Jörg Kuthe

ForDBC

Fortran Database Connectivity

Stand: 15. September 2007

Nutzungsrechte, Haftungsbeschränkung, Copyright

Das Werk

Mit Werk wird nachfolgender Text und die Software auf beiliegenden Speichermedium bezeichnet. Der Umfang der Software ist am Ende dieses Textes näher beschrieben.

Nutzung

Durch den käuflichen Erwerb dieses Werks erhält der Käufer oder ein von ihm beauftragter Nutzer das Recht, die in diesem Werk enthaltene Information zu verwenden. Die Weitergabe dieses Werks in Form einer Kopie im Ganzen oder in Teilen ist jedoch untersagt. Dem Nutzer wird jedoch das Recht eingeräumt, die auf dem Speichermedium enthaltene Software in eigenen Programmen weiterzuverarbeiten, wenn in diesen und in der Dokumentation dazu, auf den Ursprung der verwendeten Software hingewiesen wird.

Haftung

Der Autor kann keine Garantie für die Richtigkeit der in diesem Werk enthaltenen Information geben. Er garantiert jedoch, dass er das Werk sorgfältig und gewissenhaft geschaffen hat. Die Haftung für Schäden, die aus der Nutzung der in diesem Werk enthaltenen fehlerhaften Informationen bzw. Software entstanden sind, beschränkt sich auf den entrichteten Kaufpreis des Nutzers.

Inhaltsverzeichnis

ForDBC - Fortran Database Connectivitiy

1. Die Verwendung von ODBC in Fortran 90/95 Programmen

Microsoft hat mit der Umsetzung und Erweiterung der X/Open und SQL Access Group Spezifikationen unter Windows eine Schnittstelle geschaffen, die es Datenbank-Herstellern ermöglicht, Programmierern eine standardisierte, offene Schnittstelle zu ihren Datenbanken anzubieten. Diese mit ODBC (Open Database Connectivity) bezeichnete Schnittstelle gibt Programmierern Funktionen an die Hand, mit denen sie unabhängig von den internen Satzformaten der Datenbanken auf diese mittels Standard SQL Befehlen zugreifen können (SQL = Structured Query Language). Microsoft dokumentiert ODBC u.a. in einem Produkt namens „Microsoft Developer Network“ (MSDN), auf die sich diese Einführung in die Benutzung der ODBC Schnittstelle stützt [ODBC96 und ODBC98].

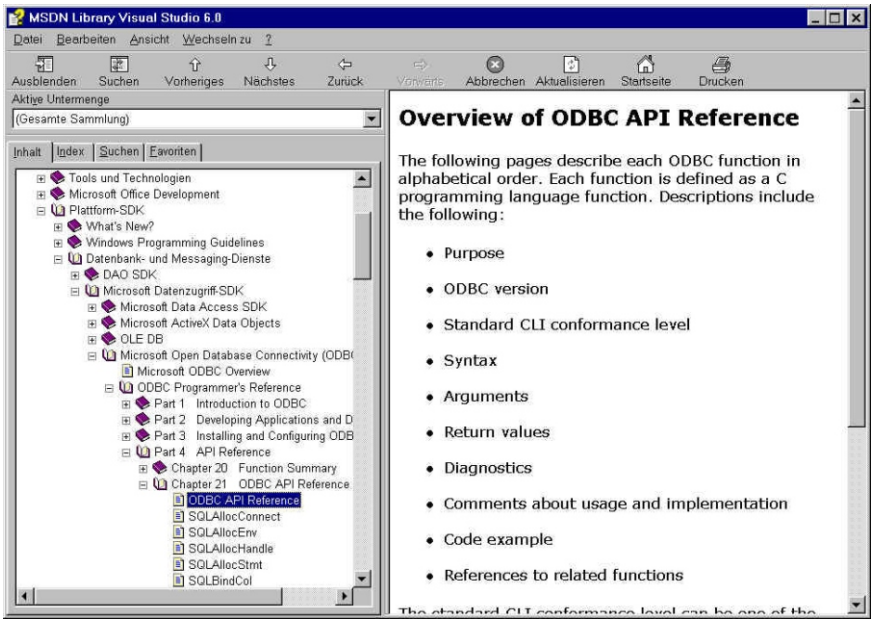

Abb. 1: Die Beschreibung ODBC API auf der MSDN CD-ROM [ODBC98].

Man findet diese Beschreibung aber auch im Internet unter:

⇨ http://msdn.microsoft.com/library/

Die Aufgabe dieser Einführung in die ODBC Programmierung ist es, Programmierern, die Fortran 90 bzw. Fortran 95 verwenden,

- die wesentliche Funktionalität der ODBC Schnittstelle zu erläutern

und

- konkrete Hilfestellung für die ersten Schritte zur Erstellung von ODBC Applikationen zu geben.

Dies ist eine hilfreiche Erleichterung, da sich Microsoft's Dokumentation im wesentlichen an C bzw. C++ Programmierer richtet und auch die ODBC spezifischen Datentypen, Funktionen und Konstanten im ODBC Application Programming Interface (ODBC API) nur für die Programmierung in C bzw. C++ definiert sind.

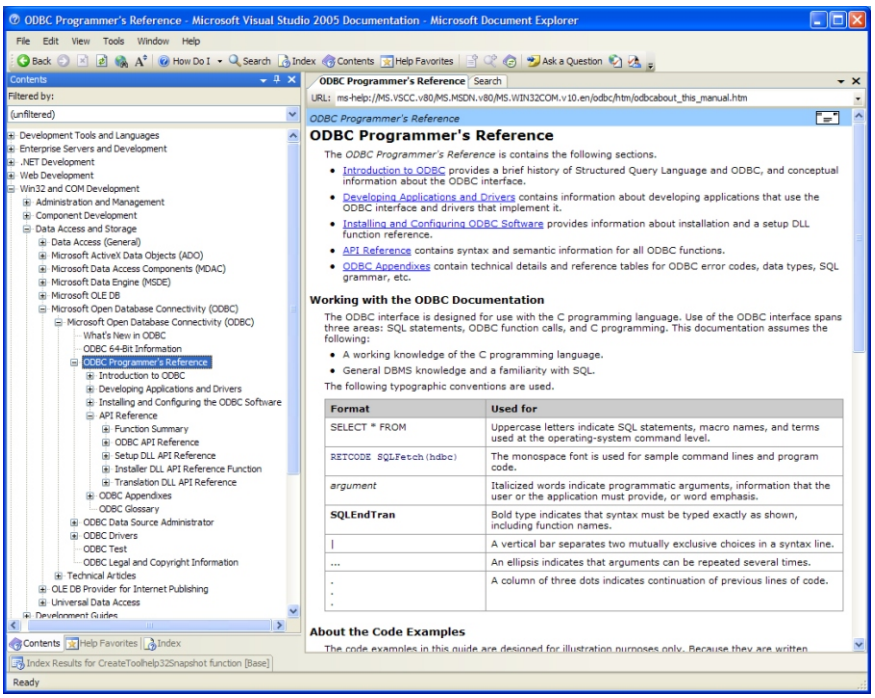

Abb. 2: Die Beschreibung ODBC API in der Online-Hilfe zu Visual Studio 2005.

Prinzipiell ist es auch möglich, ODBC Funktionen aus Fortran 77 Programmen aufzurufen, aber die Umsetzung und Bezugnahme auf die Originaldeklarationen in der ODBC API sind in Fortran 90 wesentlich leichter vorzunehmen.

Diese Abhandlung ist, um es nochmals zu betonen, eine Einführung. Ihre Benutzung ist nur in Verbindung mit einer vollständigen Beschreibung der ODBC Schnittstelle, wie sie bspw. in [ODBC96 oder ODBC98] zu finden ist, sinnvoll. Diese Einführung versucht die Grundlagen

- zur Installation von ODBC Software, inclusive Definition und Konfiguration von Datenquellen,
- zur Architektur der ODBC Schnittstelle,
- und zur Kommunikation mit Datenquellen mittels ODBC (Verbindungen, Transaktionen, Aufruf von ODBC Funktionen und Datentransfer)

zu vermitteln.

2. Installation von ODBC Software

Die Installation von ODBC (Open Database Connectivity) Software erfolgt mit einem treiber-spezifischen Programm (wird z.B. mit dem betreffenden Treiber oder mit dem verwendeten Datenbanksystem mitgeliefert).

Zur Konfiguration steht entweder ein ODBC Administrator Programm (z.B. für 32-Bit ODBC: ODBCAD32.EXE) oder ein applikations-spezifisches Setup-Programm zur Verfügung. U.a. ist ein ODBC Administrator Programm in Microsoft's ODBC Software Development Kit enthalten, das auch detaillierte Informationen zum Driver Setup Toolkit und zur ODBC Administration bereithält. [ODBC-I] gibt übersichtsmäßig Auskunft zum Setup von ODBC Applikationen.

Wesentlich ist, daß auf dem Zielrechner, auf dem eine ODBC Applikation unter Windows 9x, 2000, NT, XP, Vista etc. laufen soll, sowohl

- der Treiber Manager, d.h. die ODBC32.DLL und die Komponente CTL3D32.DLL,
- als auch der Treiber, z.B.ODBCJT32.DLL für Excel Dateien (.XLS), dBase Dateien (.DBF) usw.

vorhanden sein müssen.

2.1 Datenquelle (Data Source)

Mit dem Begriff „Datenquelle“ wird hier die Gesamtheit der Daten, auf die zugegriffen werden soll, das zugehörige Datenbankverwaltungssystem (Data Base Management System, DBMS) und seine Rechnerplattform sowie ggf. das Netzwerk, über das auf die Daten zugegriffen wird, bezeichnet. Um auf eine Datenquelle zugreifen zu können, benötigt der Treiber bestimmte Informationen, um die Verbindung herzustellen. Dies sind mindestens (gemäß ODBC Core Level - siehe Abschnitt „Treiber“)

- der Name der Datenquelle
- ggf. eine Benutzeridentifikationsnummer (User ID)
- und ggf. ein Paßwort.

ODBC Erweiterungen erlauben zusätzliche Angaben, bspw. der Netzwerkadresse oder weitere Paßwörter. Die Verbindungsinformation für jede Datenquelle ist in der ODBC.INI Datei oder in der Windows Registrierdatenbank (registry) gespeichert. Sie wird bei der Installation angelegt und in der Regel von einem eigenen Adminstrationsprogramm verwaltet (s..u.). Ein Abschnitt in dieser Initialisierungsdatei führt die verfügbaren Datenquellen auf. Z.B.:

```
[ODBC 32 bit Data Sources]
dBASE-Dateien=dBase-Treiber (*.dbf) (32 bit)
Excel-Dateien=Excel-Treiber (*.xls) (32 bit)
Waehrungen=Sybase SQL Anywhere 5.0 (32 bit)
```

In der Registrierdatenbank (registry) finden sich diese Einträge unter

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

bzw. unter

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

Weitere Abschnitte beschreiben jede Datenquelle detaillierter. Sie enthalten die Angabe des Treibernamens, ggf. eine Beschreibung, natürlich den Namen und Pfad der Datenbankdatei und weitere Angaben, die für den Aufbau der Verbindung notwendig sind. Z.B.:

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

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

[Waehrungen]
Driver=D:\sqlany50\win\wod50w.dll
UID=dba
PWD=sql
Description=WaehrungenDataSource
Start=dbeng50w
DatabaseFile=Waehrungen.DB
DatabaseName=DBWaehrungen
AutoStop=yes
```

```
TranslationName=Sybase SQL Anywhere 5.0 Transla
TranslationDLL=D:\sqlany50\win\wtr50w.dll
TranslationOption=1
Driver32=D:\sqlany50\win32\wod50t.dll
```

Als Datenquellen können somit alle Dateien dienen, für die ein entsprechender ODBC Treiber unter Windows installiert ist. ODBC Treiber werden i.A. mit allen gängigen Datenbanken, bspw. von Oracle, Sybase (SQLAnywhere u.a.), Informix, IBM (DB2) oder Microsoft (Access) mitgeliefert. Darüberhinaus gibt es u.a. auch ODBC-Treiber für Excel, Text- oder dBASE-Dateien, die bspw. in Microsoft's ODBC Software Development Kit zu finden sind.

3. Definition und Konfiguration von Datenquellen unter Windows

Damit ein ODBC Programm auf eine Datenquelle (Data Source) zugreifen kann, muß sie als solche zuerst definiert werden. Dies kann während der Laufzeit eines Programms erfolgen (zum Beispiel durch Aufruf der Funktion `SQLDriverConnect` unter Vorgabe eines Treibertyps, wie bspw. MS Access, vgl. Beispielprogramm T_ODBCDrvConnRd.f90) oder man legt die Datenquelle explizit über ein Hilfsprogramm des Betriebssystems an.

Unter Windows ruft man das ODBC Administrator Programm (z.B. für 32-Bit ODBC: ODBCAD32.EXE) auf. Dies ist i.A. in der Systemsteuerung von Windows zu finden („Startmenu: Start | Einstellungen | Systemsteuerung"; unter Windows XP ist der Eintrag dort unter „Verwaltung" zu finden).

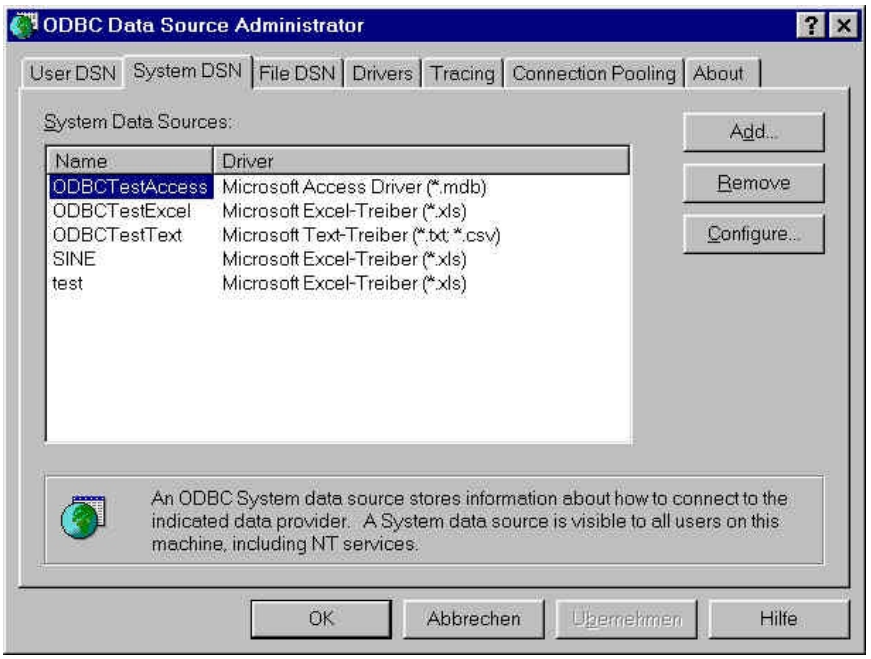

Abb. 3: Ansicht der Registerkarte „System DSN" nach Aufruf des ODBC Data Source Administrator-Programms.

Im ODBC Administrator Programm wird in einer der Registerkarten „User DSN", „System DSN" oder „File DSN" (DSN = Data Source Name) durch Betätigen der Taste „Add" (bzw. „Hinzufügen") der Dialog zur Auswahl des Treibers und anschließend das Dialogfenster zur Benennung und Definition der Datenquelle aufgerufen. Die Datenquelle bekommt hier

einen Namen, mit der die Datenbank und ihr Treiber vom ODBC Manager identifiziert werden kann. Das ODBC Administrator Programm legt die angelegte Information in der ODBC.INI Datei bzw. in der Windows Registrierdatenbank (registry) ab.

Namen für Datenquellen können auf Benutzerebene („User DSN“), Systemebene („System DSN“ und Dateiebene („File DSN“) angelegt werden, wodurch dann die Zugriffsmöglichkeiten und –rechte der Benutzer vorgegeben sind.

3.1 Excel-Datenquellen

Für Microsoft Excel Arbeitsblätter („Worksheets“) und Arbeitsmappen („Workbooks“) sind bei der Benennung der Datenquellen einige Besonderheiten zu beachten:
Man kann auf Arbeitsblätter (Worksheets), auf Arbeitsblätter innerhalb einer Arbeitsmappe (Workbooks - ab Excel 5.0), auf beliebige (unbenannte) und auf spezifizierte Zellenbereiche (z.B. A1:C14) in einem Arbeitsblatt zugreifen, wobei bestimmte Konventionen einzuhalten sind:

- Zellenbereichsangaben sind durch ein Komma abzutrennen, z.B. „C:\EXCEL\VERKAUF.XLS, A1:C14“.
- Für ein Arbeitsblatt in einer Excel 5.0 oder 7.0 Arbeitsmappe, muß das Arbeitsblatt durch seinen Namen gefolgt von einem „$“ Zeichen spezifiziert werden. Z.B, „BLATT1$“. Zellenbereiche werden hier durch direktes Anhängen an die Arbeitsblattangabe bezeichnet. Z.B.: „BLATT1$A1:C14“.
- Um einen benannten Zellenbereich eines Excel Arbeitsblattes anzusprechen, muß der Name vor dem Öffnen durch das externe Programm vorhanden sein (innerhalb von Excel kann dies durch Markierung des Zellenbereichs und dann durch Menüauswahl Einfügen | Namen | Festlegen erfolgen). Der dort angegebene Name ist gleichzeitig der Name der extern verwendeten Datentabelle. Individuelle Felder und Sätze eines Arbeitsblattes können extern nicht direkt adressiert werden.

Des weiteren existieren spezielle Beschränkungen existieren beim Gebrauch von Excel Arbeitsblättern:

- Mehrfachzugriff (durch mehrere Benutzer) ist nicht möglich (die Database Engine unterstützt keinen Mehrbenutzerbetrieb).

Anmerkung: Die Dokumentation des Zugriffs auf EXCEL-Tabellen in der ODBC API ist mehr als dürftig. Vgl. das Beispielprogramm T_ODBCExcel.f90.

4. Der Aufbau der ODBC Schnittstelle

Die Open Database Connectivity (ODBC) Schnittstelle erlaubt Applikationen mittels Structured Query Language (SQL - eine Beschreibung ist bspw. in [SQL] zu finden) auf Datenquellen diverser Datenbanksysteme (Data Base Management Systeme; DBMS) zugreifen. Der Vorteil gegenüber dem direkten Zugriff auf eine Datenquelle gegenüber dem Zugriff über ODBC ist die Unabhängigkeit von datenbankspezifischen Datei- und Satzstrukturen bei Verwendung einer standardisierten Abfragesprache (nämlich SQL).

Microsoft stellt dem Programmierer eine Softwareschnittstelle, das ODBC Application Programming Interface (ODBC API) zur Verfügung, die im wesentlichen aus Funktionen

- zur Verbindung zur Datenbank bzw. Datenquelle
- zum Anlegen und Verwalten von Speicherbereichen und -zuordnungen zur Datenkommunikation
- zum Zugriff auf Daten der Datenquelle
- zur Verwaltung von Transaktionen
- zur Fehlerbehandlung

besteht.

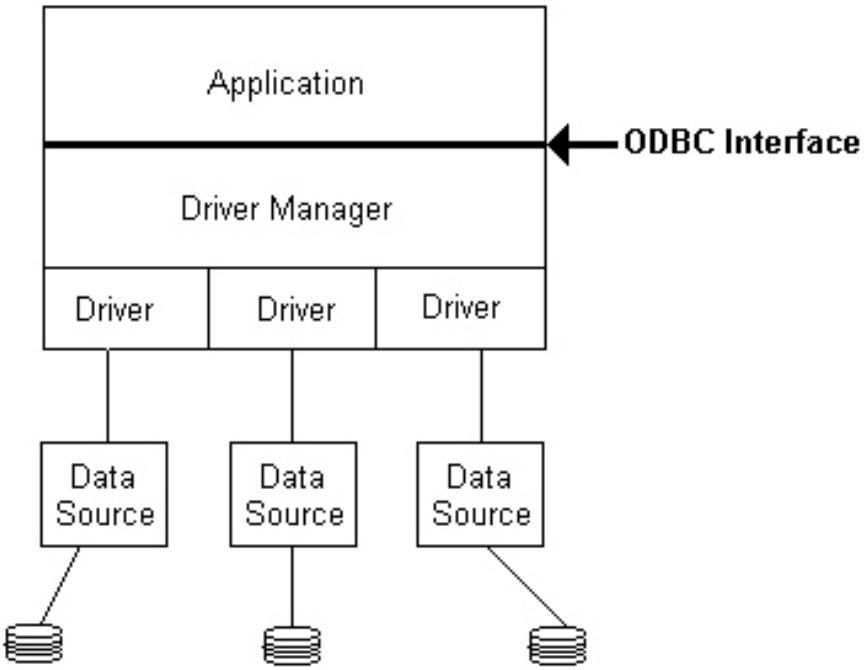

Abb. 4: Grundsätzlicher Aufbau von ODBC.

Neben der ODBC API ist der von Microsoft (oder mitunter auch vom Datenbankhersteller) in Form einer DLL bereitgestellter

- Treiber (Driver) Manager (ODBC32.DLL) inclusive Import Library (ODBC32.LIB)

sowie ein von Seiten des DBMS zur Verfügung zu stellender

- Treiber (Driver)

notwendig.

4.1 Treiber Manager

Der Treiber Manager (driver manager) hat die wesentlichen Aufgaben,

- den bzw. die Treiber zu laden, wenn die Applikation eine der ODBC Connect Funktionen `SQLBrowseConnect`, `SQLConnect` oder `SQLDriverConnect` aufruft,
- diverse ODBC Initialisierungen vorzunehmen

- und zu überprüfen, ob die Aufrufe von ODBC Funktionen gültige Parameter enthalten und in korrekter Reihenfolge erfolgen. Er wertet außerdem die ODBC.INI bzw. die Registrierdatenbank (Registry) aus, um Datenquellen spezifische DLLs zuzuordnen.

4.2 Treiber

Auch der Treiber (driver) ist eine DLL. Sie wird vom Datenbankhersteller zur Verfügung gestellt. Sie beinhaltet ODBC Funktionen, die auf die herstellerspezifische Datenbank abgestimmt sind. Ihre wesentlichen Funktionen sind

- die Herstellung der Verbindung zur Datenquelle (Connect),
- die Übermittlung von Ergebnissen (results) auf Anforderungen/Abfragen (requests) an die Datenquelle,
- die Umwandlung von Daten in ein gewünschtes Format,
- die Formatierung von Fehlern in die Standard-Fehler-Kodierung und Rückgabe an die Applikation,
- die Deklaration und Verwaltung von Cursors (SQL Cursor) (die Funktion ist i.A. für die Applikation nicht sichtbar),
- und die Auslösung von Transaktionen, wenn die Datenquelle explizite Transaktions-Initiierung erfordert (die Funktion ist i.A. für die Applikation nicht sichtbar).

Es werden zwei Treibertypen zugelassen:

- Single-tier (Einfachbindung): Der Treiber bearbeitet sowohl ODBC API Aufrufe als auch SQL Kommandos.
- Multiple-tier (Mehrfachbindung): Der Treiber bearbeit die ODBC API Aufrufe und übergibt die SQL Kommandos an die Datenquelle.

Beide Typen können innerhalb eines Systems verwendet werden. Typischerweise trifft man Multiple-tier in Internet- und Intranet-Applikationen, wo bspw. eine Applikation SQL Befehle an einen Gateway-Rechner sendet, der diese an einen dritten Rechner, den Datenbankserver weiterleitet, wo sie dann bearbeitet werden. Beide Treibertypen haben gemein, daß sowohl die Applikation, der Treiber und auch der Treiber Manager auf dem Client-Rechner ablaufen.
Aus der Sicht der Applikation besteht zwischen diesen Typen kein Unterschied, was für den Programmierer bedeutet, daß er die Applikation auf einem einzigen Rechner entwickeln und testen und anschließend eine Verteilung von Datenbank und Datenbankzugriffssoftware im Netz vornehmen kann.

4.3 ODBC Konformitätsebenen (ODBC Levels)

ODBC definiert Konformitätsebenen (conformance levels) für Treiber bzgl. der ODBC API und bzgl. der SQL Grammatik (incl. SQL Datentypen). Auf diese Weise können gewisse Funktionalitätsstandards gewährleistet sein, so daß ein DBMS Hersteller nicht gezwungen wird, die vollständige ODBC API und SQL Funktionalität zu bieten, wenn seine Datenbank dies bspw. nicht zu liefern imstande ist. Der Programmierer kann mittels der ODBC API Funktionen `SQLGetInfo`, `SQLGetFunctions` und `SQLGetTypeInfo` die Funktionalität des geladenen Treibers

ermitteln.
Die ODBC API definiert einen Satz Kernfunktionen (core) und Funktionen der X/Open und SQL Access Group Call Level Interface Spezifikation. Diese und weitere Funktionen sind in zwei Funktionsgruppen der Ebene 1 (Level 1) und der Ebene 2 (Level 2) definiert, wobei die Funktionen der Ebene 2 die der Ebene 1 enthalten. In den meisten Fällen genügt, wenn der DBMS Treiber die Funktionen der Ebene 1 bereitsstellt. Das Kapitel "ForDBC Funktionsübersicht" enthält eine Liste aller in ForDBC implementierten ODBC Funktionen und ihre ODBC Level.

Ähnlich besteht die Funktionalität der SQL Grammatik aus Kernfunktionen (core SQL grammar), die in etwa den Spezifikationen der X/Open und der SQL Access Group SQL CAE von 1992 entsprechen. ODBC faßt die SQL Funktionalität in zwei weitere Gruppen, der Minimal-Grammatik (minimum SQL grammar) und der erweiterten Grammatik (extended SQL grammar) zusammen. In vielen Fällen genügen die Funktionen und Datentypen der Core SQL Grammar.

4.4 Verbindungen und Transaktionen

Bevor eine Applikation ODBC nutzen kann, muß ODBC initialisiert werden, indem

- eine **Umgebungsidentifikationsnummer** (environment handle) angelegt wird (*hEnv*).

Für die Kommunikation mit der Datenquelle ist eine

- **Verbindungsidentifikationsnummer** (connection handle) notwendig (*hDBC*).

Mit beiden Nummern (*hEnv* und *hDBC*) kann die Applikation anschließend operieren, um die Datenquelle bzw. die Verbindung anzusprechen. Es können mehrere Verbindungen gleichzeitig zur selben Datenquelle geöffnet werden.
Zu jeder Verbindung gehört ein eigener Transaktionsrahmen (transaction space).
Innerhalb einer aktiven Verbindung können eine oder mehrere Anweisungen (SQL statements) ausgeführt werden.

Die Transaktionen werden vom Treiber für jede aktive Verbindung verwaltet. Ein COMMIT kann entweder automatisch (d.h. nach Abschluß einer jeden SQL Anweisung; Attribut: `SQL_ATTR_AUTOCOMMIT`) oder explizit von der Applikation ausgeführt werden (letzteres gilt auch für ein ROLLBACK). Nach einem COMMIT oder einem ROLLBACK, werden alle SQL Anweisungen zurückgesetzt (Reset).

Zwischen verschiedenen Verbindungen kann auch, während Transaktionen stattfinden, gewechselt werden.

5. Grundsätzliches zum Aufruf von ODBC API Funktionen in Fortran

Die Namen aller Funktionen der ODBC API beginnen mit „SQL“. Die Definitionen und Deklarationen von ODBC Konstanten, Typen und Funktionsprototypen zu diesen Funktionen sind in den C-Header-Dateien SQL.H, SQLEXT.H und WINDOWS.H zu finden (diese werden bspw. bei vielen gängigen C/C++ Compilersystemen für Windows mitgeliefert). C Programme müssen diese Header-Dateien beinhalten.

Fortran 90/95 Programmierern werden hier entsprechende Fortran 9x Module bereitgestellt, die in ODBC Applikationen mittels USE Befehl einzubinden sind:

```
USE qt_ODBC
```

Der Quellcode des Moduls befindet sich in der Datei

qt_ODBC.f90.

Das Modul qt_ODBC beinhaltet Referenzen zu weiteren Modulen, z.B. die Definition der ODBC spezifischen Datentypen (KINDs) in

qt_ODBCKinds
(siehe Datei qt_ODBCKinds.f90)

und Konstanten (PARAMETERs) in

qt_ODBCDefs
(siehe Datei qt_ODBCDefs.f90)

Das Modul verwendet grundlegende C und Windows Datentypen (KINDs), die in den Modulen

qt_CKinds
(siehe Datei qt_CKinds.f90)

und

qt_Win32Kinds
(siehe Datei qt_Win32Kinds.f90)

definiert werden. So definiert bspw. das Modul qt_Ckinds den C Datentyp LONG,

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

den seinerseits das Modul qt_Win32Kinds verwendet, um den Datentyp LP zu definieren:

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

In den Modulen qt_ODBCKinds und qt_ODBC werden diese Datentypen benutzt, um weitere ODBC spezifische Datentypen und Konstanten zu deklarieren bzw. zu definieren. Z.B.:

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

Dies erscheint verwirrend und fragwürdig, da auf diese Weise letztendlich alle Typen, Konstanten, Variablen auf die grundlegenden Datentypen, wie INTEGER*4, INTEGER*2 oder REAL*4 abgebildet werden. So ist zum Beispiel eine Variable vom Type SQLHENV nichts anderes als ein 4 Byte langer INTEGER Typ (INTEGER*4). Der Grund, warum die ODBC Datentypen (oder auch andere Windows Datentypen) eigene Namen

bekommen, ist in der Möglichkeit der flexibleren Weiterentwicklung der ODBC Software zu suchen. Denn einen Vorteil birgt die derart hierarchisch aufgebauten Datentypdeklarationen: wenn die abgeleiteten Datentypen sich bspw. beim Schritt auf ein neues Betriebssystem ändern sollten, kann durch Änderung der zugrundeliegenden Definitionen (bspw. in qt_CKinds) leicht eine komplette Änderung der abgeleiteten Datentypen erfolgen (dieser Fall trat bspw. beim Umstieg von Win16 zu Win32 auf). Aus Gründen der Referenz zur Originaldokumentation der ODBC Schnittstelle wurde also versucht, eine Analogie in den Fortran 90/95 Definitionen und Deklarationen zu denen in C herzuleiten. Ein C Programmauszug der Art

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

entspricht in Fortran 90/95 folgendes:

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

Mitunter sind aufgrund der Eigenheiten von Fortran namentlich abgewandelte Varianten der ODBC Funktionsnamen zu verwenden. Dies betrifft grundsätzlich alle ODBC Funktionen, die für ein und dasselbe Argument verschiedene Variablentypen zulassen, bspw. `SQLBindCol`. ForDBC definiert hier die Varianten `SQLBindColI2`, `SQLBindColI4`, `SQLBindColChar` etc..

Eine weitere Besonderheit ist zu beachten, wenn der Typ SQLPOINTER verwendet wird, z.B.:

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

Wie dem Beispiel zu entnehmen ist, wird hier der Variable ValPtr, die vom Typ SQLPOINTER ist, die Speicheradresse der Variablen Value zugewiesen, da die Funktion `SQLSetConnectAttr` bei Angabe des Attributs SQL_ATTR_QUIET_MODE entweder einen Null-Pointer (oder ein Window-Handle) erwartet (und nicht die Variable Value selbst). Allgemein gilt, daß eine Variable, die vom Typ SQLPOINTER ist, entweder die Adresse einer Variablen bzw. eines Feldes oder einen Wert übergibt (meist wird dies durch ein anderes Argument der Funktion gesteuert). Die Adresse kann i.A. mit der Funktion LOC() (bei manchen Compilern mit der ehemaligen VAX Funktion POINTER() – nicht zu verwechseln mit dem Fortran 90 POINTER Befehl) bestimmt werden. Ob ein Pointer oder ein

Wert zu übergeben ist, ist der jeweiligen Funktionsbeschreibung zu entnehmen.

Die Funktionsschnittstellen (INTERFACEs) sind im Modul

qt_ODBCInterfaces
(siehe Datei qt_ODBCInterfaces.f90)

niedergelegt. Im Anhang A wird das Modul aufgelistet. Es zeigt die verfügbaren Funktionen und die notwendigen Typdeklarationen für ihre Argumente.

Compiler-spezifische Definitionen sind im Modul

qt_ODBC_Compiler
(siehe Dateien qt_ODBC_*compiler*.f90
mit *compiler* = DVF, FTN, IVF, LF90, LF95 usf.)

zu finden. Da der Modulname immer der gleiche ist, die Dateinamen in Abhängigkeit vom verwendeten Compiler jedoch variieren, ist so bei einem Compilerwechsel eine compiler-spezifische Anpassung Ihrer ODBC Applikationen i.A. nicht nötig.

Der Treiber Manager ermöglicht einer Applikation den Zugriff auf ODBC Treiber mithilfe einer entsprechenden .DLL (z.B. ODBC32.DLL). **Für das Binden (Link) benötigt die Applikation hierzu meist die zugehörige Importbibliothek (z.B. ODBC32.LIB).**
Den Zugriff auf den Treiber und somit die Verbindung zu der dem Treiber zugehörigen .DLL verwaltet und leitet der Treiber Manager.

5.1 Datentransfer zwischen Applikation und ODBC Treiber bzw. Treiber Manager

Der "Transfer" der Daten zwischen Applikation und ODBC Treiber bzw. Treiber Manager erfolgt über die Argumente der Funktionen bzw. der Speicherbereiche, die durch sie den Funktionen mitgeteilt werden. Dies können also in unseren Fortran Programmen die üblichen INTEGER, REAL oder CHARACTER Variablen sein. Mitunter bekommen wir es auch mit Zeigern zu tun. D.h. wir müssen die Startadressen von Speicherbereichen angeben. Auch die Verwendung von CHARACTER Argumenten (strings) erfordert Aufmerksamkeit, da wir uns an die C-typische Behandlung anpassen und gewisse Regeln beachten müssen.

5.1.1 CHARACTER bzw. string Argumente

Diverse ODBC Funktionen („Treiberfunktionen“) erwarten Zeichenketten (strings) oder sonstige Werte beim Aufruf bzw. geben solche zurück. Z.B.:

```
szStmt = "SELECT str FROM Tabelle1"//CHAR(0)
iRet = SQLExecDirect( hStmt, szStmt, SQL_NTSL )
! Die operative Länge von szStmt wird über die
! terminierende Null bestimmt.
!
! Wir verwenden hier die LONG Version von SQL_NTS,
! da das INTERFACE von SQLExecDirect einen langen
! INTEGER (SQLINTEGER) erfordert.
```

Der ODBC Funktion wird durch die Angabe der Variablen bzw. des Feld intern ihre Speicheradresse mitgeteilt. Aber im Falle von CHARACTER Argumenten nicht ihre Länge, wie dies in Fortran üblich ist ("hidden length

argument"). Daher erfolgt die Angabe der Länge von CHARACTER Argumenten separat. Für die Längenangabe gelten folgende Regeln:

- Die Länge muß größer oder gleich 0 sein. Sie gibt die aktuelle Anzahl der Zeichen (bytes, 8 bit characters) der Daten in der CHARACTER Variablen an. Wenn die Länge gleich 0 ist, bedeutet dies die Übergabe einer „leeren“ Zeichenkette (empty string). Zeichenketten brauchen dann nicht null-terminiert zu sein (d.h. letztes Zeichen muß nicht einem ASCII 0 Wert - CHAR(0) - entsprechen).
- SQL_NTS bzw. SQL_NTSL: Wenn seine Länge mithilfe des Werts der Konstanten SQL_NTS (**N**ull **T**erminated **S**tring) der Treiberfunktion mitgeteilt wird, muß die übergebene Zeichenkette null-terminiert sein. Die operative Länge der CHARACTER Variablen wird dann vom Treiber intern ermittelt (eben anhand der terminierenden Null). Anmerkung: SQL_NTSL ist die 4-Byte INTEGER Variante. SQL_NTS die 2-Byte INTEGER Variante.
- **Zeichenketten (CHARACTER) werden immer null-terminiert zurückgegeben.**

5.1.2 Fehlende Daten (missing values)

Datenbanken erlauben üblicherweise auch die Kennzeichnung fehlender Daten. D.h., Tabellen können Zellen bzw. Elemente enthalten, denen kein Wert zugeordnet ist. Für diesen Zustand gibt es in Fortran kein Äquivalent. Man behilft sich oft durch die Verwendung eines bestimmten Variablenwerts, der den Zustand des "fehlenden Werts" markiert. Z.B. wird ein Wert zu -999999. gesetzt, um einen fehlenden Meßwert anzuzeigen.
Da die ODBC Funktionen i.A. für die Angabe eines oder mehrerer Werte nicht nur ein Argument vorsehen, sondern zwei, nämlich das zusätzliche Längenargument, wird letzteres dazu benutzt, einen fehlenden Wert zu kennzeichnen:

- SQL_NULL_DATA: Wenn die Längenangabe den Wert der Konstanten SQL_NULL_DATA hat, teilt dies mit, daß der Inhalt der Variable ignoriert werden und stattdessen der NULL Datenwert verwendet werden soll. Diese Konstante darf nur verwendet werden, um einen NULL Wert als Parameter für einen SQL Befehl einzugeben.

5.1.3 Sonstige Hinweise bezüglich des Inhalts von Argumenten

- Die Verwendung von ASCII 0 Zeichen innerhalb übergebener CHARACTER Daten sollte unterbleiben, da ASCII 0 als Terminierung von strings verwendet wird.
- Sofern nichts anderes vorgeschrieben wird, ist es erlaubt, statt eines Arguments den Wert 0 anzugeben, um einen "Null Pointer" zu übergeben. In diesem Falle wird auch ein mögliches Längenargument ignoriert. Allerdings lassen die ForDBC Fortran INTERFACEs zu den ODBC Funktionen dies nur bedingt zu.
- Sofern notwendig, werden Daten vor einer Rückgabe umgewandelt. Es wird dann auch die Länge der konvertierten Daten nach der Umwandlung zurückgegeben (in Bytes). Im Fall von Zeichenketten (CHARACTER), wird die terminierende Null nicht mitgezählt.
- Eine Längenangabe wird im Falle eines Eingangsarguments (on input) vom Treiber ignoriert, wenn der Datentyp bzw. die Datenstruktur per Definitionem in C eine feste Länge hat (z.B. gilt dies für Ganzzahlen,

Gleitkommazahlen oder Datumsstrukturen). Wie gesagt, im Fall einer Rückgabe kann ein Längenargument mittels SQL_NULL_DATA einen fehlenden Wert kennzeichnen.

- Wenn ein Argument zu klein ist (z.B. eine CHARACTER Variable), versucht der Treiber die zurückzugebenden Daten zu verkürzen (truncate). Wenn dies ohne Verlust von signifikanten Daten vonstatten geht, gibt der Treiber die verkürzten Daten (truncated data) im Argument zurück, spezifiziert die Länge der nicht-verkürzten Daten und signalisiert diesen Zustand durch den zurückgegebenen Funktionswert SQL_SUCCESS_WITH_INFO.
- Falls ein Verlust von signifikanten Daten auftritt, wird im Argument nichts zurückgegeben, und es erfolgt auch keine Längenangabe. Stattdessen wird als Funktionswert SQL_ERROR zurückgegeben (Fehlerkonstanten - siehe Abschnitt „Rückgabewerte der ODBC API Funktionen“).

5.2 Datentypen

Da die Datentypen der Datenquelle mitunter unterschiedlich von denen der ODBC Spezifikation sind (z.B. mag es einen SQL Datentyp MONEY geben, der in C oder Fortran nicht existiert), ist eine Umwandlung notwendig. Der Treiber bildet daher datenquellenspezifische SQL Datentypen auf ODBC SQL Datentypen ab (sind in der ODBC SQL Grammatik definiert). Informationen über die Typen können mittels der ODBC API Funktionen `SQLGetTypeInfo`, `SQLColAttributes`, `SQLDescribeCol` und `SQLDescribeParam` abgefragt werden.

Jedem SQL Datentyp entspricht ein ODBC Datentyp (basierend auf C Datentypen). Z.B. entspricht dem SQL Spaltentyp FLOAT der ODBC Datentyp SQL_C_FLOAT. Der Treiber setzt voraus, daß jedem C Datentyp einer Variable der SQL Datentyp der Spalte (column) bzw. des Parameters entspricht, der der Variable zugeordnet wurde. Wenn der C Datentyp der Variable nicht mit dem voreingestellten übereinstimmt, kann der korrekte C Datentyp mit dem TargetType Argument der `SQLBindCol`, `SQLGetData` oder der `SQLBindParameter` Funktionen angegeben werden. Die Konvertierung des C Datentyps in den Speichertyp der Datenquelle und umgekehrt erledigt der Treiber.

5.3 Identifikation der Umgebung, von Verbindungen und Befehlen (Environment, Connection, and Statement Handles)

Der Treiber Manager legt auf Anforderung der Applikation Speicher (allocate memory) für eine ODBC Umgebung (environment), für jede Verbindung (connection) und für jeden SQL Befehl an. Diese Speicherbereiche werden durch Nummern (handles) von Seiten der Applikation identifiziert (d.h. sie erhält diese nach Aufruf der entsprechenden ODBC API Funktionen).

Die **Umgebungsidentifikationsnummer** (environment handle) identifiziert einen Speicherbereich für globale Informationen. Dieser enthält u.a. die gültigen und die aktiven Verbindungsidentifikationsnummern (connection handles). Sie ist vom Typ HENV (ODBC v1.x/v2.x) bzw. SQLHENV (ODBC v3.x) - beide Typen basieren übrigens auf INTEGER und sind letztendlich gleich). Eine

Applikation besitzt höchstens eine Umgebungsidentifikationsnummer, und sie muß vor der Verbindung zur Datenquelle angefordert worden sein.

Der Speicherbereich für die Information zu einer spezifischen ODBC Verbindung wird über die **Verbindungsidentifikationsnummer** (connection handle) erkannt. Sie ist vom Typ HDBC (ODBC v1.x/v2.x) bzw. SQLHDBC (ODBC v3.x) und muß vor der Verbindung zur Datenquelle angelegt werden. Eine Applikation kann mehrere Verbindungsidentifikationsnummern (d.h. mehrere Verbindungen zu Datenquellen) besitzen, die jedoch alle mit der einen **Umgebungsidentifikationsnummer** der Applikation verknüpft sind.

Speicherbereiche für SQL Befehle werden über die **Befehlsidentifkationsnummer** (statement handle) identifiziert. Zu jedem SQL Befehl wird vor Ausführung des Befehls eine Befehlsidentifkationsnummer angelegt. Sie ist vom Typ HSTMT (ODBC v1.x/v2.x) bzw. SQLHSTMT (ODBC v3.x) und mit einer spezifischen Verbindungsidentifikationsnummer verknüpft.

5.4 Rückgabewerte der ODBC API Funktionen

Der Erfolg (success), Zustand bzw. Warnung (warning) oder Mißerfolg (failure) einer aufgerufenen ODBC API Funktion wird über den Funktionswert mitgeteilt. Es gibt folgende fest definierte Rückgabewerte (die Konstanten sind im Modul qt_SQLDefs bzw. in der zugehörigen Datei qt_SQLDefs.f90 definiert):

```
SQL_SUCCESS
SQL_INVALID_HANDLE
SQL_SUCCESS_WITH_INFO
SQL_STILL_EXECUTING
SQL_NO_DATA_FOUND
SQL_NEED_DATA
SQL_ERROR
```

Im Falle der Rückgabewerte SQL_SUCCESS_WITH_INFO und SQL_ERROR kann über die Funktion `SQLError` (ODBC v1.x/v2.x) bzw. `SQLGetDiagRec` (ODBC v3.x) zusätzliche Information über den Fehler abgefragt werden.

5.5 Zugriff auf die Datenquelle - Grundlegende ODBC Applikationsstruktur

Damit eine ODBC Applikation auf eine Datenquelle zugreifen kann, sind grundsätzlich folgende Schritte notwendig:

1. Verbindung zur Datenquelle herstellen. Hierzu ist die Angabe der Datenquelle und ggf. zusätzlicher Information notwendig, um die Verbindung aufzubauen.

2. SQL Befehle ausführen.
Dazu wird der Befehl im Klartext üblicherweise in einer CHARACTER Variable abgelegt und ggf. weitere Parameter beim Aufruf der ODBC Funktion übergeben.
Wenn der Befehl einen Ergebnissatz (result set) zur Folge hat, muß zuvor ein SQL Cursor in der Applikation angelegt werden (dies geschieht mitunter implizit durch das „Column bzw. Parameter Binding“).

Die Applikation schickt den Befehl zur sofortigen Ausführung oder zur Vorbereitung an den Treiber bzw. Treiber Manager.
Falls ein Ergebnissatz erstellt wird, besteht die Möglichkeit Attribute desselben (bspw. Anzahl der Spalten, deren Name und Typ) abzufragen. Jeder Spalte (column) des Ergebnissatzes ist ein Speicherort zugeordnet.
Im Fehlerfall wird die Fehlerinformation vom Treiber abgefragt und es besteht die Möglichkeit, geeignete Maßnahmen zur Behandlung durchzuführen.

3. Jede Transaktion ist mit einem COMMIT oder ROLLBACK abzuschließen, sofern die Datenbankverbindung nicht im AUTOCOMMIT Modus geöffnet ist.

4. Nach Abschluß der Interaktion mit der Datenquelle ist die Verbindung zu beenden.

Nachfolgendes Diagramm führt die ODBC API Funktionen für die Verbindung zur Datenquelle (connect), die Ausführung von SQL Befehlen (process) und für das Beenden der Verbindung (disconnect) auf (basierend auf ODBC v1.x/v2.x).

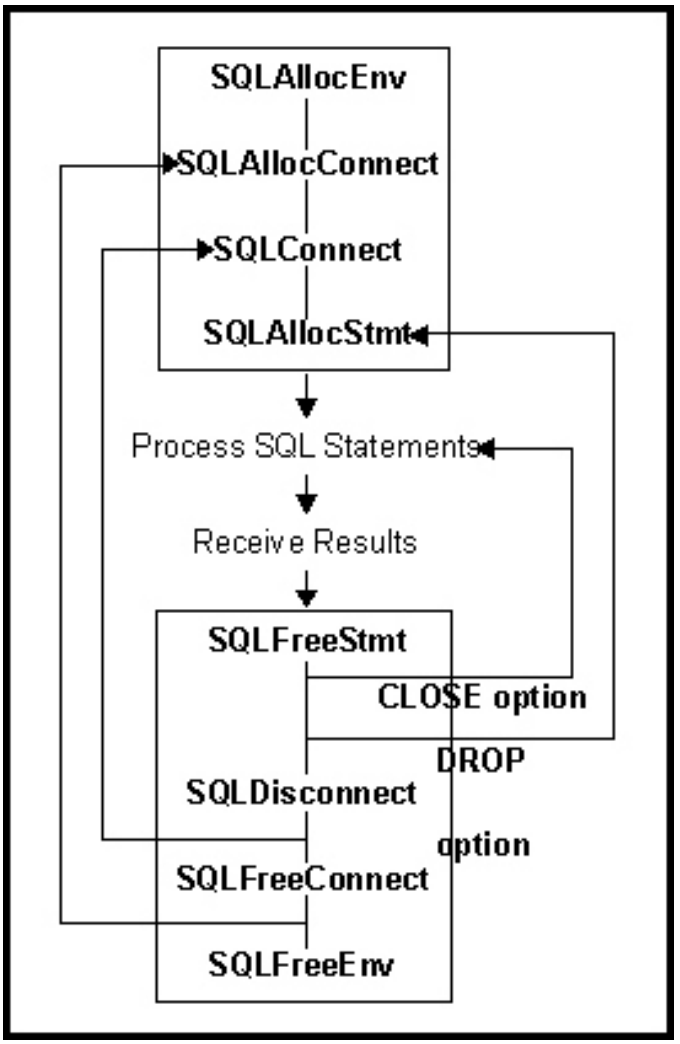

Abb. 5: ODBC Programmstruktur (ODBC v1.x/v2.x)

5.5.1 Die Initialisierung der ODBC Umgebung

Der erste Schritt einer ODBC Applikation ist die Initialisierung der ODBC Umgebung unter Zuordnung der Umgebungsidentifikationsnummer

(environment handle). Nachdem die notwendige Variablen deklariert wurden

```
INTEGER (KIND=HENV) :: env = SQL_NULL_HENV
INTEGER rtc
```

erfolgt der Aufruf zur Erzeugung der ODBC Umgebung unter ODBC v1.x/v2.x wie folgt:

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

Die Funktion `SQLAllocEnv` gibt im Erfolgsfall (rtc = SQL_SUCCESS) die Umgebungsidentifikationsnummer im Argument (env) zurück. Ab ODBC v3.0 steht ein neuer Befehl zur Initialisierung zur Verfügung:

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

Hinweis: die Initialisierung einer ODBC Umgebung sollte in einer Applikation nur ein einziges Mal stattfinden.

5.5.2 Der Verbindungsaufbau

Nachdem die ODBC Umgebung initialisiert wurde, kann die Verbindung zum Treiber hergestellt werden. Die für die Verbindungsidentifikationsnummer (connection handle) notwendige Deklaration lautet bspw.:

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

oder

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

Für die Verbindung wird die Funktion `SQLAllocConnect` (ODBC v1.x/v2.x) aufgerufen, die als Argument die zuvor angelegte Umgebungsidentifikationsnummer (environment handle) benötigt.

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

Falls kein Fehler auftritt (rtc = SQL_SUCCESS), wird im zweiten Argument der Funktion die Verbindungsidentifikationsnummer zurückgegeben. Ab ODBC v3.x kann man auch die Funktion `SQLAllocHandle` verwenden:

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

Schließlich folgt dann der eigentliche Verbindungsaufbau, z.B. mithilfe der Funktion `SQLConnect`. Die Funktion fordert die Angabe des Namens der Datenquelle, eine Benutzeridentifikationsnummer (User ID oder auch Login ID) und ein Paßwort. Z.B.:

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

Die Datenquelle im obigen Beispiel heißt „Waehrungen“, die Benutzeridentifikation (Login ID) „dba“ und das Paßwort „sql“. Sämtliche Längenangaben der Zeichenketten erfolgen intern automatisch gemäß Angabe SQL_NTS (null terminated string). Man beachte, daß sämtliche Zeichenketten null-terminiert sind („//CHAR(0)“ wurde angehängt).

Wenn `SQLConnect` aufgerufen wird, sucht der Treiber Manager in der ODBC.INI Datei bzw. in der Registrierdatenbank nach weiteren Angaben

der zugehörigen Treiber-DLL und lädt die DLL. Werden für die Verbindung zur Datenquelle mehr Informationen benötigt, versucht der Treiber Manager diese ebenfalls aus dem entsprechenden Abschnitt der Treiber-DLL zu ermitteln. Falls der Name der Datenquelle nicht in der ODBC.INI Datei bzw. Registrierdatenbank gefunden oder auch wenn kein Name angegeben wurde, sucht der Treiber Manager nach der voreingestellten Spezifikation für Datenquellen und lädt den entsprechenden Treiber. Sollten auch keine geeigneten Voreinstellungen vorhanden sein, wird ein Fehler zurückgegeben.

ODBC bietet weitere, über die Spezifikationen der X/Open und SQL Access Gruppe hinausgehende Funktionen, um Verbindungen zu Datenquellen herzustellen, bspw. wenn mehr als die drei oben genannten Parameter (Quellenname, UserId, Paßwort) benötigt werden. Man findet dazu in [ODBC-C] detaillierte Informationen.

Will man beispielsweise erst während der Laufzeit den Dateinamen einer Datenquelle angeben, so ist dies z.B. durch den Aufruf der Funktion `SQLDriverConnect` unter Vorgabe eines Treibertyps (z.B. MS Access, vgl. Beispielprogramm T_ODBCDrvConnRd.f90) möglich.

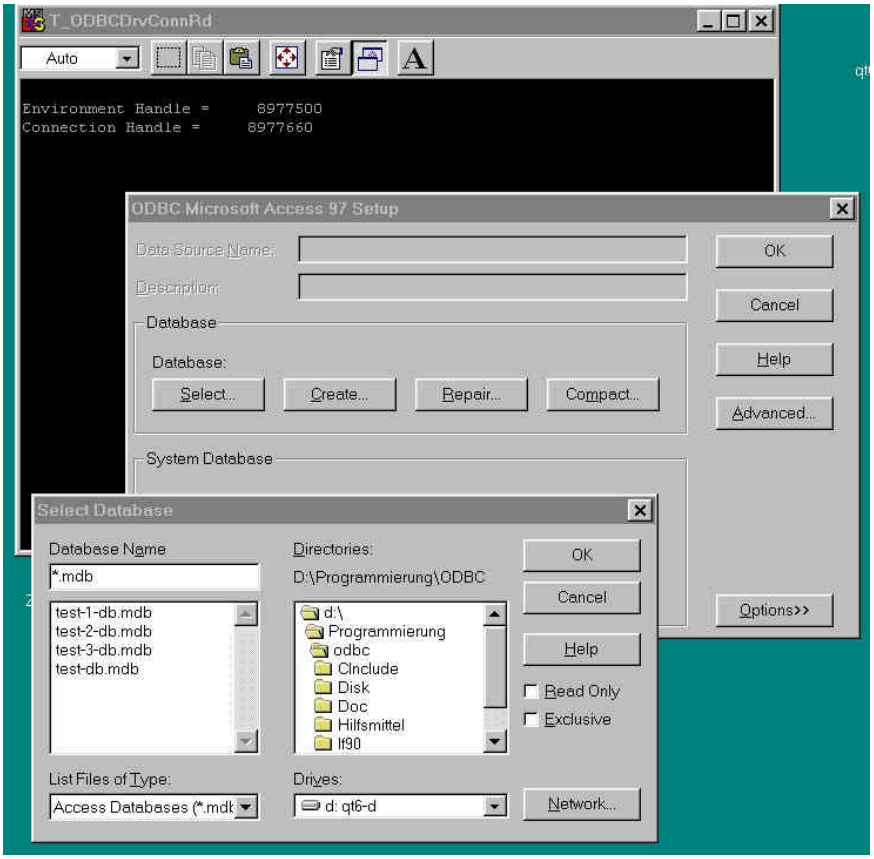

Abb.6: Auswahl der Datenquelle während der Laufzeit - vgl. Beispielprogramm T_ODBCDrvConnRd.f90.

5.5.3 Die Ausführung von SQL Befehlen

Es können alle SQL Befehle ausgeführt werden, die von der Datenquelle unterstützt werden. Die dazu verwendete Syntax sollte den Standarddefinitionen von ODBC genügen (SQL Grammar). Ein SQL Befehl wird vom Treiber in die von der Datenquelle gebräuchliche Syntax umgewandelt. Liegt der Befehl nicht in der Standard ODBC Syntax vor, wird er direkt, d.h. ohne Umwandlung, an die Datenquelle geleitet.

Man unterscheidet die

- einfache, **direkte Ausführung** eines Befehls (direct execution)

oder die

- mehrfache, **vorbereitete Ausführung** (prepared execution).

Direkte Ausführung erfolgt mittels **`SQLExecDirect`**, wenn der Befehl nur einmal ausgeführt werden soll (bspw. SELECT) und keine Information über den Ergebnisinhalt (result set) vor Ausführung bekannt ist.

Die vorbereitete Ausführung mittels **`SQLPrepare`** und folgendem `SQLExecute` wird im gegenteiligen Fall verwendet, also wenn ein Befehl mehrfach ausgeführt werden soll (bspw. INSERT, UPDATE) oder wenn Information über den Ergebnisinhalt (result set) vor Ausführung bekannt ist.

Ein vorbereiteter Befehl wird im allgemeinen schneller ausgeführt als ein direkter, da für jeden SQL Befehl intern ein „Zugriffsplan" erstellt wird, der bei wiederholter Ausführung im vorbereiteten Fall nur einmal zu generieren ist, hingegen beim direkten Fall bei jedem Befehl. Allerdings ist Vorsicht beim Verhalten von `SQLTransact` (d.h. beim COMMIT bzw. ROLLBACK) oder bei Verwendung von SQL_AUTOCOMMIT (bzw. SQL_ATTR_AUTOCOMMIT) geboten, da dabei der Zugriffsplan intern gelöscht werden kann. Näheres hierzu: siehe die Informationstypen via `SQLGetInfo` von SQL_CURSOR_COMMIT_BEHAVIOR und SQL_CURSOR_ROLLBACK_BEHAVIOR.

Vor Ausführung eines SQL Befehls müssen intern Speicherbereiche angelegt werden, die über die Befehlsidentifkationsnummer (statement handle) identifiziert werden. Die Befehlsidentifkationsnummer ist vom Typ HSTMT bzw. SQLHANDLE und kann bspw. wie folgt deklariert werden.

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

Sie wird dann durch den Aufruf von `SQLAllocStmt` erzeugt (ODBC v1.x/v2.x). `SQLAllocStmt` benötigt die zuvor generierte Verbindungsidentifikationsnummer (im Beispiel: dbc).

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

Ab ODBC v3.0 kann man äquivalent codieren:

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

Vor Ausführung eines SQL Befehls können Werte und Befehlsparameter gesetzt werden (siehe dazu Abschnitt „Das Setzen von Parametern").

Schließlich folgt im direkten Fall die Ausführung des Befehls mittels `SQLExecDirect`. Z.B.:

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

Erläuterung: Die zuvor erzeugte Befehlsidentifkationsnummer (stmt) identifiziert den intern angelegten Speicherbereich, in dem der nachfolgende SQL Befehl „DELETE FROM..." als null-terminierte Zeichenkette („//CHAR(0)") abgelegt wird. Die Länge des Befehls wird intern ermittelt, da als Längenparameter SQL_NTSL (null terminated string) verwandt wird. Hier ist die "lange" Variange von SQL_NTS notwendig, da das INTERFACE dies fordert.

Im vorbereiteten Fall wird der SQL Befehl `SQLPrepare` analog verwandt. D.h., die Funktion verwendet die gleichen Parameter wie

`SQLExecDirect`. Anschließend können Werte und Befehlsparameter gesetzt werden (siehe dazu Abschnitt „Das Setzen von Parametern“). Daraufhin besteht die Möglichkeit Information über das Ergebnis (result set) vorab zu erhalten. Der Aufruf von `SQLExecute` schließlich führt den vorbereiteten Befehl aus. Z.B.:

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

Im nachfolgenden Diagramm (mit ODBC v2 Funktionen) ist der einfache Ablauf eines Programms zu sehen, das ODBC API Funktionen aufruft, um SQL Befehle auszuführen.

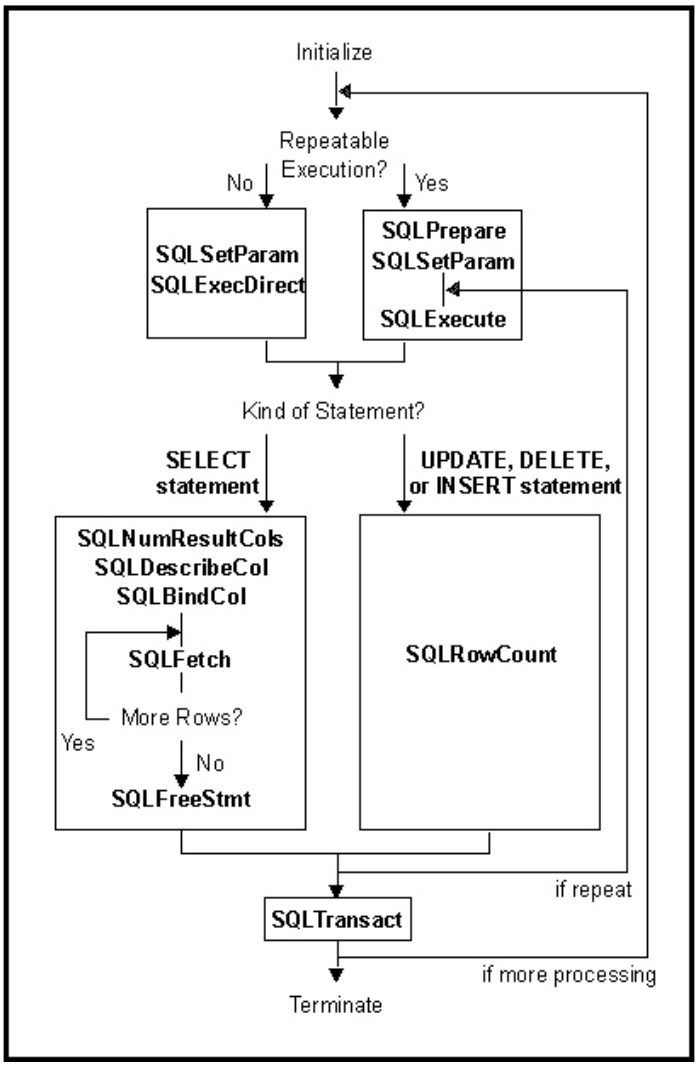

Abb. 7: Programmstruktur zur Ausführung von ODBC Befehlen

Man beachte, daß Befehle entweder nur einmal mit `SQLExecDirect` oder mehrfach nach Vorbereitung durch `SQLPrepare` mit `SQLExecute` ausgeführt werden können. `SQLTransact` wird verwandt um das COMMIT oder ROLLBACK zu veranlassen.

5.5.4 Das Setzen von Parametern ("Parameter Binding")

Ein SQL Befehl kann Platzhalter für Parameterwerte (parameter markers) enthalten. Z.B.:

```
INSERT INTO addressbuch (name, vorname, telefon)
VALUES (?, ?, ?)
```

Der Treiber erkennt an diesen Platzhaltern, daß sie während der Laufzeit durch Werte ersetzt werden. Man verwendet Platzhalter, wenn

- in einem vorbereiteten Befehl (prepared statement), in dem diese Platzhalter bei Ausführung des Befehls durch sich ändernde Werte ersetzt werden sollen (im obigen Beispiel durch Einfügen mehrerer neuer Adressbucheinträge),

oder wenn

- die Parameterwerte noch nicht bekannt sind, wenn der Befehl vorbereitet wird,

oder wenn

- die Typen von Parameterwerten in andere umgewandelt werden müssen.

Bevor ein Parameterwert gesetzt werden kann, muß ihm eine Variable, bzw. genauer ein Speicherplatz für den Platzhalter mithilfe der Funktion `SQLBindParameter` zugeordnet werden ("Parameter Binding"). `SQLBindParameter` gibt zudem den Datentyp des Speicherplatzes, die Genauigkeit, Länge und ggf. Dezimalbereich an und assoziiert die Spalte mit dem Parameter. Danach kann durch Zuweisung des gewünschten Wertes an diesen Speicherplatz der Parameter gesetzt werden. Z.B.:

```
CHARACTER szName*30
INTEGER (KIND=SDWORD) :: LENszName = 30
INTEGER (KIND=SDWORD) :: cbName
.
rtc = SQLPrepare(stmt,    &
             "INSERT INTO addressbuch (name, vorname,
               telefon) VALUES (?, ?, ?)"//CHAR(0),   &
             SQL_NTS )
rtc = SQLBindParameter( stmt, 1, SQL_PARAM_INPUT,  &
                        SQL_C_CHAR, SQL_CHAR,      &
                        LENszName, 0, szName, 0,   &
                        cbName )
```

Erläuterung: An die erste Spalte (zweites Argument = 1) vom Typ (SQL_CHAR) der Tabelle "addressbuch" wird der Speicherplatz der Eingabevariablen szName (SQL_PARAM_INPUT) vom Typ SQL_C_CHAR mit Länge LENszName gebunden. In der Variablen cbName wird die Länge des zurückgegebenen Werts abgelegt (nachdem der Befehl `SQLExecute` ausgeführt wurde). Die Angabe einer maximale Längenbegrenzung für den Rückgabewert (vorletztes Argument) ist hier nicht notwendig (und zu 0 gesetzt).

Der Datentyp des Speicherplatzes (i.e.v. der Variable) muß nicht mit dem Typ der Spalte in der Tabelle der Datenquelle übereinstimmen. Man kann bspw. die Konvertierungsfunktionen des Treibers nutzen, um bspw. einen als Ganzzahl (SQL_INTEGER) in der Tabelle abgelegten Wert in einen Zeichentyp (SQL_C_CHAR) umgewandelt an die Applikation zurückzugeben.

Die Zuordnungen der Speicherplätze (d.h. C/Fortran Variable zu ODBC/SQL Input Parameter) bleiben solange erhalten, bis die Funktion `SQLFreeStmt` unter Verwendung der Optionen SQL_RESET_PARAMS oder SQL_DROP aufgerufen wird (oder ab ODBC v3.: `SQLFreeHandle`). Während der Laufzeit kann an den Platzhalter ein anderer Speicherplatz gebunden werden, indem die Funktion `SQLBindParameter` ein weiteres Mal aufgerufen wird. Auch der Wert des Speicherplatzes kann während der Laufzeit beliebig

geändert werden. Genutzt wird vom Treiber bei Ausführung eines Befehls der jeweils aktuelle Wert.

5.5.5 Transaktionen

ODBC bzw. SQL kennt zwei COMMIT Modi:

- Im **Auto-Commit-Modus** (SQL_AUTO_COMMIT) wird nach jedem SQL Befehl eine Transaktion (COMMIT, ROLLBACK) durchgeführt.
- Im **Manual-Commit-Modus** ist es Sache des Programmierers, die Transaktion mittels `SQLTransact` (bzw. `SQLEndTran`) selbst abzuschließen. Sie kann einen oder mehrere SQL Befehle umfassen.

Wenn ein Treiber SQL_AUTO_COMMIT (bzw. SQL_ATTR_AUTOCOMMIT) unterstützt, ist dies auch der voreingestellte Transaktions-Modus. Ansonsten ist es der manuelle Commit-Modus. Mittels der Funktion `SQLSetConnectOption` (bzw. `SQLSetConnectAttr`) kann zwischen beiden Modi gewechselt werden.

Die COMMIT bzw. ROLLBACK Befehle können auch als SQL Befehle via `SQLExecDirect` ausgeführt werden. Empfohlen wird jedoch `SQLTransact` (bzw. `SQLEndTran`) zu verwenden.

Wichtig zu wissen ist, das nach einer Transaktion SQL Cursor und auch die internen Zugriffspläne verloren gehen können. Näheres hierzu: siehe die Informationstypen via `SQLGetInfo` von SQL_CURSOR_COMMIT_BEHAVIOR und SQL_CURSOR_ROLLBACK_BEHAVIOR.

5.5.6 Der Empfang von Ergebnissen (Retrieving Result Sets)

Die SQL Befehle lassen sich in solche unterteilen, die

- Ergebnissätze (result sets) produzieren und diese zurückgeben (z.B. die SELECT Befehle)

und solche, die

- keine erzeugen, sondern nur Manipulationen an der Datenquelle vornehmen (z.B. DELETE, UPDATE, INSERT, GRANT oder REVOKE).

Ob im letzteren Fall die Funktion fehlerfrei ausgeführt wurde, läßt sich über den zurückgegebenen Funktionswert oder auch ggf. über die Änderung der Satzanzahl ermitteln (`SQLRowCount`).

Im ersteren Fall muß die Applikation wissen, im welchen Format das Ergebnis zurückgegeben wird (z.B.: ein "SELECT * FROM addressbuch" liefert als Ergebnis alle Sätze und alle Spalten der Tabelle, die der Applikation mitunter nicht bekannt sind).

Wenn Ergebnissätze durch den SQL Befehl erzeugt werden, so ist dafür zu sorgen, daß die Ergebnisse so abgelegt werden können, daß die Applikation darauf Zugriff hat. Die Funktionen `SQLBindCol` (ODBC v1.0) und `SQLBindParameter` (ODBC v2.0) dienen diesem Zweck. Sie binden eine Spalte (column) einer Tabelle, die in einem SQL Befehl genannt wird, an einer Speicherort (i.A. eine Variable Ihres Programms). Dieses "Column Binding" kann vor oder nach der Vorbereitung oder Ausführung eines SQL Befehls erfolgen (danach bspw. dann, wenn der Umfang des Ergebnisses,

d.h. die Spaltenanzahl, vor Ausführung des Befehls unbekannt ist). SQLBindCol und SQLBindParameter erfordern die Angabe

- des Datentyps (ein C konformer Datentyp), in welchen das Ergebnis umzuwandeln ist
- eines hinreichend großen Ausgabepuffers (dies ist i.A. eine in der Applikation definierte Variable)
- die Länge des Ausgabepuffers, sofern der Datentyp keine voreingestellte feste Länge in C hat (bspw. INTEGER, REAL haben eine feste Länge)
- einer Variable, in der die Länge des Ergebnisses (in Byte) bei der Rückgabe zu finden ist.

Beispiel:

```
CHARACTER wName*21
INTEGER (SQLINTEGER) :: LENwName = 21
INTEGER (SQLINTEGER) :: cbwName
.
rtc = SQLExecDirect( stmt,    &
     "SELECT waehrungname FROM waehrungen"//CHAR(0), &
     SQL_NTS )
rtc = SQLBindCol( stmt, 1, SQL_C_CHAR, wName,  &
                  LENwName, cbwName)
```

Erläuterung: Der erste Spaltenname des SELECT Befehls wird an den Speicherplatz der Variable wName vom Typ SQL_C_CHAR der Länge LENwName gebunden (zweites Argument von SQLBindCol ist =1). Wenn der SELECT Befehl erfolgreich ausgeführt wird, wird *später* das Ergebnis in wName und seine Länge in cbwName abgelegt.
Ab ODBC 2.0 kann alternativ auch die Funktion SQLBindParameter verwendet werden:

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

Erhält der gebundene Spaltenwert den Wert NULL (d.h. ist nicht definiert), wird in der Längenangabe zum Puffer der Wert SQL_NULL_DATA zurückgegeben ("missing value").

Sind die Charakteristika des Ergebnisses eines SQL Befehls innerhalb der Applikation nicht bekannt, so erlauben die Funktionen

- SQLNumResultCols die Anzahl

und

- SQLColAttributes (ODBC v1.x/2.x) bzw. SQLColAttribute (ODBC v3.x) und SQLDescribeCol eine Beschreibung

der Spalten zu liefern. Diese Routinen können nachdem zuvor ein SQL Befehl vorbereitet oder ausgeführt wurde, aufgerufen werden.

Sobald die Bindung zwischen Ergebnisspalten und Applikationsvariablen bzw. Speicherplätzen erfolgt ist (siehe oben SQLBindCol bzw. SQLBindParameter), ermöglicht die Funktion SQLFetch durch die Sätze (rows) des Ergebnisses (result set) zu schreiten und sie einzeln "abzuholen".

Untenstehendes Diagramm zeigt den Ablauf des "Abholens" der Ergebnisse.

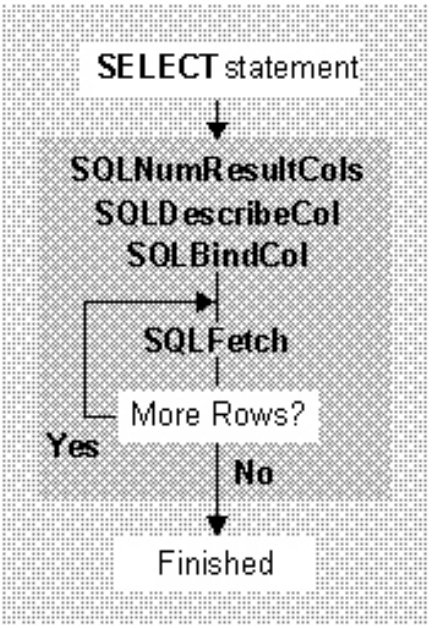

Abb. 8: Abholen von Ergebnissen

Beispiel:

```
rtc = SQLExecDirect( stmt,   &
     "SELECT waehrungname FROM waehrungen"//CHAR(0),  &
     SQL_NTS )
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
```

Erläuterung: Der SELECT Befehl soll alle Währungsnamen (in der ersten und hier einzigen Spalte „waehrungname“) der Tabelle „waehrungen“ abholen. Nach Ausführung des Befehls, erfolgt die Bindung der Variablen „wName“ an die Ergebnisspalte. In der DO WHILE Schleife wird die `SQLFetch` Funktion aufgerufen. Nach jeder Ausführung, sind die Werte für den Währungsnamen und seine Länge in den Variablen „wName“ und „cbwName“ zu finden.

Wenn eine Spalte den Wert NULL enthält (der anzeigen soll, daß sie noch nicht besetzt wurde), so wird kein Wert in die zugeordnete Variable „beim Fetch“ übertragen (d.h. die Variable ändert ihren Wert nicht). Stattdessen enthält die Längenangabe (im Beispiel cbwName den Wert SQL_NULL_DATA.

Innerhalb des Treibers wird zur Verfolgung des aktuellen Satzes, der durch ein `SQLFetch` angesprochen wird, ein Cursor verwaltet. Der Cursor wird nach jedem `SQLFetch` zum nächsten Satz vorwärtsbewegt. Eine Rückwärtsbewegung ist nicht möglich. Der Cursor kann nach einem COMMIT oder ROLLBACK geschlossen und gelöscht werden. Näheres hierzu: siehe die Informationstypen via `SQLGetInfo` von SQL_CURSOR_COMMIT_BEHAVIOR und SQL_CURSOR_ROLLBACK_BEHAVIOR.

ODBC bietet weitere, über die Spezifikationen der X/Open und SQL Access Gruppe hinausgehende Funktionen, um Ergebnisse aus Datenquellen zu erhalten. Man findet dazu in [ODBC-R] detaillierte Informationen.

Wichtiger Hinweis: Manche Compiler verändern beim „Optimieren“ das Ablaufverhalten insbesondere der „Fetch Loop“ (vgl. obiges Beispiel mit der DO WHILE Schleife), die zum Abholen der Ergebnisse dient. Da durch das „Column Binding“ die Werte der Variablen nicht durch das Fortran Programm selbst, sondern durch den ODBC Treiber geändert werden und dies der Compiler nicht „weiß“, muß im Optimierungsmodus damit gerechnet werden, daß er Teile einer „Fetch Loop“ entfernt, von denen er „meint“, daß sie nicht während des Schleifendurchlaufs verändert werden (so dass sie durch die Optimierung „vor die Schleife“ versetzt werden). Mitunter muß man dann also die Optimierung für das Compilieren einer Routine mit einer „Fetch Loop“ ausschalten.

5.5.7 Information über Status und Fehler

ODBC definiert Rückgabewerte (return codes) und ein Protokoll zur Fehlerbehandlung (error handling protocol). Letzteres gibt vor, wie die Komponenten (z.B. Treiber, Treiber Manager) einer ODBC Verbindung Fehlermeldungen erstellen und mittels der Funktion `SQLError` (ODBC v1.x/2.x) bzw. `SQLGetDiagRec` (ODBC v3.x) wiedergeben. Das Fehlerprotokoll enthält

- den SQL Status (SQLState)
- einen treiberspezifischen Fehlercode (native error)
- eine Fehlermeldung

Ein Rückgabewert zeigt an, ob eine ODBC Funktion erfolgreich ausgeführt wurde, bedingt erfolgreich war und eine Warnung zu beachten ist oder versagt hat. Die Rückgabewerte im Einzelnen:

- SQL_SUCCESS: Die Funktion wurde erfolgreich und vollständig ausgeführt. Weitere Information ist nicht verfügbar.
- SQL_SUCCESS_WITH_INFO: Die Funktion wurde erfolgreich ausgeführt, allerdings mit einem nicht schwerwiegenden Fehler. Weitere Information kann mittels der Funktion `SQLError` bzw. `SQLGetDiagRec` abgefragt werden.
- SQL_NO_DATA_FOUND: Alle Datensätze eines Ergebnisse sind abgeholt worden oder es sind keine vorhanden.
- SQL_ERROR: Die Funktion versagte. Mittels `SQLError` bzw. `SQLGetDiagRec` können weiterführende Informationen abgefragt werden.
- SQL_INVALID_HANDLE: Es wurde eine ungültige Identifikationsnummer für die Umgebung, für die Verbindung oder für den Befehl (environment, connection, statement handle) verwendet. Ursache ist meist ein Programmierfehler. `SQLError` bzw. `SQLGetDiagRec` liefern keine weitere Information.
- SQL_STILL_EXECUTING: Eine Funktion läuft asynchron zur Applikation und ist noch nicht fertig bzw. abgeschlossen.
- SQL_NEED_DATA: Während ein Befehl ausgeführt wird, stellte der Treiber fest, daß die Applikation Parameterdaten senden muß.

In Abhängigkeit vom Rückgabewert ist es Aufgabe der Applikation entsprechend zu reagieren und die Fehlersituation zu bewältigen. Mitunter ist es dabei notwendig, daß im Fehlerfall die Funktion `SQLError` bzw. `SQLGetDiagRec` mehrfach aufgerufen werden müssen, um alle Fehlermeldungen abzuholen. Wenn vorzeitig eine weitere ODBC Funktion gerufen wird (mit Ausnahme von `SQLError` bzw. `SQLGetDiagRec`

selbst), geht die evt. noch vorhandene Fehlerinformation zu der vorherigen Funktion verloren.

Die Fehlerinformationen, die von `SQLError` bzw. `SQLGetDiagRec` zurückgegeben werden, erfolgen im gleichen Format, das SQLSTATE im X/Open und SQL Access Group SQL CAE Spezifikation (1992) zugrunde liegt.

Weitere Informationen und Spezifikationen bzgl. ODBC Fehlermeldung sind in [ODBC-E] zu finden.

5.5.8 Abbruch asynchroner Funktionen

Asynchron ablaufende Funktionen/Prozesse können mithilfe von `SQLCancel` abgebrochen werden. Wann der Abbruch erfolgt, hängt jedoch von der Datenquelle ab. Anschließend kann die gleiche asynchrone Funktion nochmals aufgerufen werden. Gibt sie statt SQL_STILL_EXECUTING zurück, war der Abbruch noch nicht erfolgreich. War der Abbruch erfolgreich, geben sie SQL_ERROR und SQLSTATE S1008 (= operation cancelled) zurück. Wenn die Funktion vollständig ausgeführt wurde, verhält sie sich normal und gibt die üblichen Fehlerwerte zurück, wenn ein Fehler auftrat. Die Funktionen `SQLError` bzw. `SQLGetDiagRec` geben ebenfalls S1008 zurück, wenn der Abbruch erfolgreich war.

5.5.9 Beenden von Verbindungen

Zur Freigabe von Ressourcen (z.B. Speicher), die in einer ODBC Applikation angelegt wurden, dienen die Funktionen `SQLFreeStmt`, `SQLFreeConnect` und `SQLFreeEnv` oder `SQLFreeHandle` (ab ODBC v3.0).

- `SQLFreeStmt` gibt die Resourcen einer Befehlsidentifikationsnummer (statement handle) frei. Die Funktion hat vier Optionen:
  - SQL_CLOSE: Schließt einen Cursor - vorausgesetzt, daß einer existierte und verwirft noch ausstehende Ergebnisse. Die freigegebene Befehlsidentifikationsnummer (bzw. deren Variable) kann später wieder verwendet werden.
  - SQL_DROP: Umfaßt die Funktionalität von SQL_CLOSE und gibt überdies alle Ressourcen frei, die mit der Befehlsidentifikationsnummer verknüpft sind.
  - SQL_UNBIND: Gibt alle Ausgabepuffer frei, die über `SQLBindCol` für die betroffene Befehlsidentifikationsnummer angelegt wurden.
  - SQL_RESET_PARAMS: Gibt alle Parameter Puffer frei, die über `SQLBindParameter` für die betroffene Befehlsidentifikationsnummer angelegt wurden.

Nach Freigabe der Befehlsidentifikationsnummer(n) kann eine Verbindung mithilfe der Funktion

- `SQLDisconnect`, die die Verbindung beendet, abgebaut werden.

Anschließend ist die Funktion

- `SQLFreeConnect` aufzurufen, die die Ressourcen der Verbindung und die Verbindungsidentifikationsnummer (connection handle) frei gibt.

Zuletzt bleibt noch mittels

- `SQLFreeEnv` die Umgebungsidentifikationsnummer und ihre Ressourcen freizugeben.

5.6 Besonderheiten hinsichtlich des Zugriffs auf Excel-Datenquellen

Spezielle Hinweise sind beim Gebrauch von Excel Arbeitsblättern ("Worksheets") zu beachten:

- Die Spaltennamen ergeben sich i.a. aus der ersten Zeile des Arbeitsblattes.
- Zeilen (rows) können nicht gelöscht werden
- Individuelle Zelleninhalte können gelöscht werden, mit Ausnahme von Zellen, die Formeln enthalten. Letztere können auch nicht modifiziert werden.
- Indizierungen können nicht vorgenommen werden.
- Mehrfachzugriff (durch mehrere Benutzer) ist nicht möglich (die Database Engine unterstützt keinen Mehrbenutzerbetrieb).
- Daten, die innerhalb von Excel verschlüsselt wurden, können nicht gelesen werden.

Anmerkung: Die Dokumentation des Zugriffs auf EXCEL-Tabellen in der ODBC API ist leider mehr als dürftig. Vgl. das Beispielprogramm T_ODBCExcel.f90.

6. Installation und Inbetriebnahme von ForDBC

ForDBC wird auf einer CD-ROM oder in gepackter Form (ZIP) via e-Mail geliefert oder steht zum Download zur Verfügung. Im Stammverzeichnis der CD-ROM befindet sich ein Installationprogramm mit dem sich die Installation zum Teil automatisiert durchführen läßt. Selbst wenn Sie dieses verwenden, sollten Sie die Installation anhand der Anleitung zur manuellen Installation durchlesen, einerseits um zu erfahren, was installiert wurde, und andererseits, um Ihre Installation zu überprüfen.
Falls Sie ForDBC in gepackter Form (ZIP) erhalten haben, genügt das Entpacken in ein Verzeichnis Ihrer Wahl.

6.1 Automatisierte Installation

Zur automatisierten Installation von ForDBC dient ein Stapelprogramm namens

```
install.bat
```

das Sie bitte von der DOS-Eingabeaufforderung aus aufrufen:

`install [Q] [Z] [C]` <Enter/Return>-Taste drücken

wobei die Parameter bzw. Argumente folgende Bedeutung haben:

[Q]: Laufwerksbuchstabe Ihres CD-ROM-Laufwerks, z.B. D

[Z]: Namen des Zielverzeichnisses ohne Laufwerksangabe, z.B. ForDBC

[C]: Compilerkürzel, entweder DVF, FTN95, IVF, LF90 oder LF95

Beispielaufruf:

```
install  D  FORDBC  FTN95
```

Die Installation geht davon aus, daß das gegenwärtige Festplattenlaufwerk (z.B. C:) dasjenige ist, auf dem Sie ForDBC installieren.

Wenn die automatisierten Installation fehlerfrei verlaufen ist, nehmen Sie bitte die Installation der Test-Datenquellen vor (siehe übernächster Abschnitt).

6.2 Manuelle Installation

Wenn sie ForDBC manuell installieren oder die Installation überprüfen möchten, bspw. weil bei der Compilation Fehler auftraten, beachten Sie bitte folgendes:

ForDBC besteht aus den Fortran 90 Modulen
qt_Ckinds.f90
qt_Win32Kinds.f90
qt_Win32Types.f90
qt_Win32Constants.f90
qt_ODBCKinds.f90
qt_ODBCDefs.f90
qt_ODBC.f90
qt_ODBCInterfaces.f90
die im Hauptverzeichnis der Installations CD-ROM zu finden sind.

Im compiler-spezifischen Unterverzeichnis befindet sich zudem ein Modul namens

qt_ODBC_*compiler*.f90 mit *compiler* = DVF, FTN, IVF, LF90, oder LF95

Alle diese Module sind auf Ihre Festplatte zu kopieren -

- legen Sie hierzu ein eigenes Verzeichnis an, bspw. mit Namen ForDBC
- und kopieren alle o.g. Module in dieses Verzeichnis.

Diese Module sind anschließend in der genannten Reihenfolge zu compilieren. Dazu steht für alle Compiler mit Ausnahme von IVF, eine Stapeldatei namens

compile_Modules.bat

zur Verfügung, die Sie ggf. Ihren spezifischen Installationsgegebenheiten anpassen müssen.

Für IVF (Intel Visual Fortran) laden Sie bitte in die Entwicklungsumgebung die Datei IVF.sln, die im Unterverzeichnis IVF zu finden ist. Rufen Sie dann mittels Menü "Build | Batch Build" den "Batch Build" Dialog auf und drücken "Build", um alle Projekte der "Solution" zu erstellen.

Die nach der erfolgreichen Compilation entstandenen objekt- und compilerspezifischen Moduldateien müssen in einem Verzeichnis sein, auf das Ihr Compiler beim Übersetzen einer ODBC Applikation, die eines der o.g. Module verwendet, zugreifen kann (die meisten Compiler verwenden ein voreingestelltes Verzeichnis hierfür oder erlauben die Angabe eine Modulpfads).

Des weiteren sind Testprogramme vorhanden, die o.g. Module verwenden:

T_ODBCDataSources.f90	listet Datenquellen
T_ODBCDrivers.f90	listet Treiber
T_ODBCDrvConnRd.f90	liest Access- oder Excel-Dateien (test-db.mdb bzw. TestODBCDrvConnRd.xls, die während der Laufzeit auszuwählen sind)
T_ODBCTestAccessInfo.f90	informiert über Datenquelle test-db.mdb
T_ODBCTestAccessInfo.f90	informiert über Datenquelle test-db.mdb
T_ODBCTestAccessRd.f90	liest Datenquelle test-db.mdb
T_ODBCTestAccessWr.f90	schreibt in Datenquelle test-db.mdb
T_ODBCTestExcelRd.f90	liest Datenquelle ODBCTestExcel.xls (zeigt auch Tabelleninformation, wie Spaltennamen etc. an)
T_ODBCTestExcelWr.f90	schreibt in Datei TestODBCExcelWr.xls (die Datei ist während der Laufzeit auszuwählen)

Diese sind ebenfalls in das zuvor angelegte Verzeichnis (bspw. ForDBC) zu kopieren und zu compilieren und mit der **ODBC32.DLL bzw. in Abhängigkeit vom verwendeten Fortran Compiler/Linker mit der entsprechenden Import-Library ODBC32.LIB und ggf. einer zusätzlichen Schnittstellenbibliothek zu binden**. Dazu steht für alle Compiler mit Ausnahme von IVF eine Stapeldatei namens

compile&link_Testprograms.bat

zur Verfügung, die Sie ggf. Ihren spezifischen Installationsgegebenheiten anpassen müssen.
Wenn Sie die Testprogramme aus einer Entwicklungsumgebung heraus compilieren und binden wollen, finden Sie zu Beginn einer jeden Datei Anweisungen, welche Einstellungen für die jeweiligen Compiler vorzunehmen sind.

Auf dem Installationsmedium ist noch eine Datei

Addendum.txt

zu finden, die ggf. weitere Informationen enthält.

6.3 Anlegen der Test-Datenquellen

Die nach der erfolgreichen Compilation erzeugten ODBC Applikationen (.EXE) sind allerdings erst lauffähig, wenn zuvor die in diesen Testprogrammen verwendeten Datenquellen

ODBCTest.xls	[Excel 95 / 7.0 Worksheet]
test-db.mdb	[MS Access Database]

angelegt wurden. Hierzu ist der ODBC Administrator aufzurufen (siehe Abschnitt "Definition und Konfiguration von Datenquellen"), und es sind die Datenquellen mit Namen

ODBCTestExcel	[für die Datei ODBCTest.xls mit Excel 5.0/7.0 Treiber]
ODBCTestAccess	[für die Datei test-db.mdb mit MS Access Treiber]

zu definieren. Erst dann können die meisten der o.g. Testprogramme ausgeführt werden.

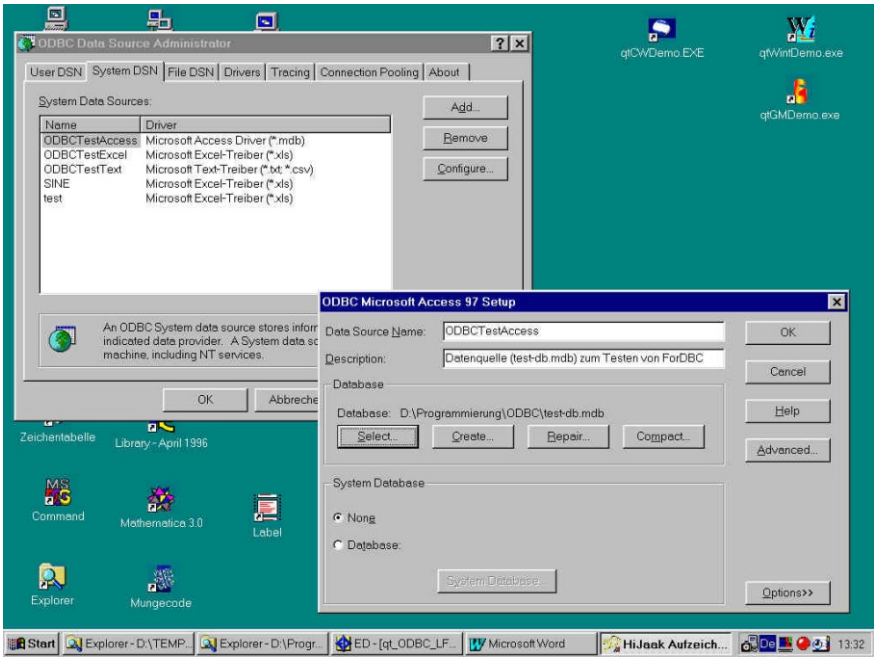

Abb. 9: Anlegen einer Datenquelle mit dem ODBC Administrator.

Die anderen beiden Excel-Dateien

TestODBCDrvConnRd.xls

und

TestODBCExcelWr.xls

brauchen nicht als Datenquellen angelegt zu werden.

6.4 Compilerspezifische Anmerkungen

6.4.1 Compaq bzw. Digital und Intel Visual Fortran

Wenn Sie ForDBC in der Entwicklungsumgebung verwenden wollen, müssen die Module im Modulpfad zu finden sein. Ggf. geben Sie diesen in den Einstellungen (Settings) des Projekts an (Project Settings, für „All Configurations“ | Registerkarte Fortran, Category „Preprocessor“, Eingabefeld Module Path: <Modulpfad angeben>).

Beim Binden (Link) ist darauf zu achten, daß die ODBC32.LIB dazugebunden wird (Project Settings, für „All Configurations“ | Registerkarte Link, Eingabefeld Object/library modules: kernel32.lib odbc32.lib) oder als Datei dem Projekt hinzufügen.

6.4.2 Lahey LF90

Es wurde versucht eine direkte Verwendung der ODBCAPI mit dem LF90 v4.5 zu programmieren, was aber an der fehlenden Import-Library scheiterte. Der Compiler-Hersteller liefert selbst keine Import-Library (die dem LF90 beigefügten Import-Libraries decken nur die Win32API ab, aber nicht die ODBC API). Auch der Versuch die ODBC32.LIB direkt

einzubinden scheiterte: Obwohl ein Testprogramm, das den Aufruf der ODBC Funktion `SQLAllocEnv` enthielt, anscheinend korrekt übersetzt und fehlerfrei gebunden wurde, trat beim Start der Fehler „fehlender Export _SQLAllocEnv@4“ auf, obwohl die ODBC32.DLL vorhanden war. Um LF90 Benutzern die ODBCAPI trotzdem zugänglich zu machen, wurde eine Schnittstellenbibliothek geschaffen (qtODBCLF90.LIB und zugehörige qtODBCLF90.DLL), die nicht die gesamte, aber wesentliche ODBCAPI Funktionen enthält (siehe ForDBC Funktionsübersicht). Der Quellcode ist in der Datei qtODBCLF90.f90 zu finden und kann leicht erweitert werden, wenn man sich an das dort anzutreffende Programmierschema hält.

6.4.3 Lahey LF95

Zur Generierung der Programme wird neben den zuvor genannten Modulen und der ODBC Library (ODBC32.LIB bzw. ODBC32.DLL) noch die Schnittstellenbibliothek

qtODBCLF95.LIB

benötigt. Damit wird ein Programm beispielsweise wie folgt übersetzt:

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

6.4.4 Salford FTN95

Bedingt durch den „typenlosen“ Import der ODBC API Funktionen (vgl. qt_ODBC_FTN.f90) kann es gelegentlich zu Problemen bei der Übergabe von Parametern bzw. Argumenten, insbesondere bei 2-Byte-INTEGER Werten kommen. In vielen Fällen sind diese durch Änderung der Definition in 4-Byte-INTEGER zu lösen, wobei man jedoch darauf achten sollte, daß diese „vergrößerten“ Variablen vor Gebrauch korrekt initialisiert werden (bspw. durch „zu 0 setzen“).

Hinsichtlich des Auffindens der Module, sei daran erinnert, daß dem Compiler mittels der Option /mod_path dieser Pfad mitgeteilt werden kann.

Beim Binden (mit SLINK) genügt die ODBC32.DLL mit anzugeben (oder auch die Importlibrary ODBC32.LIB).

7. ForDBC Funktionsübersicht

Nachfolgende Tabelle führt die in ForDBC implementierten Funktionen und den ODBC Implementierungs Level auf (vgl. Kapitel "ODBC Konformitätsebenen (ODBC Levels)"). Eine Auflistung der ForDBC INTERFACEs, d.h. der vollständigen Funktionsdeklarationen, finden sich im Anhang A.

Funktionsname	Kurzbeschreibung	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. Quellen

Referenzen zu [ODBC..] beziehen sich auf:

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

[ODBC98] Microsoft Developer Network, Library Visual Studio 6.0, deutsche Ausgabe, 1998: Plattform-SDK\Datenbank- und Messaging-Dienste\Microsoft Datenzugriff 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

Anhang A - ForDBC Funktionsübersicht

Datei qt_ODBCInterfaces.f90

```
00001 ! ========================================
00002 ! qt_ODBCInterfaces for LF95, FTN95, DVF...
00003 ! ----------------------
00004 ! (C) Jörg Kuthe, QT software, 1999-2007.
00005 ! ----------------------
00006 ! Kontakt: email: jk@qtsoftware.de   http://www.qtsoftware.de
00007
00008 ! DVF/CVF
00009 ! ---
00010 ! compile: DF qt_ODBCInterfaces.F90 -c -win -compile_only -nologo -libs:dll /warn:nofileopt -dll
00011
00012 ! LF95
00013 ! --
00014 ! compile: LF95 qt_ODBCInterfaces.f90 -nwrap -c -win -mod d:.mod&obj -ml msvc
00015 !          mit "d:.mod&obj" als dem Modulpfad
00016
00017 MODULE qt_ODBCInterfaces
00018    USE qt_ODBCKinds
00019
00020    INTERFACE SQLAllocConnect
00021       FUNCTION SQLAllocConnect(env, dbc)
00022          USE qt_ODBCKinds
00023          INTEGER (SQLRETURN) :: SQLAllocConnect
00025          INTEGER (SQLHENV) :: env
00026          INTEGER (SQLHDBC) :: dbc
00028       END FUNCTION SQLAllocConnect
00029    END INTERFACE
00030
00031    INTERFACE SQLAllocEnv
00032       FUNCTION SQLAllocEnv( env )
00033          USE qt_ODBCKinds
00034          INTEGER (SQLRETURN) :: SQLAllocEnv
00035          INTEGER (SQLHENV) :: env
00038       END FUNCTION SQLAllocEnv
00039    END INTERFACE
00040
00041    INTERFACE SQLAllocHandle
00042       FUNCTION SQLAllocHandle( HandleType, InputHandle, OutputHandlePtr )
00043          USE qt_ODBCKinds
00044          INTEGER (SQLRETURN) :: SQLAllocHandle
00045          INTEGER (SQLSMALLINT) :: HandleType
00046          INTEGER (SQLHANDLE) :: InputHandle
00047          INTEGER (SQLHANDLE) :: OutputHandlePtr
00050       END FUNCTION SQLAllocHandle
00051    END INTERFACE
00052
00053    INTERFACE SQLAllocStmt
00054       FUNCTION SQLAllocStmt( dbc, phstmt )
00055          USE qt_ODBCKinds
00056          INTEGER (SQLRETURN) :: SQLAllocStmt
00057          INTEGER (SQLHDBC) :: dbc
00058          INTEGER (SQLHSTMT) :: phstmt
00061       END FUNCTION SQLAllocStmt
00062    END INTERFACE
00063
00064    INTERFACE SQLBindCol
00065
00066       FUNCTION SQLBindColChar( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
00067       ! bind CHAR column
00068          USE qt_ODBCKinds
00069          INTEGER (SQLRETURN) :: SQLBindColChar
00070          INTEGER (SQLHSTMT) :: stmt
00071          INTEGER (SQLUSMALLINT) :: icol
00072          INTEGER (SQLSMALLINT) :: fCType
00073          CHARACTER*(*) rgbValue
00074          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00077       END FUNCTION SQLBindColChar
00078
00079       FUNCTION SQLBindColI1( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
00080       ! bind INTEGER*1 column
00081          USE qt_ODBCKinds
00082          INTEGER (SQLRETURN) :: SQLBindColI1
00083          INTEGER (SQLHSTMT) :: stmt
00084          INTEGER (SQLUSMALLINT) :: icol
00085          INTEGER (SQLSMALLINT) :: fCType
00086          INTEGER*1 rgbValue
00087          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00090       END FUNCTION SQLBindColI1
```

```
00091
00092       FUNCTION SQLBindColI2( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
00093       ! bind INTEGER*2 column
00094          USE qt_ODBCKinds
00095          INTEGER (SQLRETURN) :: SQLBindColI2
00096          INTEGER (SQLHSTMT) :: stmt
00097          INTEGER (SQLUSMALLINT) :: icol
00098          INTEGER (SQLSMALLINT) :: fCType
00099          INTEGER*2 rgbValue
00100          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00103       END FUNCTION SQLBindColI2
00104
00105       FUNCTION SQLBindColI4( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
00106       ! bind INTEGER*4 column
00107          USE qt_ODBCKinds
00108          INTEGER (SQLRETURN) :: SQLBindColI4
00109          INTEGER (SQLHSTMT) :: stmt
00110          INTEGER (SQLUSMALLINT) :: icol
00111          INTEGER (SQLSMALLINT) :: fCType
00112          INTEGER*4 rgbValue
00113          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00116       END FUNCTION SQLBindColI4
00117
00118       FUNCTION SQLBindColR4( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
00119       ! bind REAL*4 column
00120          USE qt_ODBCKinds
00121          INTEGER (SQLRETURN) :: SQLBindColR4
00122          INTEGER (SQLHSTMT) :: stmt
00123          INTEGER (SQLUSMALLINT) :: icol
00124          INTEGER (SQLSMALLINT) :: fCType
00125          REAL*4 rgbValue
00126          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00129       END FUNCTION SQLBindColR4
00130
00131       FUNCTION SQLBindColDP( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )
00132       ! bind DOUBLE PRECISION column
00133          USE qt_ODBCKinds
00134          INTEGER (SQLRETURN) :: SQLBindColDP
00135          INTEGER (SQLHSTMT) :: stmt
00136          INTEGER (SQLUSMALLINT) :: icol
00137          INTEGER (SQLSMALLINT) :: fCType
00138          DOUBLE PRECISION rgbValue
00139          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00142       END FUNCTION SQLBindColDP
00143
00144    END INTERFACE
00145
00146    INTERFACE
00147       FUNCTION SQLBindColLP( stmt, icol, fCType, rgbValue, cbValueMax, pcbValue )   ! added 15.10.2000
00148       ! bind column via pointer (use LOC() function to bind variable)
00149          USE qt_ODBCKinds
00150          INTEGER (SQLRETURN) :: SQLBindColLP
00151          INTEGER (SQLHSTMT) :: stmt
00152          INTEGER (SQLUSMALLINT) :: icol
00153          INTEGER (SQLSMALLINT) :: fCType
00154          INTEGER (LP) :: rgbValue
00155          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00158       END FUNCTION SQLBindColLP
00159    END INTERFACE
00160
00161    INTERFACE SQLBindParameter
00162
00163       FUNCTION SQLBindParameterChar( stmt, ipar,    &
00164                                      fParamType, fCType, fSqlType, cbColDef,            &
00165                                      ibScale, rgbValue, cbValueMax, pcbValue )
00166       ! rgbValue is a CHARACTER buffer
00167          USE qt_ODBCKinds
00168          INTEGER (SQLRETURN) :: SQLBindParameterChar
00169          INTEGER (SQLHSTMT) :: stmt
00170          INTEGER (SQLUSMALLINT) :: ipar
00171          CHARACTER (LEN=*) :: rgbValue
00172          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00173          INTEGER (SQLUINTEGER) :: cbColDef
00174          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00177       END FUNCTION SQLBindParameterChar
00178
00179       FUNCTION SQLBindParameterI1( stmt, ipar,    &
00180                                    fParamType, fCType, fSqlType, cbColDef,           &
00181                                    ibScale, rgbValue, cbValueMax, pcbValue )
00182       ! rgbValue is an INTEGER*1 value
00183          USE qt_ODBCKinds
00184          INTEGER (SQLRETURN) :: SQLBindParameterI1
00185          INTEGER (SQLHSTMT) :: stmt
00186          INTEGER (SQLUSMALLINT) :: ipar
00187          INTEGER*1 rgbValue
00188          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00189          INTEGER (SQLUINTEGER) :: cbColDef
00190          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00193       END FUNCTION SQLBindParameterI1
00194
00195       FUNCTION SQLBindParameterI2( stmt, ipar,    &
00196                                    fParamType, fCType, fSqlType, cbColDef,           &
```

```
00197                                ibScale, rgbValue, cbValueMax, pcbValue )
00198       ! rgbValue is an INTEGER*2 value
00199          USE qt_ODBCKinds
00200          INTEGER (SQLRETURN) :: SQLBindParameterI2
00201          INTEGER (SQLHSTMT) :: stmt
00202          INTEGER (SQLUSMALLINT) :: ipar
00203          INTEGER*2 rgbValue
00204          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00205          INTEGER (SQLUINTEGER) :: cbColDef
00206          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00209       END FUNCTION SQLBindParameterI2
00210
00211       FUNCTION SQLBindParameterI4( stmt, ipar,    &
00212                                fParamType, fCType, fSqlType, cbColDef,           &
00213                                ibScale, rgbValue, cbValueMax, pcbValue )
00214       ! rgbValue is an INTEGER*4 value
00215          USE qt_ODBCKinds
00216          INTEGER (SQLRETURN) :: SQLBindParameterI4
00217          INTEGER (SQLHSTMT) :: stmt
00218          INTEGER (SQLUSMALLINT) :: ipar
00219          INTEGER*4 rgbValue
00220          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00221          INTEGER (SQLUINTEGER) :: cbColDef
00222          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00225       END FUNCTION SQLBindParameterI4
00226
00227       FUNCTION SQLBindParameterR4( stmt, ipar,    &
00228                                fParamType, fCType, fSqlType, cbColDef,           &
00229                                ibScale, rgbValue, cbValueMax, pcbValue )
00230       ! rgbValue is a REAL*4 value
00231          USE qt_ODBCKinds
00232          INTEGER (SQLRETURN) :: SQLBindParameterR4
00233          INTEGER (SQLHSTMT) :: stmt
00234          INTEGER (SQLUSMALLINT) :: ipar
00235          REAL*4 rgbValue
00236          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00237          INTEGER (SQLUINTEGER) :: cbColDef
00238          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00241       END FUNCTION SQLBindParameterR4
00242
00243       FUNCTION SQLBindParameterDP( stmt, ipar,    &
00244                                fParamType, fCType, fSqlType, cbColDef,           &
00245                                ibScale, rgbValue, cbValueMax, pcbValue )
00246       ! rgbValue is an DOUBLE PRECISION value
00247          USE qt_ODBCKinds
00248          INTEGER (SQLRETURN) :: SQLBindParameterDP
00249          INTEGER (SQLHSTMT) :: stmt
00250          INTEGER (SQLUSMALLINT) :: ipar
00251          DOUBLE PRECISION rgbValue
00252          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00253          INTEGER (SQLUINTEGER) :: cbColDef
00254          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00257       END FUNCTION SQLBindParameterDP
00258
00259    END INTERFACE
00260
00261    INTERFACE   ! added 19.10.2000
00262       FUNCTION SQLBindParameterLP( stmt, ipar,    &
00263                                fParamType, fCType, fSqlType, cbColDef,           &
00264                                ibScale, rgbValue, cbValueMax, pcbValue )
00265       ! rgbValue is a pointer (use LOC()
00266          USE qt_ODBCKinds
00267          INTEGER (SQLRETURN) :: SQLBindParameterLP
00268          INTEGER (SQLHSTMT) :: stmt
00269          INTEGER (SQLUSMALLINT) :: ipar
00270          INTEGER (LP) :: rgbValue
00271          INTEGER (SQLSMALLINT) :: fParamType, fCType, fSqlType, ibScale
00272          INTEGER (SQLUINTEGER) :: cbColDef
00273          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00276       END FUNCTION SQLBindParameterLP
00277    END INTERFACE
00278
00279    INTERFACE SQLBrowseConnect
00280       FUNCTION SQLBrowseConnect( dbc, InConnectionString, cbInConnStr,    &
00281                                OutConnectionString, cbOutConnStr, pbOutConnStrLength )
00282          USE qt_ODBCKinds
00283          INTEGER (SQLRETURN) :: SQLBrowseConnect
00284          INTEGER (SQLHDBC) :: dbc
00285          CHARACTER*(*) InConnectionString, OutConnectionString
00286          INTEGER (SQLSMALLINT) :: cbInConnStr, cbOutConnStr, pbOutConnStrLength
00289       END FUNCTION SQLBrowseConnect
00290    END INTERFACE
00291
00292    INTERFACE SQLBulkOperations
00293       FUNCTION SQLBulkOperations( Stmt, Operation )
00294          USE qt_ODBCKinds
00295          INTEGER (SQLRETURN) :: SQLBulkOperations
00296          INTEGER (SQLHSTMT) :: Stmt
00297          INTEGER (SQLUSMALLINT) :: Operation
00299       END FUNCTION SQLBulkOperations
00300    END INTERFACE
00301
```

```
00302    INTERFACE SQLCancel
00303       FUNCTION SQLCancel( stmt )
00304          USE qt_ODBCKinds
00305          INTEGER (SQLRETURN) :: SQLCancel
00306          INTEGER (SQLHSTMT) :: stmt
00308       END FUNCTION SQLCancel
00309    END INTERFACE
00310
00311    INTERFACE SQLCloseCursor
00312       FUNCTION SQLCloseCursor( Stmt )
00313          USE qt_ODBCKinds
00314          INTEGER (SQLRETURN) :: SQLCloseCursor
00315          INTEGER (SQLHSTMT) :: Stmt
00317       END FUNCTION SQLCloseCursor
00318    END INTERFACE
00319
00320    INTERFACE SQLColAttributeChar
00321
00322    ! charAttribute is a CHARACTER buffer
00323       FUNCTION SQLColAttributeChar( stmt, icol, fieldId, charAttribute,      &
00324                                     lenCharAttribute, CharAttrLength, NumAttribute )
00325          USE qt_ODBCKinds
00326          INTEGER (SQLRETURN) :: SQLColAttributeChar
00327          INTEGER (SQLHSTMT) :: stmt
00328          INTEGER (SQLUSMALLINT) :: icol, fieldId
00329          CHARACTER (LEN=*) :: charAttribute
00330          INTEGER (SQLSMALLINT) :: lenCharAttribute, CharAttrLength
00331          INTEGER (SQLINTEGER) :: NumAttribute
00334       END FUNCTION SQLColAttributeChar
00335    END INTERFACE
00336
00337    INTERFACE SQLColAttribute
00338    ! charAttribute is a pointer
00339       FUNCTION SQLColAttribute( stmt, icol, fieldId, charAttribute,    &
00340                                 lenCharAttribute, CharAttrLength, NumAttribute )
00341          USE qt_ODBCKinds
00342          INTEGER (SQLRETURN) :: SQLColAttribute
00343          INTEGER (SQLHSTMT) :: stmt
00344          INTEGER (SQLUSMALLINT) :: icol, fieldId
00345          INTEGER (SQLPOINTER) :: charAttribute
00346          INTEGER (SQLSMALLINT) :: lenCharAttribute, CharAttrLength
00347          INTEGER (SQLINTEGER) :: NumAttribute
00350        END FUNCTION SQLColAttribute
00351
00352    END INTERFACE
00353
00354    INTERFACE SQLColAttributes
00355       FUNCTION SQLColAttributes( stmt, icol,  &
00356                                  fDescType, rgbDesc, cbDescMax, pcbDesc, pfDesc )
00357          USE qt_ODBCKinds
00358          INTEGER (SQLRETURN) :: SQLColAttributes
00359          INTEGER (SQLHSTMT) :: stmt
00360          INTEGER (SQLUSMALLINT) :: icol, fDescType
00361          CHARACTER (LEN=*) :: rgbDesc
00362          INTEGER (SQLSMALLINT) :: cbDescMax, pcbDesc
00363          INTEGER (SQLINTEGER) :: pfDesc
00366       END FUNCTION SQLColAttributes
00367    END INTERFACE
00368
00369    INTERFACE SQLColumnPrivileges
00370       FUNCTION SQLColumnPrivileges( stmt,   &
00371                                     CatalogName, LenCatName,   &
00372                                     SchemaName, LenSchemaName, &
00373                                     TableName, LenTableName,  &
00374                                     ColumnName, LenColName )
00375          USE qt_ODBCKinds
00376          INTEGER (SQLRETURN) :: SQLColumnPrivileges
00377          INTEGER (SQLHSTMT) :: stmt
00378          CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName, ColumnName
00379          INTEGER (SQLSMALLINT) :: LenCatName, LenSchemaName, LenTableName, LenColName
00382       END FUNCTION SQLColumnPrivileges
00383    END INTERFACE
00384
00385    INTERFACE SQLColumns
00386
00387       FUNCTION SQLColumnsChar( stmt,  &
00388                                szTableQualifier, cbTableQualifier,       &
00389                                szTableOwner, cbTableOwner,               &
00390                                szTableName, cbTableName,                 &
00391                                szColumnName, cbColumnName )      ! changed 14.10.2000: szColumnName, cbColum
00391-1                                 nName )    ! changed 14.10.2000: SQLColumns -> SQLColumnsChar
00392          USE qt_ODBCKinds
00393          INTEGER (SQLRETURN) :: SQLColumnsChar
00394          INTEGER (SQLHSTMT) :: stmt
00395          CHARACTER (LEN=*) :: szTableQualifier, szTableOwner,  &
00396                               szTableName, szColumnName
00397          INTEGER (SQLSMALLINT) :: cbTableQualifier, cbTableOwner, cbTableName, cbColumnName
00400          !DEC$ ATTRIBUTES REFERENCE :: szTableName, szColumnName
00401       END FUNCTION SQLColumnsChar
00402
00403       ! 14.10.2000: added SQLColumnsLP (all arguments being transferred as values, use LOC() to pass a refer
```

```
00403-1          ence)
00404       FUNCTION SQLColumnsLP( stmt,  &
00405                              szTableQualifier, cbTableQualifier,       &
00406                              szTableOwner, cbTableOwner,               &
00407                              szTableName, cbTableName,                 &
00408                              szColumnName, cbColumnName )
00409          USE qt_ODBCKinds
00410          INTEGER (SQLRETURN) :: SQLColumnsLP
00411          INTEGER (SQLHSTMT) :: stmt
00412          INTEGER (LP) :: szTableQualifier, szTableOwner,  &
00413                          szTableName, szColumnName
00414          INTEGER (SQLSMALLINT) :: cbTableQualifier, cbTableOwner, cbTableName, cbColumnName
00416       END FUNCTION SQLColumnsLP
00417
00418    END INTERFACE
00419
00420    INTERFACE SQLConnect
00421       FUNCTION SQLConnect( dbc, szDSN, cbDSN, szUID, cbUID, szAuthStr, cbAuthStr )
00422          USE qt_ODBCKinds
00423          INTEGER (SQLRETURN) ::SQLConnect
00424          INTEGER (SQLHDBC) :: dbc
00425          CHARACTER*(*) szDSN, szUID, szAuthStr
00426          INTEGER (SQLSMALLINT) :: cbDSN, cbUID, cbAuthStr
00429       END FUNCTION SQLConnect
00430    END INTERFACE
00431
00432    INTERFACE SQLCopyDesc
00433       FUNCTION SQLCopyDesc( SourceDescHandle, TargetDescHandle )
00434          USE qt_ODBCKinds
00435          INTEGER (SQLRETURN) :: SQLCopyDesc
00436          INTEGER (SQLHDESC) :: SourceDescHandle, TargetDescHandle
00438       END FUNCTION SQLCopyDesc
00439    END INTERFACE
00440
00441    INTERFACE SQLDataSources
00442       FUNCTION SQLDataSources( env, fDirection,   &
00443                                szDSN, cbDSNMax, pcbDSN,   &
00444                                szDescription, cbDescriptionMax, pcbDescription )
00445          USE qt_ODBCKinds
00446          INTEGER (SQLRETURN) :: SQLDataSources
00447          INTEGER (SQLHENV) :: env
00448          INTEGER (SQLUSMALLINT) :: fDirection
00449          CHARACTER (LEN=*) :: szDSN, szDescription
00450          INTEGER (SQLSMALLINT) :: cbDSNMax, pcbDSN, cbDescriptionMax, pcbDescription
00454       END FUNCTION SQLDataSources
00455    END INTERFACE
00456
00457    INTERFACE SQLDescribeCol
00458       FUNCTION SQLDescribeCol( stmt, icol,    &
00459                                szColName, cbColNameMax, pcbColName,   &
00460                                pfSqlType, pcbColDef, pibScale, pfNullable )
00461          USE qt_ODBCKinds
00462          INTEGER (SQLRETURN) :: SQLDescribeCol
00463          INTEGER (SQLHSTMT) :: stmt
00464          INTEGER (SQLUSMALLINT) :: icol
00465          CHARACTER (LEN=*) :: szColName
00466          INTEGER (SQLSMALLINT) :: cbColNameMax, pcbColName, pfSqlType, pibScale, pfNullable
00467          INTEGER (SQLUINTEGER) :: pcbColDef
00472       END FUNCTION SQLDescribeCol
00473    END INTERFACE
00474
00475    INTERFACE SQLDescribeParam
00476       FUNCTION SQLDescribeParam( stmt, ipar, pfSqlType,   &
00477                                  pcbColDef, pibScale, pfNullable )
00478          USE qt_ODBCKinds
00479          INTEGER (SQLRETURN) :: SQLDescribeParam
00480          INTEGER (SQLHSTMT) :: stmt
00481          INTEGER (SQLUSMALLINT) :: ipar
00482          INTEGER (SQLSMALLINT) :: pfSqlType, pibScale, pfNullable
00483          INTEGER (SQLUINTEGER) :: pcbColDef
00487       END FUNCTION SQLDescribeParam
00488    END INTERFACE
00489
00490    INTERFACE SQLDisconnect
00491       FUNCTION SQLDisconnect( dbc )
00492          USE qt_ODBCKinds
00493          INTEGER (SQLRETURN) :: SQLDisconnect
00494          INTEGER (SQLHDBC) :: dbc
00496       END FUNCTION SQLDisconnect
00497    END INTERFACE
00498
00499    INTERFACE   ! SQLDriverConnect; DVF5 -> ERROR (could not find generic interface specific function...!)
00500       FUNCTION SQLDriverConnect( dbc, wnd,  &
00501                                  szConnStrIn, cbConnStrIn,  &
00502                                  szConnStrOut, cbConnStrOutMax, pcbConnStrOut, &
00503                                  fDriverCompletion)
00504          USE qt_ODBCKinds
00505          INTEGER (SQLRETURN) :: SQLDriverConnect
00506          INTEGER (SQLHDBC) :: dbc
00507          INTEGER (SQLHWND) :: wnd
00508          CHARACTER (LEN=*) :: szConnStrIn, szConnStrOut
```

```
00509          INTEGER (SQLSMALLINT) :: cbConnStrIn, cbConnStrOutMax, pcbConnStrOut
00510          INTEGER (SQLUSMALLINT) :: fDriverCompletion
00514       END FUNCTION SQLDriverConnect
00515    END INTERFACE
00516
00517    INTERFACE SQLDrivers
00518       FUNCTION SQLDrivers( env, fDirection,   &
00519                            szDrvDesc, cbDrvDescMax, pcbDrvDesc,   &
00520                            szDrvAttr, cbDrvAttrMax, pcbDrvAttr )
00521          USE qt_ODBCKinds
00522          INTEGER (SQLRETURN) :: SQLDrivers
00523          INTEGER (SQLHENV) :: env
00524          INTEGER (SQLUSMALLINT) :: fDirection
00525          CHARACTER (LEN=*) :: szDrvDesc, szDrvAttr
00526          INTEGER (SQLSMALLINT) :: cbDrvDescMax, pcbDrvDesc, cbDrvAttrMax, pcbDrvAttr
00530       END FUNCTION SQLDrivers
00531    END INTERFACE
00532
00533    INTERFACE SQLEndTran
00534      FUNCTION SQLEndTran( HandleType, hndl, CompletionType )
00535          USE qt_ODBCKinds
00536          INTEGER (SQLRETURN) :: SQLEndTran
00537          INTEGER (SQLSMALLINT) :: HandleType
00538          INTEGER (SQLHANDLE) :: hndl
00539          INTEGER (SQLSMALLINT) :: CompletionType
00541       END FUNCTION SQLEndTran
00542    END INTERFACE
00543
00544    INTERFACE SQLError
00545       FUNCTION SQLError( env, dbc, stmt, szSqlState, pfNativeError,  &
00546                          szErrorMsg, cbErrorMsgMax, pcbErrorMsg )
00547          USE qt_ODBCKinds
00548          INTEGER (SQLRETURN) :: SQLError
00549          INTEGER (SQLHENV) :: env
00550          INTEGER (SQLHDBC) :: dbc
00551          INTEGER (SQLHSTMT) :: stmt
00552          CHARACTER*(*) szSqlState, szErrorMsg
00553          INTEGER (SQLINTEGER) :: pfNativeError
00554          INTEGER (SQLSMALLINT) :: cbErrorMsgMax, pcbErrorMsg
00555           !DEC$ ATTRIBUTES STDCALL, ALIAS : '_SQLError@32' :: SQLError
00557      END FUNCTION SQLError
00558    END INTERFACE
00559
00560    INTERFACE SQLExecDirect
00561       FUNCTION SQLExecDirect( stmt, szSqlStr, cbSqlStr )
00562          USE qt_ODBCKinds
00563          INTEGER (SQLRETURN) :: SQLExecDirect
00564          INTEGER (SQLHSTMT) :: stmt
00565          CHARACTER*(*) szSqlStr
00566          INTEGER (SQLINTEGER) :: cbSqlStr
00569       END FUNCTION SQLExecDirect
00570    END INTERFACE
00571
00572    INTERFACE SQLExecute
00573       FUNCTION SQLExecute( stmt )
00574          USE qt_ODBCKinds
00575          INTEGER (SQLRETURN) :: SQLExecute
00576          INTEGER (SQLHSTMT) :: stmt
00578       END FUNCTION SQLExecute
00579    END INTERFACE
00580
00581    INTERFACE SQLExtendedFetch
00582       FUNCTION SQLExtendedFetch( stmt, fFetchType, irow, pcrow, rgfRowStatus )
00583          USE qt_ODBCKinds
00584          INTEGER (RETCODE) :: SQLExtendedFetch
00586          INTEGER (HSTMT) :: stmt
00587          INTEGER (UWORD) :: fFetchType, rgfRowStatus
00588          INTEGER (SDWORD) :: irow
00589          INTEGER (UDWORD) :: pcrow
00591       END FUNCTION SQLExtendedFetch
00592    END INTERFACE
00593
00594    INTERFACE SQLFetch
00595       FUNCTION SQLFetch( stmt )
00596          USE qt_ODBCKinds
00597          INTEGER (SQLRETURN) :: SQLFetch
00598          INTEGER (SQLHSTMT) :: stmt
00600       END FUNCTION SQLFetch
00601    END INTERFACE
00602
00603    INTERFACE SQLFetchScroll
00604       FUNCTION SQLFetchScroll( stmt, FetchOrientation, FetchOffset )
00605          USE qt_ODBCKinds
00606          INTEGER (SQLRETURN) :: SQLFetchScroll
00608          INTEGER (SQLHSTMT) :: stmt
00609          INTEGER (SQLSMALLINT) :: FetchOrientation
00610          INTEGER (SQLINTEGER) :: FetchOffset
00611       END FUNCTION SQLFetchScroll
00612    END INTERFACE
00613
00614    INTERFACE SQLForeignKeys
```

```
00615       FUNCTION SQLForeignKeys( stmt, PKCatalogName, PKCatNameLength,    &
00616                                      PKSchemaName, PKSchemaNameLength,  &
00617                                      PKTableName, PKTableNameLength,    &
00618                                      FKCatalogName, FKCatalogNameLength,&
00619                                      FKSchemaName, FKSchemaNameLength,  &
00620                                      FKTableName, FKTableNameLength)
00621          USE qt_ODBCKinds
00622          INTEGER (SQLRETURN) :: SQLForeignKeys
00624          INTEGER (SQLHSTMT) :: stmt
00625          CHARACTER (LEN=*) :: PKCatalogName, PKSchemaName, PKTableName,  &
00626                               FKCatalogName, FKSchemaName, FKTableName
00628          !DEC$ ATTRIBUTES REFERENCE :: FKCatalogName, FKSchemaName, FKTableName
00629          INTEGER (SQLSMALLINT) :: PKCatNameLength, PKSchemaNameLength, PKTableNameLength, &
00630                                   FKCatalogNameLength, FKSchemaNameLength, FKTableNameLength
00631       END FUNCTION SQLForeignKeys
00632    END INTERFACE
00633
00634    INTERFACE SQLFreeConnect
00635       FUNCTION SQLFreeConnect( dbc )
00636          USE qt_ODBCKinds
00637          INTEGER (SQLRETURN) :: SQLFreeConnect
00638          INTEGER (SQLHDBC) :: dbc
00640       END FUNCTION SQLFreeConnect
00641    END INTERFACE
00642
00643    INTERFACE SQLFreeEnv
00644       FUNCTION SQLFreeEnv( env )
00645          USE qt_ODBCKinds
00646          INTEGER (SQLRETURN) :: SQLFreeEnv
00647          INTEGER (SQLHENV) :: env
00649       END FUNCTION SQLFreeEnv
00650    END INTERFACE
00651
00652    INTERFACE SQLFreeHandle
00653       FUNCTION SQLFreeHandle( HndType, Hnd )
00654          USE qt_ODBCKinds
00655          INTEGER (SQLRETURN) :: SQLFreeHandle
00656          INTEGER (SQLSMALLINT) :: HndType
00657          INTEGER (SQLHANDLE) :: Hnd
00659       END FUNCTION SQLFreeHandle
00660    END INTERFACE
00661
00662    INTERFACE SQLFreeStmt
00663       FUNCTION SQLFreeStmt( stmt, fOption )
00664          USE qt_ODBCKinds
00665          INTEGER (SQLRETURN) :: SQLFreeStmt
00666          INTEGER (SQLHSTMT) :: stmt
00667          INTEGER (SQLUSMALLINT) :: fOption
00669       END FUNCTION SQLFreeStmt
00670    END INTERFACE
00671
00672    INTERFACE SQLGetConnectAttrChar
00673    ! ValuePtr is a CHARACTER buffer
00674       FUNCTION SQLGetConnectAttrChar( dbc, Attrib, ValuePtr, LenValuePtr, ValuePtrLength)
00675          USE qt_ODBCKinds
00677          INTEGER (SQLRETURN) :: SQLGetConnectAttrChar
00678          INTEGER (SQLHDBC) :: dbc
00679          INTEGER (SQLINTEGER) :: Attrib, LenValuePtr, ValuePtrLength
00680          CHARACTER (LEN=*) :: ValuePtr
00682       END FUNCTION SQLGetConnectAttrChar
00683    END INTERFACE
00684
00685    INTERFACE SQLGetConnectAttr
00686    ! ValuePtr is a pointer to a buffer
00687       FUNCTION SQLGetConnectAttr( dbc, Attrib, ValuePtr, LenValuePtr, ValuePtrLength)
00688          USE qt_ODBCKinds
00690          INTEGER (SQLRETURN) :: SQLGetConnectAttr
00691          INTEGER (SQLHDBC) :: dbc
00692          INTEGER (SQLINTEGER) :: Attrib, LenValuePtr, ValuePtrLength
00693          INTEGER (SQLPOINTER) :: ValuePtr
00695       END FUNCTION SQLGetConnectAttr
00696    END INTERFACE
00697
00698    INTERFACE SQLGetConnectOption
00699
00700       FUNCTION SQLGetConnectOptionChar( dbc, fOption, pvParam )
00701       ! pvParam is a CHARACTER buffer
00702          USE qt_ODBCKinds
00703          INTEGER (SQLRETURN) :: SQLGetConnectOptionChar
00704          INTEGER (SQLHDBC) :: dbc
00705          INTEGER (SQLUSMALLINT) :: fOption
00706          CHARACTER (LEN=*) :: pvParam
00709       END FUNCTION SQLGetConnectOptionChar
00710
00711       FUNCTION SQLGetConnectOptionI4( dbc, fOption, pvParam )
00712       ! pvParam is an INTEGER*4 value
00713          USE qt_ODBCKinds
00714          INTEGER (SQLRETURN) :: SQLGetConnectOptionI4
00715          INTEGER (SQLHDBC) :: dbc
00716          INTEGER (SQLUSMALLINT) :: fOption
00717          INTEGER*4 pvParam
```

```
00720       END FUNCTION SQLGetConnectOptionI4
00721
00722    END INTERFACE
00723
00724    INTERFACE SQLGetCursorName
00725       FUNCTION SQLGetCursorName( stmt, szCursor, cbCursorMax, pcbCursor )
00726          USE qt_ODBCKinds
00727          INTEGER (SQLRETURN) :: SQLGetCursorName
00728          INTEGER (SQLHSTMT) :: stmt
00729          CHARACTER (LEN=*) :: szCursor
00730          INTEGER (SQLSMALLINT) :: cbCursorMax, pcbCursor
00733       END FUNCTION SQLGetCursorName
00734    END INTERFACE
00735
00736    INTERFACE SQLGetData
00737
00738       FUNCTION SQLGetDataChar( stmt, icol, fCType,   &
00739                                rgbValue, cbValueMax, pcbValue )
00740       ! rgbValue is a CHARACTER buffer
00741          USE qt_ODBCKinds
00742          INTEGER (SQLRETURN) :: SQLGetDataChar
00743          INTEGER (SQLHSTMT) :: stmt
00744          INTEGER (SQLUSMALLINT) :: icol
00745          CHARACTER (LEN=*) :: rgbValue
00746          INTEGER (SQLSMALLINT) :: fCType
00747          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00750       END FUNCTION SQLGetDataChar
00751
00752       FUNCTION SQLGetDataI2( stmt, icol, fCType,   &
00753                              rgbValue, cbValueMax, pcbValue )
00754       ! rgbValue is an INTEGER*2 value
00755          USE qt_ODBCKinds
00756          INTEGER (SQLRETURN) :: SQLGetDataI2
00757          INTEGER (SQLHSTMT) :: stmt
00758          INTEGER (SQLUSMALLINT) :: icol
00759          INTEGER*2 rgbValue
00760          INTEGER (SQLSMALLINT) :: fCType
00761          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00764       END FUNCTION SQLGetDataI2
00765
00766       FUNCTION SQLGetDataI4( stmt, icol, fCType,   &
00767                              rgbValue, cbValueMax, pcbValue )
00768       ! rgbValue is an INTEGER*4 value
00769          USE qt_ODBCKinds
00770          INTEGER (SQLRETURN) :: SQLGetDataI4
00771          INTEGER (SQLHSTMT) :: stmt
00772          INTEGER (SQLUSMALLINT) :: icol
00773          INTEGER*4 rgbValue
00774          INTEGER (SQLSMALLINT) :: fCType
00775          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00778       END FUNCTION SQLGetDataI4
00779
00780       FUNCTION SQLGetDataR4( stmt, icol, fCType,   &
00781                              rgbValue, cbValueMax, pcbValue )
00782       ! rgbValue is a REAL*4 value
00783          USE qt_ODBCKinds
00784          INTEGER (SQLRETURN) :: SQLGetDataR4
00785          INTEGER (SQLHSTMT) :: stmt
00786          INTEGER (SQLUSMALLINT) :: icol
00787          REAL*4 rgbValue
00788          INTEGER (SQLSMALLINT) :: fCType
00789          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00792       END FUNCTION SQLGetDataR4
00793
00794       FUNCTION SQLGetDataDP( stmt, icol, fCType,   &
00795                              rgbValue, cbValueMax, pcbValue )
00796       ! rgbValue is a DOUBLE PRECISION value
00797          USE qt_ODBCKinds
00798          INTEGER (SQLRETURN) :: SQLGetDataDP
00799          INTEGER (SQLHSTMT) :: stmt
00800          INTEGER (SQLUSMALLINT) :: icol
00801          DOUBLE PRECISION rgbValue
00802          INTEGER (SQLSMALLINT) :: fCType
00803          INTEGER (SQLINTEGER) :: cbValueMax, pcbValue
00806       END FUNCTION SQLGetDataDP
00807
00808    END INTERFACE
00809
00810    INTERFACE SQLGetDescFieldChar
00811
00812    ! ValuePtr is a CHARACTER buffer
00813       FUNCTION SQLGetDescFieldChar( DescriptorHandle, RecNumber, FieldIdentifier,  &
00814                                     ValuePtr, LenValuePtr, ValuePtrLen )
00815          USE qt_ODBCKinds
00817          INTEGER (SQLRETURN) :: SQLGetDescFieldChar
00818          INTEGER (SQLHDESC) :: DescriptorHandle
00819          INTEGER (SQLSMALLINT) :: RecNumber, FieldIdentifier
00820          CHARACTER (LEN=*) :: ValuePtr
00821          INTEGER (SQLINTEGER) :: LenValuePtr, ValuePtrLen
00823       END FUNCTION SQLGetDescFieldChar
00824    END INTERFACE
00825
```

```
00826    INTERFACE SQLGetDescField
00827    ! ValuePtr is a pointer
00828       FUNCTION SQLGetDescField( DescriptorHandle, RecNumber, FieldIdentifier,  &
00829                                 ValuePtr, LenValuePtr, ValuePtrLen )
00830          USE qt_ODBCKinds
00832          INTEGER (SQLRETURN) :: SQLGetDescField
00833          INTEGER (SQLHDESC) :: DescriptorHandle
00834          INTEGER (SQLSMALLINT) :: RecNumber, FieldIdentifier
00835          INTEGER (SQLPOINTER) :: ValuePtr
00836          INTEGER (SQLINTEGER) :: LenValuePtr, ValuePtrLen
00838       END FUNCTION SQLGetDescField
00839    END INTERFACE
00840
00841    INTERFACE SQLGetDescRec
00842       FUNCTION SQLGetDescRec( DescriptorHandle, RecNumber, DescName,  &
00843                               LenDescName, DescNameLength, TypePtr, SubTypePtr,   &
00844                               LengthPtr, PrecisionPtr, ScalePtr, NullablePtr )
00845          USE qt_ODBCKinds
00847          INTEGER (SQLRETURN) :: SQLGetDescRec
00848          INTEGER (SQLHDESC) :: DescriptorHandle
00849          INTEGER (SQLSMALLINT) :: RecNumber, LenDescName, DescNameLength,  &
00850                                   TypePtr, SubTypePtr, PrecisionPtr, ScalePtr, NullablePtr
00851          INTEGER (SQLINTEGER) :: LengthPtr
00852          CHARACTER (LEN=*) :: DescName
00854          !DEC$ ATTRIBUTES REFERENCE :: LengthPtr, PrecisionPtr, ScalePtr, NullablePtr
00855       END FUNCTION SQLGetDescRec
00856    END INTERFACE
00857
00858    INTERFACE SQLGetDiagField
00859       FUNCTION SQLGetDiagField( HandleType, Hndl, RecNumber, DiagIdentifier,  &
00860                                 DiagInfoPtr, LenDiagInfo, DiagInfoLen )
00861          USE qt_ODBCKinds
00863          INTEGER (SQLRETURN) :: SQLGetDiagField
00864          INTEGER (SQLSMALLINT) :: HandleType, RecNumber, DiagIdentifier,   &
00865                                   LenDiagInfo, DiagInfoLen
00866          INTEGER (SQLHANDLE) :: Hndl
00867          INTEGER (SQLPOINTER) ::  DiagInfoPtr
00869       END FUNCTION SQLGetDiagField
00870    END INTERFACE
00871
00872    INTERFACE SQLGetDiagRec
00873       FUNCTION SQLGetDiagRec( HandleType, Hndl, RecNumber, Sqlstate,  &
00874                               NativeError, MessageText, LenMsgText, MsgTextLen )
00875          USE qt_ODBCKinds
00877          INTEGER (SQLRETURN) :: SQLGetDiagRec
00878          INTEGER (SQLSMALLINT) :: HandleType, RecNumber, LenMsgText, MsgTextLen
00879          INTEGER (SQLHANDLE) :: Hndl
00880          CHARACTER (LEN=*) :: Sqlstate, MessageText
00881          INTEGER (SQLINTEGER) :: NativeError
00883       END FUNCTION SQLGetDiagRec
00884    END INTERFACE
00885
00886    INTERFACE SQLGetEnvAttr
00887
00888    ! Value is a CHARACTER buffer
00889       FUNCTION SQLGetEnvAttrChar( env, Attribute, Value, LenValue, ValueLength )
00890          USE qt_ODBCKinds
00892          INTEGER (SQLRETURN) :: SQLGetEnvAttrChar
00893          INTEGER (SQLHENV) :: env
00894          INTEGER (SQLINTEGER) :: Attribute, LenValue, ValueLength
00895          CHARACTER (LEN=*) :: Value
00897       END FUNCTION SQLGetEnvAttrChar
00898    ! Value is an INTEGER
00899       FUNCTION SQLGetEnvAttrI4( env, Attribute, Value, LenValue, ValueLength )
00900          USE qt_ODBCKinds
00902          INTEGER (SQLRETURN) :: SQLGetEnvAttrI4
00903          INTEGER (SQLHENV) :: env
00904          INTEGER (SQLINTEGER) :: Attribute, LenValue, ValueLength
00905          INTEGER (SQLINTEGER) :: Value
00907       END FUNCTION SQLGetEnvAttrI4
00908
00909    END INTERFACE
00910
00911    INTERFACE SQLGetFunctions
00912       FUNCTION SQLGetFunctions( dbc, fFunction, pfExists )
00913          USE qt_ODBCKinds
00914          INTEGER (SQLRETURN) :: SQLGetFunctions
00915          INTEGER (SQLHDBC) :: dbc
00916          INTEGER (SQLUSMALLINT) :: fFunction, pfExists
00919       END FUNCTION SQLGetFunctions
00920    END INTERFACE
00921
00922    INTERFACE SQLGetInfo
00923
00924       FUNCTION SQLGetInfoChar( dbc, fInfoType, rgbInfoValue,     &
00925                                cbInfoValueMax, pcbInfoValue )
00926       ! rgbInfoValue is a CHARACTER buffer
00927          USE qt_ODBCKinds
00928          INTEGER (SQLRETURN) :: SQLGetInfoChar
00929          INTEGER (SQLHDBC) :: dbc
00930          INTEGER (SQLUSMALLINT) :: fInfoType
```

```
00931          CHARACTER (LEN=*) :: rgbInfoValue
00932          INTEGER (SQLSMALLINT) :: cbInfoValueMax, pcbInfoValue
00935       END FUNCTION SQLGetInfoChar
00936
00937       FUNCTION SQLGetInfoI2( dbc, fInfoType, rgbInfoValue,    &
00938                              cbInfoValueMax, pcbInfoValue )
00939       ! rgbInfoValue is of type INTEGER*2
00940          USE qt_ODBCKinds
00941          INTEGER (SQLRETURN) :: SQLGetInfoI2
00942          INTEGER (SQLHDBC) :: dbc
00943          INTEGER (SQLUSMALLINT) :: fInfoType
00944          INTEGER*2 rgbInfoValue
00945          INTEGER (SQLSMALLINT) :: cbInfoValueMax, pcbInfoValue
00948       END FUNCTION SQLGetInfoI2
00949
00950       FUNCTION SQLGetInfoI4( dbc, fInfoType, rgbInfoValue,    &
00951                              cbInfoValueMax, pcbInfoValue )
00952       ! rgbInfoValue is of type INTEGER*4
00953          USE qt_ODBCKinds
00954          INTEGER (SQLRETURN) :: SQLGetInfoI4
00955          INTEGER (SQLHDBC) :: dbc
00956          INTEGER (SQLUSMALLINT) :: fInfoType
00957          INTEGER*4 rgbInfoValue
00958          INTEGER (SQLSMALLINT) :: cbInfoValueMax, pcbInfoValue
00961       END FUNCTION SQLGetInfoI4
00962
00963    END INTERFACE
00964
00965    INTERFACE SQLGetStmtAttrChar
00966
00967    ! Value is a CHARACTER buffer
00968       FUNCTION SQLGetStmtAttrChar( stmt, Attribute, Value, LenValue, ValueLength )
00969          USE qt_ODBCKinds
00971          INTEGER (SQLRETURN) :: SQLGetStmtAttrChar
00972          INTEGER (SQLHSTMT) :: stmt
00973          INTEGER (SQLINTEGER) :: Attribute, LenValue, ValueLength
00974          CHARACTER (LEN=*) :: Value
00976       END FUNCTION SQLGetStmtAttrChar
00977    END INTERFACE
00978
00979    INTERFACE SQLGetStmtAttr
00980    ! Value is a pointer to a buffer
00981       FUNCTION SQLGetStmtAttr( stmt, Attribute, ValuePtr, LenValue, ValueLength )
00982          USE qt_ODBCKinds
00984          INTEGER (SQLRETURN) :: SQLGetStmtAttr
00985          INTEGER (SQLHSTMT) :: stmt
00986          INTEGER (SQLINTEGER) :: Attribute, LenValue, ValueLength
00987          INTEGER (SQLPOINTER) :: ValuePtr
00989       END FUNCTION SQLGetStmtAttr
00990
00991    END INTERFACE
00992
00993    INTERFACE SQLGetStmtOption
00994
00995       FUNCTION SQLGetStmtOptionChar( stmt, fOption, pvParam )
00996       ! pvParam is a CHARACTER buffer
00997          USE qt_ODBCKinds
00998          INTEGER (SQLRETURN) :: SQLGetStmtOptionChar
00999          INTEGER (SQLHSTMT) :: stmt
01000          INTEGER (SQLUSMALLINT) :: fOption
01001          CHARACTER (LEN=*) :: pvParam
01004       END FUNCTION SQLGetStmtOptionChar
01005
01006       FUNCTION SQLGetStmtOptionI4( stmt, fOption, pvParam )
01007       ! pvParam is an INTEGER*4 value
01008          USE qt_ODBCKinds
01009          INTEGER (SQLRETURN) :: SQLGetStmtOptionI4
01010          INTEGER (SQLHSTMT) :: stmt
01011          INTEGER (SQLUSMALLINT) :: fOption
01012          INTEGER*4 pvParam
01015       END FUNCTION SQLGetStmtOptionI4
01016
01017    END INTERFACE
01018
01019    INTERFACE SQLGetTypeInfo
01020       FUNCTION SQLGetTypeInfo( stmt, fSqlType )
01021          USE qt_ODBCKinds
01022          INTEGER (SQLRETURN) :: SQLGetTypeInfo
01023          INTEGER (SQLHSTMT) :: stmt
01024          INTEGER (SQLSMALLINT) :: fSqlType
01026       END FUNCTION SQLGetTypeInfo
01027    END INTERFACE
01028
01029    INTERFACE SQLMoreResults
01030       FUNCTION SQLMoreResults( stmt )
01031          USE qt_ODBCKinds
01032          INTEGER (SQLRETURN) :: SQLMoreResults
01033          INTEGER (SQLHSTMT) :: stmt
01035       END FUNCTION SQLMoreResults
01036    END INTERFACE
01037
```

```
01038    INTERFACE SQLNativeSql
01039       FUNCTION SQLNativeSql( dbc, szSqlStrIn, cbSqlStrIn,  &
01040                              szSqlStr, cbSqlStrMax, pcbSqlStr )
01041          USE qt_ODBCKinds
01042          INTEGER (SQLRETURN) :: SQLNativeSql
01043          INTEGER (SQLHDBC) :: dbc
01044          CHARACTER (LEN=*) :: szSqlStrIn, szSqlStr
01045          INTEGER (SQLINTEGER) :: cbSqlStrIn, cbSqlStrMax, pcbSqlStr
01048       END FUNCTION SQLNativeSql
01049    END INTERFACE
01050
01051    INTERFACE SQLNumParams
01052       FUNCTION SQLNumParams( stmt, pcpar )
01053          USE qt_ODBCKinds
01054          INTEGER (SQLRETURN) :: SQLNumParams
01055          INTEGER (SQLHSTMT) :: stmt
01056          INTEGER (SQLSMALLINT) :: pcpar
01059       END FUNCTION SQLNumParams
01060    END INTERFACE
01061
01062    INTERFACE SQLNumResultCols
01063       FUNCTION SQLNumResultCols( stmt, pccol )
01064          USE qt_ODBCKinds
01065          INTEGER (SQLRETURN) :: SQLNumResultCols
01066          INTEGER (SQLHSTMT) :: stmt
01067          INTEGER (SQLSMALLINT) :: pccol
01070       END FUNCTION SQLNumResultCols
01071    END INTERFACE
01072
01073    INTERFACE SQLParamData
01074
01075       FUNCTION SQLParamDataChar( stmt, prgbValue )
01076       ! prgbValue is a CHARACTER buffer
01077          USE qt_ODBCKinds
01078          INTEGER (SQLRETURN) :: SQLParamDataChar
01079          INTEGER (SQLHSTMT) :: stmt
01080          CHARACTER (LEN=*) :: prgbValue
01083       END FUNCTION SQLParamDataChar
01084
01085       FUNCTION SQLParamDataI2( stmt, prgbValue )
01086       ! prgbValue is an INTEGER*2 value
01087          USE qt_ODBCKinds
01088          INTEGER (SQLRETURN) :: SQLParamDataI2
01089          INTEGER (SQLHSTMT) :: stmt
01090          INTEGER*2 prgbValue
01093       END FUNCTION SQLParamDataI2
01094
01095       FUNCTION SQLParamDataI4( stmt, prgbValue )
01096       ! prgbValue is an INTEGER*4 value
01097          USE qt_ODBCKinds
01098          INTEGER (SQLRETURN) :: SQLParamDataI4
01099          INTEGER (SQLHSTMT) :: stmt
01100          INTEGER*4 prgbValue
01103       END FUNCTION SQLParamDataI4
01104
01105       FUNCTION SQLParamDataR4( stmt, prgbValue )
01106       ! prgbValue is an REAL*4 value
01107          USE qt_ODBCKinds
01108          INTEGER (SQLRETURN) :: SQLParamDataR4
01109          INTEGER (SQLHSTMT) :: stmt
01110          REAL*4 prgbValue
01113       END FUNCTION SQLParamDataR4
01114
01115       FUNCTION SQLParamDataDP( stmt, prgbValue )
01116       ! prgbValue is an DOUBLE PRECISION value
01117          USE qt_ODBCKinds
01118          INTEGER (SQLRETURN) :: SQLParamDataDP
01119          INTEGER (SQLHSTMT) :: stmt
01120          DOUBLE PRECISION prgbValue
01123       END FUNCTION SQLParamDataDP
01124
01125    END INTERFACE
01126
01127    INTERFACE SQLParamOptions
01128       FUNCTION SQLParamOptions( stmt, crow, pirow )
01129          USE qt_ODBCKinds
01131          INTEGER (RETCODE) :: SQLParamOptions
01132          INTEGER (HSTMT) :: stmt
01133          INTEGER (UDWORD) :: crow, pirow
01135       END FUNCTION SQLParamOptions
01136    END INTERFACE
01137
01138    INTERFACE SQLPrepare
01139       FUNCTION SQLPrepare( stmt, szSqlStr, cbSqlStr )
01140          USE qt_ODBCKinds
01141          INTEGER (SQLRETURN) :: SQLPrepare
01142          INTEGER (SQLHSTMT) :: stmt
01143          CHARACTER (LEN=*) :: szSqlStr
01144          INTEGER (SQLINTEGER) :: cbSqlStr
01147       END FUNCTION SQLPrepare
01148    END INTERFACE
```

```
01149
01150    INTERFACE SQLPrimaryKeys
01151       FUNCTION SQLPrimaryKeys( stmt, CatalogName, CatNameLength,   &
01152                                SchemaName, SchemaNameLength, &
01153                                TableName, TableNameLength )
01154          USE qt_ODBCKinds
01156          INTEGER (SQLRETURN) :: SQLPrimaryKeys
01157          INTEGER (SQLHSTMT) :: stmt
01158          CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName
01159          INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength, TableNameLength
01161       END FUNCTION SQLPrimaryKeys
01162    END INTERFACE
01163
01164    INTERFACE SQLProcedureColumns
01165       FUNCTION SQLProcedureColumns( stmt, CatalogName, CatNameLength,   &
01166                                     SchemaName, SchemaNameLength, &
01167                                     ProcName, ProcNameLength,     &
01168                                     ColumnName, ColNameLength )
01169          USE qt_ODBCKinds
01171          INTEGER (SQLRETURN) :: SQLProcedureColumns
01172          INTEGER (SQLHSTMT) :: stmt
01173          CHARACTER (LEN=*) :: CatalogName, SchemaName, ProcName, ColumnName
01174          INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength,   &
01175                                   ProcNameLength, ColNameLength
01177       END FUNCTION SQLProcedureColumns
01178    END INTERFACE
01179
01180    INTERFACE SQLProcedures
01181       FUNCTION SQLProcedures( stmt, CatalogName, CatNameLength,   &
01182                               SchemaName, SchemaNameLength, &
01183                               ProcName, ProcNameLength )
01184          USE qt_ODBCKinds
01186          INTEGER (SQLRETURN) :: SQLProcedures
01187          INTEGER (SQLHSTMT) :: stmt
01188          CHARACTER (LEN=*) :: CatalogName, SchemaName, ProcName
01189          INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength, ProcNameLength
01191       END FUNCTION SQLProcedures
01192    END INTERFACE
01193
01194    INTERFACE SQLPutData
01195
01196       FUNCTION SQLPutDataChar( stmt, rgbValue, cbValue )
01197       ! rgbValue is a CHARACTER buffer
01198          USE qt_ODBCKinds
01199          INTEGER (SQLRETURN) :: SQLPutDataChar
01200          INTEGER (SQLHSTMT) :: stmt
01201          CHARACTER (LEN=*) :: rgbValue
01202          INTEGER (SQLINTEGER) :: cbValue
01205       END FUNCTION SQLPutDataChar
01206
01207       FUNCTION SQLPutDataI2( stmt, rgbValue, cbValue )
01208       ! rgbValue is an INTEGER*2 value
01209          USE qt_ODBCKinds
01210          INTEGER (SQLRETURN) :: SQLPutDataI2
01211          INTEGER (SQLHSTMT) :: stmt
01212          INTEGER*2 rgbValue
01213          INTEGER (SQLINTEGER) :: cbValue
01216       END FUNCTION SQLPutDataI2
01217
01218       FUNCTION SQLPutDataI4( stmt, rgbValue, cbValue )
01219       ! rgbValue is an INTEGER*4 value
01220          USE qt_ODBCKinds
01221          INTEGER (SQLRETURN) :: SQLPutDataI4
01222          INTEGER (SQLHSTMT) :: stmt
01223          INTEGER*4 rgbValue
01224          INTEGER (SQLINTEGER) :: cbValue
01227       END FUNCTION SQLPutDataI4
01228
01229       FUNCTION SQLPutDataR4( stmt, rgbValue, cbValue )
01230       ! rgbValue is an REAL*4 value
01231          USE qt_ODBCKinds
01232          INTEGER (SQLRETURN) :: SQLPutDataR4
01233          INTEGER (SQLHSTMT) :: stmt
01234          REAL*4 rgbValue
01235          INTEGER (SQLINTEGER) :: cbValue
01238       END FUNCTION SQLPutDataR4
01239
01240       FUNCTION SQLPutDataDP( stmt, rgbValue, cbValue )
01241       ! rgbValue is an DOUBLE PRECISION value
01242          USE qt_ODBCKinds
01243          INTEGER (SQLRETURN) :: SQLPutDataDP
01244          INTEGER (SQLHSTMT) :: stmt
01245          DOUBLE PRECISION rgbValue
01246          INTEGER (SQLINTEGER) :: cbValue
01249       END FUNCTION SQLPutDataDP
01250
01251    END INTERFACE
01252
01253    INTERFACE SQLRowCount
01254       FUNCTION SQLRowCount( stmt, pcrow )
01255          USE qt_ODBCKinds
```

```
01256          INTEGER (SQLRETURN) :: SQLRowCount
01257          INTEGER (SQLHSTMT) :: stmt
01258          INTEGER (SQLINTEGER) :: pcrow
01261       END FUNCTION SQLRowCount
01262    END INTERFACE
01263
01264    INTERFACE SQLSetConnectAttr
01265       FUNCTION SQLSetConnectAttrLP( dbc, Attribute, ValuePtr, StringLength )
01266          USE qt_ODBCKinds
01267          INTEGER (SQLRETURN) :: SQLSetConnectAttrLP
01268          INTEGER (SQLHDBC) :: dbc
01269          INTEGER (SQLINTEGER) :: Attribute
01270          INTEGER (SQLPOINTER) :: ValuePtr
01271          INTEGER (SQLINTEGER) :: StringLength
01273       END FUNCTION SQLSetConnectAttrLP
01274    END INTERFACE
01275
01276    INTERFACE SQLSetConnectAttrChar
01277       FUNCTION SQLSetConnectAttrChar( dbc, Attribute, ValuePtr, StringLength )
01278       ! ValuePtr is a zero terminated string
01279          USE qt_ODBCKinds
01280          INTEGER (SQLRETURN) :: SQLSetConnectAttrChar
01281          INTEGER (SQLHDBC) :: dbc
01282          INTEGER (SQLINTEGER) :: Attribute
01283          CHARACTER (LEN=*) :: ValuePtr
01284          INTEGER (SQLINTEGER) :: StringLength
01287       END FUNCTION SQLSetConnectAttrChar
01288    END INTERFACE
01289
01290    INTERFACE SQLSetConnectOption
01291       FUNCTION SQLSetConnectOption( dbc, fOption, vParam )
01292          USE qt_ODBCKinds
01293          INTEGER (SQLRETURN) :: SQLSetConnectOption
01294          INTEGER (SQLHDBC) :: dbc
01295          INTEGER (SQLUSMALLINT) :: fOption
01296          INTEGER (SQLUINTEGER) :: vParam
01298       END FUNCTION SQLSetConnectOption
01299    END INTERFACE
01300
01301    INTERFACE SQLSetCursorName
01302       FUNCTION SQLSetCursorName( stmt, szCursor, cbCursor )
01303          USE qt_ODBCKinds
01304          INTEGER (SQLRETURN) :: SQLSetCursorName
01305          INTEGER (SQLHSTMT) :: stmt
01306          CHARACTER (LEN=*) :: szCursor
01307          INTEGER (SQLSMALLINT) :: cbCursor
01310       END FUNCTION SQLSetCursorName
01311    END INTERFACE
01312
01313    INTERFACE SQLSetDescFieldChar
01314    ! ValuePtr is a CHARACTER buffer
01315       FUNCTION SQLSetDescFieldChar( DescriptorHandle, RecNumber, FieldIdentifier,  &
01316                                     ValuePtr, LenValuePtr )
01317          USE qt_ODBCKinds
01319          INTEGER (SQLRETURN) :: SQLSetDescFieldChar
01320          INTEGER (SQLHDESC) :: DescriptorHandle
01321          INTEGER (SQLSMALLINT) :: RecNumber, FieldIdentifier
01322          CHARACTER (LEN=*) :: ValuePtr
01324          INTEGER (SQLINTEGER) :: LenValuePtr
01325       END FUNCTION SQLSetDescFieldChar
01326    END INTERFACE
01327
01328    INTERFACE SQLSetDescField
01329    ! ValuePtr is a pointer
01330       FUNCTION SQLSetDescField( DescriptorHandle, RecNumber, FieldIdentifier,  &
01331                                 ValuePtr, LenValuePtr )
01332          USE qt_ODBCKinds
01334          INTEGER (SQLRETURN) :: SQLSetDescField
01335          INTEGER (SQLHDESC) :: DescriptorHandle
01336          INTEGER (SQLSMALLINT) :: RecNumber, FieldIdentifier
01337          INTEGER (SQLPOINTER) :: ValuePtr
01338          INTEGER (SQLINTEGER) :: LenValuePtr
01339       END FUNCTION SQLSetDescField
01340    END INTERFACE
01341
01342    INTERFACE SQLSetDescRec
01343       FUNCTION SQLSetDescRec( DescriptorHandle, RecNumber, ValType, SubType,   &
01344                               fldLength, PrecVal, ScaleVal, DataPtr,       &
01345                               StringLength, Indicator )
01346          USE qt_ODBCKinds
01348          INTEGER (SQLRETURN) :: SQLSetDescRec
01349          INTEGER (SQLHDESC) :: DescriptorHandle
01350          INTEGER (SQLSMALLINT) :: RecNumber, ValType, SubType, PrecVal, ScaleVal, NullablePtr
01351          INTEGER (SQLINTEGER) :: fldLength, StringLength, Indicator
01352          INTEGER (SQLPOINTER) :: DataPtr
01354       END FUNCTION SQLSetDescRec
01355    END INTERFACE
01356
01357    INTERFACE SQLSetEnvAttr
01358       FUNCTION SQLSetEnvAttrI4( env, Attribute, ValuePtr, StringLength )      ! corr. 12.10.2000:
SQLSetEnvAttr -> SQLSetEnvAttrI4
01358-1          ttr -> SQLSetEnvAttrI4
```

```
01359       ! ValuePtr is a 32-bit unsigned integer value
01360          USE qt_ODBCKinds
01361          INTEGER (SQLRETURN) :: SQLSetEnvAttrI4
01362          INTEGER (SQLHENV) :: env
01363          INTEGER (SQLINTEGER) :: Attribute
01364          INTEGER (SQLPOINTER) :: ValuePtr
01365          INTEGER (SQLINTEGER) :: StringLength
01367       END FUNCTION SQLSetEnvAttrI4
01368    END INTERFACE
01369
01370    INTERFACE SQLSetEnvAttrChar
01371       FUNCTION SQLSetEnvAttrChar( env, Attribute, ValuePtr, StringLength )
01372       ! ValuePtr is a zero terminated string
01373          USE qt_ODBCKinds
01374          INTEGER (SQLRETURN) :: SQLSetEnvAttrChar
01375          INTEGER (SQLHENV) :: env
01376          INTEGER (SQLINTEGER) :: Attribute
01377          CHARACTER (LEN=*) :: ValuePtr
01378          INTEGER (SQLINTEGER) :: StringLength
01381       END FUNCTION SQLSetEnvAttrChar
01382    END INTERFACE
01383
01384    INTERFACE
01385       FUNCTION SQLSetPos( stmt, irow, fOption, fLock )
01386          USE qt_ODBCKinds
01387          INTEGER (SQLRETURN) :: SQLSetPos
01388          INTEGER (SQLHSTMT) :: stmt
01389          INTEGER (SQLUSMALLINT) :: irow, fOption, fLock
01391       END FUNCTION SQLSetPos
01392    END INTERFACE
01393
01394    INTERFACE SQLSetScrollOptions
01395       FUNCTION SQLSetScrollOptions( stmt, fConcurrency, crowKeyset, crowRowset )
01396          USE qt_ODBCKinds
01397          INTEGER (SQLRETURN) :: SQLSetScrollOptions
01398          INTEGER (SQLHSTMT) :: stmt
01399          INTEGER (SQLUSMALLINT) :: fConcurrency, crowRowset
01400          INTEGER (SQLINTEGER) :: crowKeyset
01402       END FUNCTION SQLSetScrollOptions
01403    END INTERFACE
01404
01405    INTERFACE SQLSetStmtAttrChar
01406
01407    ! Value is a CHARACTER buffer
01408       FUNCTION SQLSetStmtAttrChar( stmt, Attribute, Value, LenValue )
01409          USE qt_ODBCKinds
01411          INTEGER (SQLRETURN) :: SQLSetStmtAttrChar
01412          INTEGER (SQLHSTMT) :: stmt
01413          INTEGER (SQLINTEGER) :: Attribute, LenValue
01414          CHARACTER (LEN=*) :: Value
01416       END FUNCTION SQLSetStmtAttrChar
01417    ! Value is an INTEGER*4
01418    END INTERFACE
01419
01420    INTERFACE SQLSetStmtAttrI4
01421       FUNCTION SQLSetStmtAttrI4( stmt, Attribute, Value, LenValue )
01422          USE qt_ODBCKinds
01424          INTEGER (SQLRETURN) :: SQLSetStmtAttrI4
01425          INTEGER (SQLHSTMT) :: stmt
01426          INTEGER (SQLINTEGER) :: Attribute, LenValue
01427          INTEGER*4 Value
01429       END FUNCTION SQLSetStmtAttrI4
01430    ! Value is a pointer to a buffer
01431    END INTERFACE
01432
01433    INTERFACE SQLSetStmtAttr
01434       FUNCTION SQLSetStmtAttr( stmt, Attribute, ValuePtr, LenValue )
01435          USE qt_ODBCKinds
01437          INTEGER (SQLRETURN) :: SQLSetStmtAttr
01438          INTEGER (SQLHSTMT) :: stmt
01439          INTEGER (SQLINTEGER) :: Attribute, LenValue
01440          INTEGER (SQLPOINTER) :: ValuePtr
01441       END FUNCTION SQLSetStmtAttr
01442    END INTERFACE
01443
01444    INTERFACE SQLSetStmtOption
01445       FUNCTION SQLSetStmtOption( stmt, fOption, vParam )
01446          USE qt_ODBCKinds
01447          INTEGER (SQLRETURN) :: SQLSetStmtOption
01448          INTEGER (SQLHSTMT) :: stmt
01449          INTEGER (SQLUSMALLINT) :: fOption
01450          INTEGER (SQLUINTEGER) :: vParam
01452       END FUNCTION SQLSetStmtOption
01453    END INTERFACE
01454
01455    INTERFACE SQLSpecialColumns
01456       FUNCTION SQLSpecialColumns( stmt, IdentifierType,  &
01457                                   CatalogName, CatNameLength,    &
01458                                   SchemaName, SchemaNameLength,  &
01459                                   TableName, TableNameLength,    &
01460                                   Scope, Nullable)
```

```
01461          USE qt_ODBCKinds
01462          INTEGER (SQLRETURN) :: SQLSpecialColumns
01464          INTEGER (SQLHSTMT) :: stmt
01465          CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName
01467          INTEGER (SQLSMALLINT) :: IdentifierType, CatNameLength, SchemaNameLength,  &
01468                                   TableNameLength, Scope, Nullable
01469       END FUNCTION SQLSpecialColumns
01470    END INTERFACE
01471
01472    INTERFACE SQLStatistics
01473       FUNCTION SQLStatistics( stmt, CatalogName, CatNameLength,    &
01474                                   SchemaName, SchemaNameLength,  &
01475                                   TableName, TableNameLength,    &
01476                                   Unique, Reserved )
01477          USE qt_ODBCKinds
01478          INTEGER (SQLRETURN) :: SQLStatistics
01480          INTEGER (SQLHSTMT) :: stmt
01481          CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName
01483          INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength, TableNameLength
01484          INTEGER (SQLUSMALLINT) :: Unique, Reserved
01485       END FUNCTION SQLStatistics
01486
01487    END INTERFACE
01488
01489    INTERFACE SQLTablePrivileges
01490       FUNCTION SQLTablePrivileges( stmt, CatalogName, CatNameLength,    &
01491                                       SchemaName, SchemaNameLength,  &
01492                                       TableName, TableNameLength )
01493          USE qt_ODBCKinds
01494          INTEGER (SQLRETURN) :: SQLTablePrivileges
01496          INTEGER (SQLHSTMT) :: stmt
01497          CHARACTER (LEN=*) :: CatalogName, SchemaName, TableName
01499          INTEGER (SQLSMALLINT) :: CatNameLength, SchemaNameLength, TableNameLength
01500       END FUNCTION SQLTablePrivileges
01501    END INTERFACE
01502
01503   INTERFACE
01504
01505       FUNCTION SQLTables( stmt, szTableQualifier, cbTableQualifier,          &
01506                                 szTableOwner, cbTableOwner,                  &
01507                                 szTableName, cbTableName, szTableType, cbTableType )
01508          USE qt_ODBCKinds
01509          INTEGER (SQLRETURN) :: SQLTables
01510          INTEGER (SQLHSTMT) :: stmt
01511          CHARACTER (LEN=*) :: szTableQualifier, szTableOwner,  &
01512                               szTableName, szTableType
01513          INTEGER (SQLSMALLINT) :: cbTableQualifier, cbTableOwner,    &
01514                                   cbTableName, cbTableType
01518       END FUNCTION SQLTables
01519
01520       ! added 14.10.2000, case: all pointer variables to be treated as Values (use LOC() function to specify
01520-1          a pointer to a variable)
01521       FUNCTION SQLTablesLP( stmt, szTableQualifier, cbTableQualifier,          &
01522                                   szTableOwner, cbTableOwner,                  &
01523                                   szTableName, cbTableName, szTableType, cbTableType )
01524          USE qt_ODBCKinds
01525          INTEGER (SQLRETURN) :: SQLTablesLP
01526          INTEGER (SQLHSTMT) :: stmt
01527          INTEGER (LP) :: szTableQualifier, szTableOwner,  &
01528                          szTableName, szTableType
01529          INTEGER (SQLSMALLINT) :: cbTableQualifier, cbTableOwner,    &
01530                                   cbTableName, cbTableType
01532       END FUNCTION SQLTablesLP
01533
01534    END INTERFACE
01535
01536    INTERFACE SQLTransact
01537       FUNCTION SQLTransact( env, dbc, fType )
01538          USE qt_ODBCKinds
01539          INTEGER (SQLRETURN) :: SQLTransact
01540          INTEGER (SQLHENV) :: env
01541          INTEGER (SQLHDBC) :: dbc
01542          INTEGER (SQLUSMALLINT) :: fType
01544       END FUNCTION SQLTransact
01545    END INTERFACE
01546
01547 END MODULE qt_ODBCInterfaces
01548 ! ------------------------------------
01549 ! (C) Jörg Kuthe, Germany, 1999-2007. All rights reserved. www.qtsoftware.de
01550 ! ====================================================================
```

Index

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z