE) :: Hnd
   END FUNCTION SQLFreeHandle
END INTERFACE

INTERFACE SQLFreeStmt
   FUNCTION SQLFreeStmt( stmt, fOption )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLFreeStmt
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: fOption
   END FUNCTION SQLFreeStmt
END INTERFACE

INTERFACE SQLGetConnectAttrChar
! ValuePtr is a CHARACTER buffer
   FUNCTION SQLGetConnectAttrChar( dbc, Attrib, ValuePtr, LenValuePtr, ValuePtrLength)
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetConnectAttrChar
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLINTEGER) :: Attrib, LenValuePtr, ValuePtrLength
      CHARACTER (LEN=*) :: ValuePtr
   END FUNCTION SQLGetConnectAttrChar
END INTERFACE

INTERFACE SQLGetConnectAttr
! ValuePtr is a pointer to a buffer
   FUNCTION SQLGetConnectAttr( dbc, Attrib, ValuePtr, LenValuePtr, ValuePtrLength)
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetConnectAttr
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLINTEGER) :: Attrib, LenValuePtr, ValuePtrLength
      INTEGER (SQLPOINTER) :: ValuePtr
   END FUNCTION SQLGetConnectAttr
END INTERFACE

INTERFACE SQLGetConnectOption

   FUNCTION SQLGetConnectOptionChar( dbc, fOption, pvParam )
   ! pvParam is a CHARACTER buffer
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetConnectOptionChar
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLUSMALLINT) :: fOption
      CHARACTER (LEN=*) :: pvParam
   END FUNCTION SQLGetConnectOptionChar

   FUNCTION SQLGetConnectOptionI4( dbc, fOption, pvParam )
   ! pvParam is an INTEGER*4 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetConnectOptionI4
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLUSMALLINT) :: fOption
      INTEGER*4 pvParam
   END FUNCTION SQLGetConnectOptionI4

END INTERFACE

INTERFACE SQLGetCursorName
   FUNCTION SQLGetCursorName( stmt, szCursor, cbCursorMax, pcbCursor )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetCursorName
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER (LEN=*) :: szCursor
      INTEGER (SQLSMALLINT) :: cbCursorMax, pcbCursor
   END FUNCTION SQLGetCursorName
END INTERFACE

INTERFACE SQLGetData

   FUNCTION SQLGetDataChar( stmt, icol, fCType,   &
                            rgbValue, cbValueMax, pcbValue )
   ! rgbValue is a CHARACTER buffer
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetDataChar
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol
      CHARACTER (LEN=*) :: rgbValue
      INTEGER (SQLSMALLINT) :: fCType
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLGetDataChar

   FUNCTION SQLGetDataI2( stmt, icol, fCType,   &
                          rgbValue, cbValueMax, pcbValue )
   ! rgbValue is an INTEGER*2 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetDataI2
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol
      INTEGER*2 rgbValue
      INTEGER (SQLSMALLINT) :: fCType
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLGetDataI2

   FUNCTION SQLGetDataI4( stmt, icol, fCType,   &
                          rgbValue, cbValueMax, pcbValue )
   ! rgbValue is an INTEGER*4 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetDataI4
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol
      INTEGER*4 rgbValue
      INTEGER (SQLSMALLINT) :: fCType
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLGetDataI4

   FUNCTION SQLGetDataR4( stmt, icol, fCType,   &
                          rgbValue, cbValueMax, pcbValue )
   ! rgbValue is a REAL*4 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetDataR4
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol
      REAL*4 rgbValue
      INTEGER (SQLSMALLINT) :: fCType
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLGetDataR4

   FUNCTION SQLGetDataDP( stmt, icol, fCType,   &
                          rgbValue, cbValueMax, pcbValue )
   ! rgbValue is a DOUBLE PRECISION value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetDataDP
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: icol
      DOUBLE PRECISION rgbValue
      INTEGER (SQLSMALLINT) :: fCType
      INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
   END FUNCTION SQLGetDataDP

END INTERFACE

INTERFACE SQLGetDescFieldChar

! ValuePtr is a CHARACTER buffer
   FUNCTION SQLGetDescFieldChar( DescriptorHandle, RecNumber, FieldIdentifier,  &
                                 ValuePtr, LenValuePtr, ValuePtrLen )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetDescFieldChar
      INTEGER (SQLHDESC) :: DescriptorHandle
      INTEGER (SQLSMALLINT) :: RecNumber, FieldIdentifier
      CHARACTER (LEN=*) :: ValuePtr
      INTEGER (SQLINTEGER) :: LenValuePtr, ValuePtrLen
   END FUNCTION SQLGetDescFieldChar
END INTERFACE
```

```
INTERFACE SQLGetDescField
! ValuePtr is a pointer
   FUNCTION SQLGetDescField( DescriptorHandle, RecNumber, FieldIdentifier,  &
                             ValuePtr, LenValuePtr, ValuePtrLen )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetDescField
      INTEGER (SQLHDESC) :: DescriptorHandle
      INTEGER (SQLSMALLINT) :: RecNumber, FieldIdentifier
      INTEGER (SQLPOINTER) :: ValuePtr
      INTEGER (SQLINTEGER) :: LenValuePtr, ValuePtrLen
   END FUNCTION SQLGetDescField
END INTERFACE

INTERFACE SQLGetDescRec
   FUNCTION SQLGetDescRec( DescriptorHandle, RecNumber, DescName,  &
                           LenDescName, DescNameLength, TypePtr, SubTypePtr,   &
                           LengthPtr, PrecisionPtr, ScalePtr, NullablePtr )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetDescRec
      INTEGER (SQLHDESC) :: DescriptorHandle
      INTEGER (SQLSMALLINT) :: RecNumber, LenDescName, DescNameLength,  &
                               TypePtr, SubTypePtr, PrecisionPtr, ScalePtr, NullablePtr
      INTEGER (SQLINTEGER) :: LengthPtr
      CHARACTER (LEN=*) :: DescName
      !DEC$ ATTRIBUTES REFERENCE :: LengthPtr, PrecisionPtr, ScalePtr, NullablePtr
   END FUNCTION SQLGetDescRec
END INTERFACE

INTERFACE SQLGetDiagField
   FUNCTION SQLGetDiagField( HandleType, Hndl, RecNumber, DiagIdentifier,  &
                             DiagInfoPtr, LenDiagInfo, DiagInfoLen )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetDiagField
      INTEGER (SQLSMALLINT) :: HandleType, RecNumber, DiagIdentifier,   &
                               LenDiagInfo, DiagInfoLen
      INTEGER (SQLHANDLE) :: Hndl
      INTEGER (SQLPOINTER) ::  DiagInfoPtr
   END FUNCTION SQLGetDiagField
END INTERFACE

INTERFACE SQLGetDiagRec
   FUNCTION SQLGetDiagRec( HandleType, Hndl, RecNumber, Sqlstate,  &
                           NativeError, MessageText, LenMsgText, MsgTextLen )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetDiagRec
      INTEGER (SQLSMALLINT) :: HandleType, RecNumber, LenMsgText, MsgTextLen
      INTEGER (SQLHANDLE) :: Hndl
      CHARACTER (LEN=*) :: Sqlstate, MessageText
      INTEGER (SQLINTEGER) :: NativeError
   END FUNCTION SQLGetDiagRec
END INTERFACE

INTERFACE SQLGetEnvAttr

! Value is a CHARACTER buffer
   FUNCTION SQLGetEnvAttrChar( env, Attribute, Value, LenValue, ValueLength )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetEnvAttrChar
      INTEGER (SQLHENV) :: env
      INTEGER (SQLINTEGER) :: Attribute, LenValue, ValueLength
      CHARACTER (LEN=*) :: Value
   END FUNCTION SQLGetEnvAttrChar
! Value is an INTEGER
   FUNCTION SQLGetEnvAttrI4( env, Attribute, Value, LenValue, ValueLength )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetEnvAttrI4
      INTEGER (SQLHENV) :: env
      INTEGER (SQLINTEGER) :: Attribute, LenValue, ValueLength
      INTEGER (SQLINTEGER) :: Value
   END FUNCTION SQLGetEnvAttrI4

END INTERFACE

INTERFACE SQLGetFunctions
   FUNCTION SQLGetFunctions( dbc, fFunction, pfExists )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetFunctions
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLUSMALLINT) :: fFunction, pfExists
   END FUNCTION SQLGetFunctions
END INTERFACE

INTERFACE SQLGetInfo

   FUNCTION SQLGetInfoChar( dbc, fInfoType, rgbInfoValue,     &
                            cbInfoValueMax, pcbInfoValue )
   ! rgbInfoValue is a CHARACTER buffer
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetInfoChar
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLUSMALLINT) :: fInfoType
      CHARACTER (LEN=*) :: rgbInfoValue
      INTEGER (SQLSMALLINT) :: cbInfoValueMax, pcbInfoValue
   END FUNCTION SQLGetInfoChar

   FUNCTION SQLGetInfoI2( dbc, fInfoType, rgbInfoValue,    &
                          cbInfoValueMax, pcbInfoValue )
   ! rgbInfoValue is of type INTEGER*2
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetInfoI2
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLUSMALLINT) :: fInfoType
      INTEGER*2 rgbInfoValue
      INTEGER (SQLSMALLINT) :: cbInfoValueMax, pcbInfoValue
   END FUNCTION SQLGetInfoI2

   FUNCTION SQLGetInfoI4( dbc, fInfoType, rgbInfoValue,    &
                          cbInfoValueMax, pcbInfoValue )
   ! rgbInfoValue is of type INTEGER*4
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetInfoI4
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLUSMALLINT) :: fInfoType
      INTEGER*4 rgbInfoValue
      INTEGER (SQLSMALLINT) :: cbInfoValueMax, pcbInfoValue
   END FUNCTION SQLGetInfoI4

END INTERFACE

INTERFACE SQLGetStmtAttrChar

! Value is a CHARACTER buffer
   FUNCTION SQLGetStmtAttrChar( stmt, Attribute, Value, LenValue, ValueLength )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetStmtAttrChar
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLINTEGER) :: Attribute, LenValue, ValueLength
      CHARACTER (LEN=*) :: Value
   END FUNCTION SQLGetStmtAttrChar
END INTERFACE

INTERFACE SQLGetStmtAttr
! Value is a pointer to a buffer
   FUNCTION SQLGetStmtAttr( stmt, Attribute, ValuePtr, LenValue, ValueLength )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetStmtAttr
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLINTEGER) :: Attribute, LenValue, ValueLength
      INTEGER (SQLPOINTER) :: ValuePtr
   END FUNCTION SQLGetStmtAttr

END INTERFACE

INTERFACE SQLGetStmtOption

   FUNCTION SQLGetStmtOptionChar( stmt, fOption, pvParam )
   ! pvParam is a CHARACTER buffer
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetStmtOptionChar
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: fOption
      CHARACTER (LEN=*) :: pvParam
   END FUNCTION SQLGetStmtOptionChar

   FUNCTION SQLGetStmtOptionI4( stmt, fOption, pvParam )
   ! pvParam is an INTEGER*4 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetStmtOptionI4
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: fOption
      INTEGER*4 pvParam
   END FUNCTION SQLGetStmtOptionI4

END INTERFACE

INTERFACE SQLGetTypeInfo
   FUNCTION SQLGetTypeInfo( stmt, fSqlType )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLGetTypeInfo
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLSMALLINT) :: fSqlType
   END FUNCTION SQLGetTypeInfo
END INTERFACE

INTERFACE SQLMoreResults
   FUNCTION SQLMoreResults( stmt )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLMoreResults
      INTEGER (SQLHSTMT) :: stmt
   END FUNCTION SQLMoreResults
END INTERFACE

```

```
INTERFACE SQLNativeSql
   FUNCTION SQLNativeSql( dbc, szSqlStrIn, cbSqlStrIn,  &
                               szSqlStr, cbSqlStrMax, pcbSqlStr )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLNativeSql
      INTEGER (SQLHDBC) :: dbc
      CHARACTER (LEN=*) :: szSqlStrIn, szSqlStr
      INTEGER (SQLINTEGER) :: cbSqlStrIn, cbSqlStrMax, pcbSqlStr
   END FUNCTION SQLNativeSql
END INTERFACE

INTERFACE SQLNumParams
   FUNCTION SQLNumParams( stmt, pcpar )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLNumParams
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLSMALLINT) :: pcpar
   END FUNCTION SQLNumParams
END INTERFACE

INTERFACE SQLNumResultCols
   FUNCTION SQLNumResultCols( stmt, pccol )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLNumResultCols
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLSMALLINT) :: pccol
   END FUNCTION SQLNumResultCols
END INTERFACE

INTERFACE SQLParamData

   FUNCTION SQLParamDataChar( stmt, prgbValue )
   ! prgbValue is a CHARACTER buffer
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLParamDataChar
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER (LEN=*) :: prgbValue
   END FUNCTION SQLParamDataChar

   FUNCTION SQLParamDataI2( stmt, prgbValue )
   ! prgbValue is an INTEGER*2 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLParamDataI2
      INTEGER (SQLHSTMT) :: stmt
      INTEGER*2 prgbValue
   END FUNCTION SQLParamDataI2

   FUNCTION SQLParamDataI4( stmt, prgbValue )
   ! prgbValue is an INTEGER*4 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLParamDataI4
      INTEGER (SQLHSTMT) :: stmt
      INTEGER*4 prgbValue
   END FUNCTION SQLParamDataI4

   FUNCTION SQLParamDataR4( stmt, prgbValue )
   ! prgbValue is an REAL*4 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLParamDataR4
      INTEGER (SQLHSTMT) :: stmt
      REAL*4 prgbValue
   END FUNCTION SQLParamDataR4

   FUNCTION SQLParamDataDP( stmt, prgbValue )
   ! prgbValue is an DOUBLE PRECISION value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLParamDataDP
      INTEGER (SQLHSTMT) :: stmt
      DOUBLE PRECISION prgbValue
   END FUNCTION SQLParamDataDP

END INTERFACE

INTERFACE SQLParamOptions
   FUNCTION SQLParamOptions( stmt, crow, pirow )
      USE qt_ODBCKinds
      INTEGER (RETCODE) :: SQLParamOptions
      INTEGER (HSTMT) :: stmt
      INTEGER (UDWORD) :: crow, pirow
   END FUNCTION SQLParamOptions
END INTERFACE

INTERFACE SQLPrepare
   FUNCTION SQLPrepare( stmt, szSqlStr, cbSqlStr )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLPrepare
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER (LEN=*) :: szSqlStr
      INTEGER (SQLINTEGER) :: cbSqlStr
   END FUNCTION SQLPrepare
END INTERFACE
```

```

INTERFACE SQLPrimaryKeys
   FUNCTION SQLPrimaryKeys( stmt, CatalogName, CatNameLength,   &
                                  SchemaName, SchemaNameLength, &
                                  TableName, TableNameLength )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLPrimaryKeys
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName
      INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength, TableNameLength
   END FUNCTION SQLPrimaryKeys
END INTERFACE

INTERFACE SQLProcedureColumns
   FUNCTION SQLProcedureColumns( stmt, CatalogName, CatNameLength,   &
                                       SchemaName, SchemaNameLength, &
                                       ProcName, ProcNameLength,     &
                                       ColumnName, ColNameLength )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLProcedureColumns
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER (LEN=*) :: CatalogName, SchemaName, ProcName, ColumnName
      INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength,   &
                               ProcNameLength, ColNameLength
   END FUNCTION SQLProcedureColumns
END INTERFACE

INTERFACE SQLProcedures
   FUNCTION SQLProcedures( stmt, CatalogName, CatNameLength,   &
                                 SchemaName, SchemaNameLength, &
                                 ProcName, ProcNameLength )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLProcedures
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER (LEN=*) :: CatalogName, SchemaName, ProcName
      INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength, ProcNameLength
   END FUNCTION SQLProcedures
END INTERFACE

INTERFACE SQLPutData

   FUNCTION SQLPutDataChar( stmt, rgbValue, cbValue )
   ! rgbValue is a CHARACTER buffer
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLPutDataChar
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER (LEN=*) :: rgbValue
      INTEGER (SQLINTEGER) :: cbValue
   END FUNCTION SQLPutDataChar

   FUNCTION SQLPutDataI2( stmt, rgbValue, cbValue )
   ! rgbValue is an INTEGER*2 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLPutDataI2
      INTEGER (SQLHSTMT) :: stmt
      INTEGER*2 rgbValue
      INTEGER (SQLINTEGER) :: cbValue
   END FUNCTION SQLPutDataI2

   FUNCTION SQLPutDataI4( stmt, rgbValue, cbValue )
   ! rgbValue is an INTEGER*4 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLPutDataI4
      INTEGER (SQLHSTMT) :: stmt
      INTEGER*4 rgbValue
      INTEGER (SQLINTEGER) :: cbValue
   END FUNCTION SQLPutDataI4

   FUNCTION SQLPutDataR4( stmt, rgbValue, cbValue )
   ! rgbValue is an REAL*4 value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLPutDataR4
      INTEGER (SQLHSTMT) :: stmt
      REAL*4 rgbValue
      INTEGER (SQLINTEGER) :: cbValue
   END FUNCTION SQLPutDataR4

   FUNCTION SQLPutDataDP( stmt, rgbValue, cbValue )
   ! rgbValue is an DOUBLE PRECISION value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLPutDataDP
      INTEGER (SQLHSTMT) :: stmt
      DOUBLE PRECISION rgbValue
      INTEGER (SQLINTEGER) :: cbValue
   END FUNCTION SQLPutDataDP

END INTERFACE

INTERFACE SQLRowCount
   FUNCTION SQLRowCount( stmt, pcrow )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLRowCount
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLINTEGER) :: pcrow
   END FUNCTION SQLRowCount
END INTERFACE

INTERFACE SQLSetConnectAttr
   FUNCTION SQLSetConnectAttrLP( dbc, Attribute, ValuePtr, StringLength )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetConnectAttrLP
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLINTEGER) :: Attribute
      INTEGER (SQLPOINTER) :: ValuePtr
      INTEGER (SQLINTEGER) :: StringLength
   END FUNCTION SQLSetConnectAttrLP
END INTERFACE

INTERFACE SQLSetConnectAttrChar
   FUNCTION SQLSetConnectAttrChar( dbc, Attribute, ValuePtr, StringLength )
   ! ValuePtr is a zero terminated string
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetConnectAttrChar
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLINTEGER) :: Attribute
      CHARACTER (LEN=*) :: ValuePtr
      INTEGER (SQLINTEGER) :: StringLength
   END FUNCTION SQLSetConnectAttrChar
END INTERFACE

INTERFACE SQLSetConnectOption
   FUNCTION SQLSetConnectOption( dbc, fOption, vParam )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetConnectOption
      INTEGER (SQLHDBC) :: dbc
      INTEGER (SQLUSMALLINT) :: fOption
      INTEGER (SQLUINTEGER) :: vParam
   END FUNCTION SQLSetConnectOption
END INTERFACE

INTERFACE SQLSetCursorName
   FUNCTION SQLSetCursorName( stmt, szCursor, cbCursor )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetCursorName
      INTEGER (SQLHSTMT) :: stmt
      CHARACTER (LEN=*) :: szCursor
      INTEGER (SQLSMALLINT) :: cbCursor
   END FUNCTION SQLSetCursorName
END INTERFACE

INTERFACE SQLSetDescFieldChar
! ValuePtr is a CHARACTER buffer
   FUNCTION SQLSetDescFieldChar( DescriptorHandle, RecNumber, FieldIdentifier,  &
                                 ValuePtr, LenValuePtr )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetDescFieldChar
      INTEGER (SQLHDESC) :: DescriptorHandle
      INTEGER (SQLSMALLINT) :: RecNumber, FieldIdentifier
      CHARACTER (LEN=*) :: ValuePtr
      INTEGER (SQLINTEGER) :: LenValuePtr
   END FUNCTION SQLSetDescFieldChar
END INTERFACE

INTERFACE SQLSetDescField
! ValuePtr is a pointer
   FUNCTION SQLSetDescField( DescriptorHandle, RecNumber, FieldIdentifier,  &
                             ValuePtr, LenValuePtr )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetDescField
      INTEGER (SQLHDESC) :: DescriptorHandle
      INTEGER (SQLSMALLINT) :: RecNumber, FieldIdentifier
      INTEGER (SQLPOINTER) :: ValuePtr
      INTEGER (SQLINTEGER) :: LenValuePtr
   END FUNCTION SQLSetDescField
END INTERFACE

INTERFACE SQLSetDescRec
   FUNCTION SQLSetDescRec( DescriptorHandle, RecNumber, ValType, SubType,   &
                           fldLength, PrecVal, ScaleVal, DataPtr,       &
                           StringLength, Indicator )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetDescRec
      INTEGER (SQLHDESC) :: DescriptorHandle
      INTEGER (SQLSMALLINT) :: RecNumber, ValType, SubType, PrecVal, ScaleVal, NullablePtr
      INTEGER (SQLINTEGER) :: fldLength, StringLength, Indicator
      INTEGER (SQLPOINTER) :: DataPtr
   END FUNCTION SQLSetDescRec
END INTERFACE

INTERFACE SQLSetEnvAttr
   FUNCTION SQLSetEnvAttrI4( env, Attribute, ValuePtr, StringLength )      ! corr. 12.10.2000:
SQLSetEnvAttr -> SQLSetEnvAttrI4
        ttr -> SQLSetEnvAttrI4
   ! ValuePtr is a 32-bit unsigned integer value
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetEnvAttrI4
      INTEGER (SQLHENV) :: env
      INTEGER (SQLINTEGER) :: Attribute
      INTEGER (SQLPOINTER) :: ValuePtr
      INTEGER (SQLINTEGER) :: StringLength
   END FUNCTION SQLSetEnvAttrI4
END INTERFACE

INTERFACE SQLSetEnvAttrChar
   FUNCTION SQLSetEnvAttrChar( env, Attribute, ValuePtr, StringLength )
   ! ValuePtr is a zero terminated string
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetEnvAttrChar
      INTEGER (SQLHENV) :: env
      INTEGER (SQLINTEGER) :: Attribute
      CHARACTER (LEN=*) :: ValuePtr
      INTEGER (SQLINTEGER) :: StringLength
   END FUNCTION SQLSetEnvAttrChar
END INTERFACE

INTERFACE
   FUNCTION SQLSetPos( stmt, irow, fOption, fLock )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetPos
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: irow, fOption, fLock
   END FUNCTION SQLSetPos
END INTERFACE

INTERFACE SQLSetScrollOptions
   FUNCTION SQLSetScrollOptions( stmt, fConcurrency, crowKeyset, crowRowset )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetScrollOptions
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: fConcurrency, crowRowset
      INTEGER (SQLINTEGER) :: crowKeyset
   END FUNCTION SQLSetScrollOptions
END INTERFACE

INTERFACE SQLSetStmtAttrChar

! Value is a CHARACTER buffer
   FUNCTION SQLSetStmtAttrChar( stmt, Attribute, Value, LenValue )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetStmtAttrChar
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLINTEGER) :: Attribute, LenValue
      CHARACTER (LEN=*) :: Value
   END FUNCTION SQLSetStmtAttrChar
! Value is an INTEGER*4
END INTERFACE

INTERFACE SQLSetStmtAttrI4
   FUNCTION SQLSetStmtAttrI4( stmt, Attribute, Value, LenValue )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetStmtAttrI4
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLINTEGER) :: Attribute, LenValue
      INTEGER*4 Value
   END FUNCTION SQLSetStmtAttrI4
! Value is a pointer to a buffer
END INTERFACE

INTERFACE SQLSetStmtAttr
   FUNCTION SQLSetStmtAttr( stmt, Attribute, ValuePtr, LenValue )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetStmtAttr
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLINTEGER) :: Attribute, LenValue
      INTEGER (SQLPOINTER) :: ValuePtr
   END FUNCTION SQLSetStmtAttr
END INTERFACE

INTERFACE SQLSetStmtOption
   FUNCTION SQLSetStmtOption( stmt, fOption, vParam )
      USE qt_ODBCKinds
      INTEGER (SQLRETURN) :: SQLSetStmtOption
      INTEGER (SQLHSTMT) :: stmt
      INTEGER (SQLUSMALLINT) :: fOption
      INTEGER (SQLUINTEGER) :: vParam
   END FUNCTION SQLSetStmtOption
END INTERFACE

INTERFACE SQLSpecialColumns
   FUNCTION SQLSpecialColumns( stmt, IdentifierType,  &
                                     CatalogName, CatNameLength,    &
                                     SchemaName, SchemaNameLength,  &
                                     TableName, TableNameLength,    &
                                     Scope, Nullable)
```

```
         USE qt_ODBCKinds
         INTEGER (SQLRETURN) :: SQLSpecialColumns
         INTEGER (SQLHSTMT) :: stmt
         CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName
         INTEGER (SQLSMALLINT) :: IdentifierType, CatNameLength, SchemaNameLength,  &
                                  TableNameLength, Scope, Nullable
      END FUNCTION SQLSpecialColumns
   END INTERFACE

   INTERFACE SQLStatistics
      FUNCTION SQLStatistics( stmt, CatalogName, CatNameLength,    &
                                    SchemaName, SchemaNameLength,  &
                                    TableName, TableNameLength,    &
                                    Unique, Reserved )
         USE qt_ODBCKinds
         INTEGER (SQLRETURN) :: SQLStatistics
         INTEGER (SQLHSTMT) :: stmt
         CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName
         INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength, TableNameLength
         INTEGER (SQLUSMALLINT) :: Unique, Reserved
      END FUNCTION SQLStatistics

   END INTERFACE

   INTERFACE SQLTablePrivileges
      FUNCTION SQLTablePrivileges( stmt, CatalogName, CatNameLength,    &
                                         SchemaName, SchemaNameLength,  &
                                         TableName, TableNameLength )
         USE qt_ODBCKinds
         INTEGER (SQLRETURN) :: SQLTablePrivileges
         INTEGER (SQLHSTMT) :: stmt
         CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName
         INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength, TableNameLength
      END FUNCTION SQLTablePrivileges
   END INTERFACE

  INTERFACE

      FUNCTION SQLTables( stmt, szTableQualifier, cbTableQualifier,          &
                                szTableOwner, cbTableOwner,                  &
                                szTableName, cbTableName, szTableType, cbTableType )
         USE qt_ODBCKinds
         INTEGER (SQLRETURN) :: SQLTables
         INTEGER (SQLHSTMT) :: stmt
         CHARACTER (LEN=*) :: szTableQualifier, szTableOwner,  &
                              szTableName, szTableType
         INTEGER (SQLSMALLINT) :: cbTableQualifier, cbTableOwner,    &
                                  cbTableName, cbTableType
      END FUNCTION SQLTables

      ! added 14.10.2000, case: all pointer variables to be treated as Values (use LOC() function to specify a pointer to a variable)
      FUNCTION SQLTablesLP( stmt, szTableQualifier, cbTableQualifier,          &
                                  szTableOwner, cbTableOwner,                  &
                                  szTableName, cbTableName, szTableType, cbTableType )
         USE qt_ODBCKinds
         INTEGER (SQLRETURN) :: SQLTablesLP
         INTEGER (SQLHSTMT) :: stmt
         INTEGER (LP) :: szTableQualifier, szTableOwner,  &
                         szTableName, szTableType
         INTEGER (SQLSMALLINT) :: cbTableQualifier, cbTableOwner,    &
                                  cbTableName, cbTableType
      END FUNCTION SQLTablesLP

   END INTERFACE

   INTERFACE SQLTransact
      FUNCTION SQLTransact( env, dbc, fType )
         USE qt_ODBCKinds
         INTEGER (SQLRETURN) :: SQLTransact
         INTEGER (SQLHENV) :: env
         INTEGER (SQLHDBC) :: dbc
         INTEGER (SQLUSMALLINT) :: fType
      END FUNCTION SQLTransact
   END INTERFACE

END MODULE qt_ODBCInterfaces
! ----------------------------------------
! (C) Jörg Kuthe, Germany, 1999-2007. All rights reserved. www.qtsoftware.de
! ==========================================================================
```

Index