Jörg Kuthe

ForDBC

Fortran Database Connectivity

Stand: 19. August 2014

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

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

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

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].

Man findet die Beschreibung der ODBC API (API = Application Programming Interface) in der aktuellen Form auch im Internet unter:

⇨ http://msdn.microsoft.com/library/

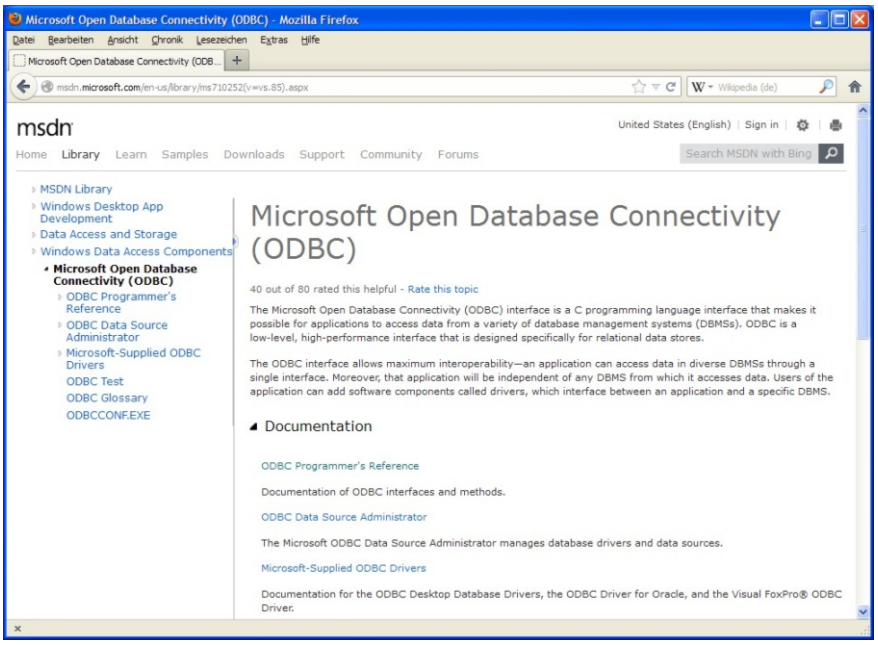

Abb. 1: Die Beschreibung der ODBC API im Internet.

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

- die wesentliche Funktionalität der ODBC Schnittstelle zu erläutern

und

- konkrete Hilfestellung für die ersten Schritte zur Erstellung von Programmen (.exe), die ODBC Funktionen verwenden (sog. ODBC Applikationen) zu geben.

Dies ist eine hilfreiche Erleichterung, da sich Microsofts Dokumentation im wesentlichen an C bzw. C++ Programmierer richtet und auch die ODBC spezifischen Datentypen, Funktionen und Konstanten im ODBC

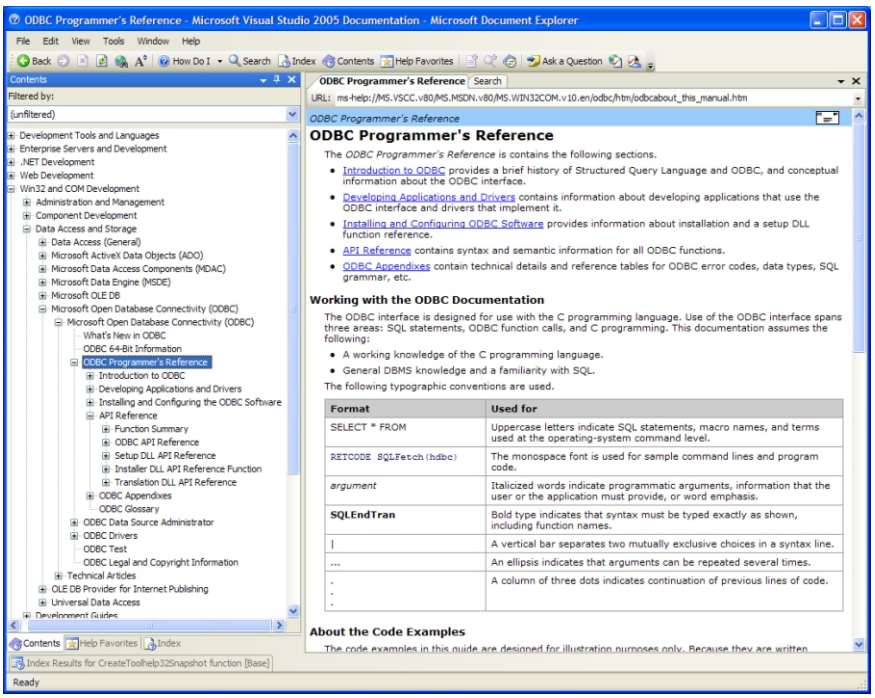

Abb. 2: Die Beschreibung ODBC API in der Online-Hilfe zu Visual Studio 2005.

Application Programming Interface (ODBC API) nur für die Programmierung in C bzw. C++ definiert sind.

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 95 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] bzw. im Internet 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).
Wenn eine gängige Datenbank auf Ihrem PC installiert ist (bspw. von Microsoft, IBM, Oracle, Sybase), dann ist meist auch ODBC Software installiert, so daß Sie sich mit den nachfolgend beschriebenen Einzelheiten erst später auseinandersetzen müssen; vor allem dann, wenn Ihr Programm keinen Zugriff auf die Datenbank erhält.

Zur Konfiguration von ODBC steht entweder ein ODBC Administrator Programm (z.B. ODBCAD32.EXE) oder ein applikations-spezifisches

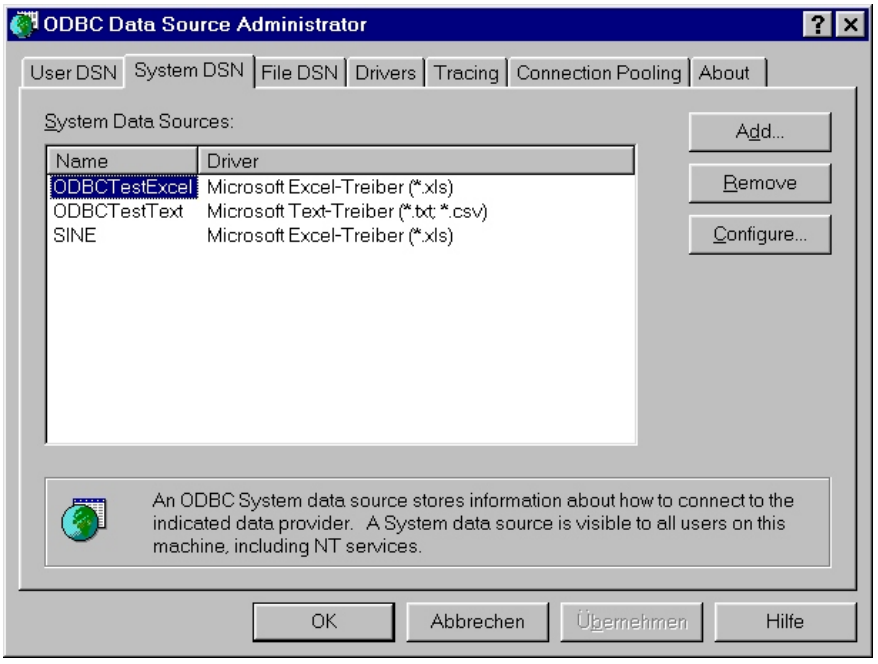

Abb. 3: ODBC Adminstrator Programm (odbcad32.exe)

Setup-Programm zur Verfügung. U.a. ist ein ODBC Administrator Programm in Microsofts 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. Für die meisten Fortran Programmierer, dürfte zunächst das Wissen um das Vorhandensein des ODBC Administrator Programms ausreichen (weiteres folgt unten).

Wesentlich ist, daß auf dem Zielrechner, auf dem eine ODBC Applikation unter Windows laufen soll, sowohl

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

vorhanden sein müssen. In den meisten Fällen muß man sich nicht um diese Dateien im Einzelnen kümmern, da sie bei Installation des bzw. der Treiber normalerweise korrekt kopiert werden.

Zur Verwendung der Datenbanken basierend auf MS/Access und MS/Excel, die in den ForDBC Beispielen benutzt werden, sind von Microsoft frei verfügbare Treiber erhältlich, die man installieren sollte, falls man nicht über MS/Office verfügt (welches diese Treiber normalerweise installiert):

Für 32-Bit Windows Applikationen sind dies die Microsoft Data Access Components (MDAC), die Sie auf Microsofts Website bspw. zum Zeitpunkt der Veröffentlichung dieses Dokuments unter

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

finden können (falls der Link nicht mehr aktuell sein sollte, hilft eine Suche nach MDAC, um zu der WebSeite zu gelangen). Von dort können Sie die MDAC laden (MDAC_TYP.EXE) und die Datei ausführen. Dabei werden die Treiber für Access und Excel installiert.

Für 64-Bit Windows Applikationen findet man die Treiber bspw. im "Microsoft Access Database Engine 2010 Redistributable", das unter

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

zum Download bereitsteht.

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) inklusive seiner Treiber und seine Rechnerplattform sowie ggf. das Netzwerk, über das auf die Daten zugegriffen wird, bezeichnet.
Um auf eine Datenquelle zugreifen zu können, benötigt das System bestimmte Informationen, um die Verbindung herzustellen. Dies sind mindestens (gemäß ODBC Core Level - siehe Abschnitt „Treiber“)

- der Name bzw. die Bezeichnung der Datenquelle (**Data Source Name**, kurz **DSN**)
- ggf. eine Benutzeridentifikation (User ID)
- und ggf. ein Paßwort.

ODBC Erweiterungen erlauben zusätzliche Angaben, bspw. der Netzwerkadresse oder weitere Paßwörter.
Die Datenquellenbezeichnung steht dann für die Verbindungsinformation. Für jede Datenquelle wurde früher in der ODBC.INI Datei und wird heutzutage in der Windows Registrierdatenbank (registry) gespeichert. Sie wird bei der Installation angelegt und in der Regel von einem eigenen Adminstrationsprogramm verwaltet (bspw. ODBCAD32.exe, s.o.). Ein Abschnitt in dieser ODBC.INI 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 derartige 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
```

Hier im Beispiel ist "Währungen" der Datenquellenname einer Datenbank (Dateiname: Waehrungen.DB), die auf "Sybase SQL Anywhere 5.0 (32 bit)" basiert.

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 (SQL Anywhere u.a.), Informix, IBM (DB2) oder Microsoft (Access, SQL) mitgeliefert. Darüberhinaus gibt es u.a. auch ODBC-Treiber für Excel, Text- oder dBASE-Dateien, die bspw. in Microsoft's ODBC Software Development Kit bzw. im MDAC (s.o.) 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. ODBCAD32.EXE) auf. Dies ist i.A. in der Systemsteuerung von Windows zu finden („Startmenu: Start | Einstellungen | Systemsteuerung"; unter Windows XP ist dort ein Eintrag "Datenquellen (ODBC)" unter „Verwaltung" zu finden). Für 64-Bit Windows sind mitunter zwei Versionen des ODBC Administrator Programms vorhanden, deren eine die 32-Bit Treiber und Datenquellen verwaltet und deren andere die 64-Bit Treiber und Datenquellen.

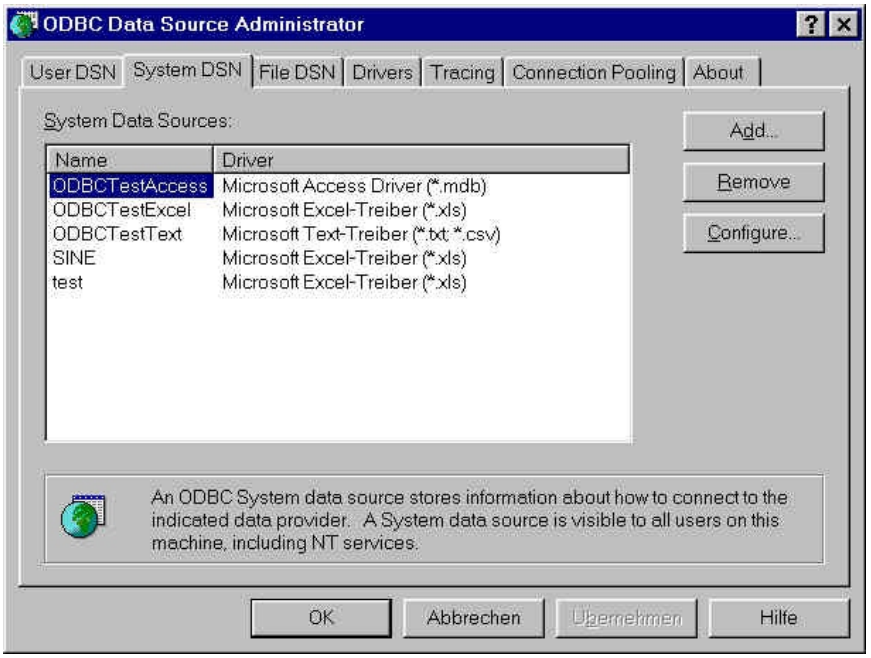

Abb. 4: Ansicht der Registerkarte „System DSN" nach Aufruf des ODBC Administrator-Programms.

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.

Datenquellen, die mithilfe des ODBC Administrator-Programms angelegt werden und auf die dann mittels DSN zugegriffen werden kann, sind "systemweit" nutzbar, also nicht nur durch Ihr Fortran Programm, sondern, wenn man das will, auch im Netzwerk. Der Vorteil der Verwendung des Zugriffs über DSN besteht in der freieren Konfigurierbarkeit der Datenquelle. Man kann bspw. zunächst mit einer durch eine DSN identifizierte lokalen Datenbank sein ODBC Fortran Programm entwickeln und später schlicht durch Änderung des Datenbankpfades in der Datenquelle zu einer anderen Datenbank wechseln, ohne das erstellte eigene Programm ändern zu müssen. Will man bspw. sein ODBC Programm mehreren Benutzern auf verschiedenen Rechnern im Netzwerk zur Verfügung stellen, so ist der Zugriff über DSN der flexibelste. Würde man in diesem Fall auf den festen Dateinamen der Datenbank, die sich auf irgendeinem Server im Netzwerk befindet, zugreifen, würde man Gefahr laufen bei Änderung oder Ausfall des Servers an seinem Programm mglw. Änderungen vornehmen und dies dann wieder an die Benutzer neu verteilen zu müssen.

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 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. Entsprechend sind die Excel spezifischen Testprogramme in ForDBC das funktionsfähige Ergebnis von zahlreichen Versuchen.

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.

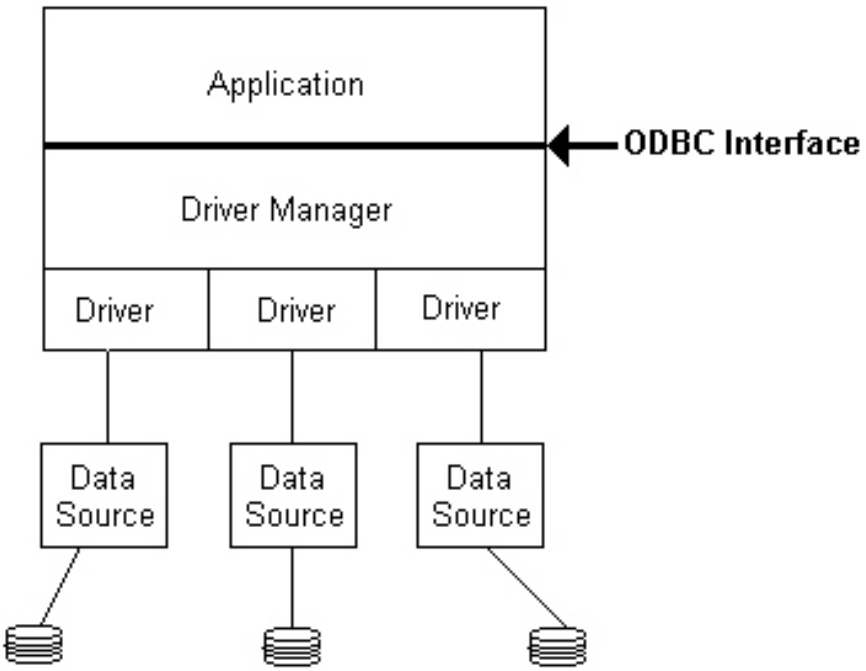

Abb. 5: Grundsätzlicher Aufbau von ODBC.

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 Spezifikationen. 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 bereitstellt. 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 (zuständige Funktion: `SQLAllocHandle`), indem in Ihrem Programm

- 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. Innerhalb einer aktiven Verbindung können eine oder mehrere Anweisungen (SQL statements) ausgeführt werden. Diese lassen sich zu einer Transaktion zusammenfassen (sofern der ODBC Treiber Transaktionen unterstützt; für MS/Access unter WindowsXP ist das bspw. nicht der Fall), und sie werden innerhalb eines

Transaktionsrahmens ausgeführt. Zu jeder Verbindung gehört ein eigener Transaktionsrahmen (transaction space). Eine Transaktion wird mit dem COMMIT-Befehl ausgelöst, d.h. erst dann werden die SQL Anweisungen tatsächlich auf die Datenbank angewendet.

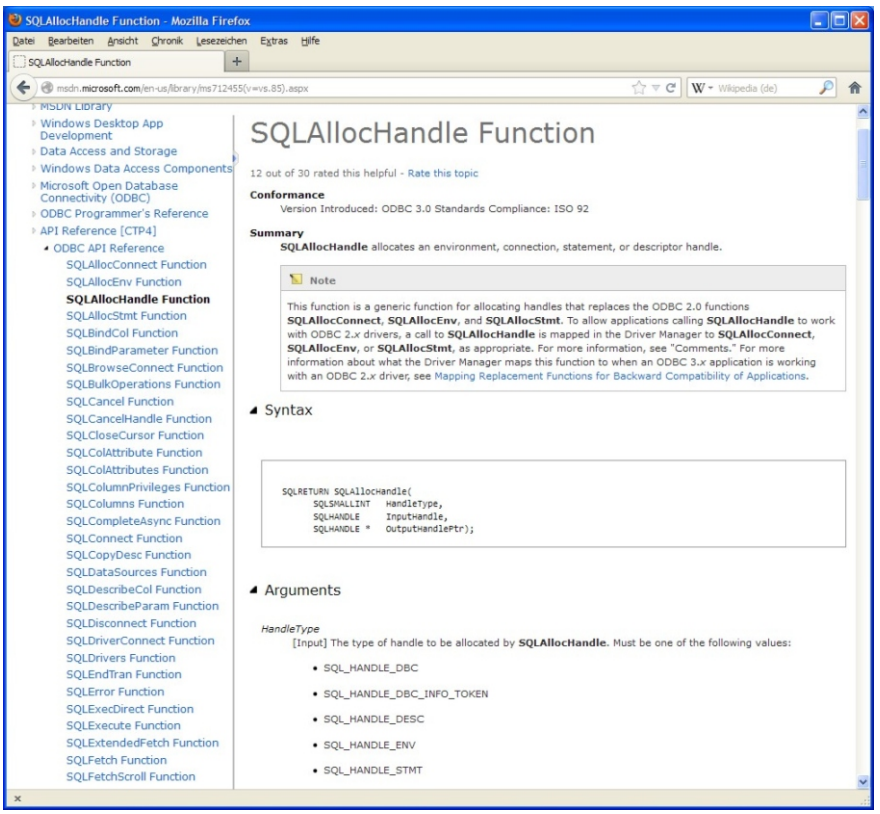

Abb. 6: Die Beschreibung der ODBC Funktion SQLAllocHandle

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 Funktionen in Fortran

Die Namen aller Funktionen der ODBC API (API = Application Programming Interface) **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 in ForDBC entsprechende Fortran Module bereitgestellt, die in ODBC Applikationen mittels USE Befehl einzubinden sind:

```
USE ForDBC
```

Das Modul befindet sich in der Datei

```
fordbc.mod
```

Das Modul ForDBC beinhaltet Referenzen zu weiteren Modulen, faßt also alle anderen, nachfolgend genannten Module zusammen, deren explizite Angabe mittels `USE` in Ihrem Fortran Programm damit nicht notwendig ist.

So erfolgt z.B. die Definition der ODBC spezifischen Datentypen (KINDs) in

```
qtODBCKinds
(Datei qtodbckinds.mod)
```

und der Konstanten (PARAMETERs) in

```
qtODBCDefs
(Datei qtodbcdefs.mod)
```

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

```
qtCKinds
(Datei qtckinds.mod)
```

und

```
qtODBCCoreKinds
(Datei qtodbccorekinds.mod)
```

definiert werden.

So definiert bspw. das Modul `qtCKinds` den "Datentyp" LONG,

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

der damit mit einem 4-Byte INTEGER (INTEGER*4) identisch ist, und das Modul `qtODBCCoreKinds` definiert den Datentyp LP, der für 32-Bit und 64-Bit Systeme unterschiedlich ist (woraus mancher Leser erahnen mag, warum KINDs verwendet werden).

In den Modulen `qtODBCKinds` und `qtODBC` 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 = LP
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 Typ SQLHANDLE in einer 32-Bit Umgebung 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, kann durch Änderung der zugrundeliegenden Definitionen (bspw. in `qtCKinds`) leicht eine komplette Änderung der abgeleiteten Datentypen erfolgen (dieser Fall trat bspw. beim Umstieg von Win16 zu Win32 und von Win32 zu Win64 auf). So stellt der Typ SQLHANDLE in einer 64-Bit Umgebung dann einen 8-Byte INTEGER dar.
Aus Gründen der Referenz zur Originaldokumentation der ODBC Schnittstelle wurde also versucht, eine Analogie in den Fortran 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 ForDBC
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.. Aufgrund der Definition von "generic interfaces" könnte der Fortran Programmierer aber auch grundsätzlich die "generice" Variante, also hier im Beispiel `SQLBindCol` verwenden. Leider gilt dies jedoch nicht für alle unterstützten Fortran Compiler, da für manche die Bereitstellung von "generic interfaces" nicht möglich war.

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

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

```
                                ValPtr, 0 )
.
END
```

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() bestimmt werden (bei manchen Compilern auch mit der ehemaligen VAX Funktion POINTER(), die nicht mit dem Fortran 90 POINTER Befehl zu verwechseln ist). Ob ein Pointer oder ein Wert zu übergeben ist, ist der jeweiligen Funktionsbeschreibung in der ODBC API bzw. dem Anhang A mit der ForDBC Funktionsübersicht zu entnehmen.

Die Funktionsschnittstellen (INTERFACEs) sind im Modul

```
qtODBCInterfaces
(Datei: qtODBCInterfaces.mod)
```

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

```
qtODBC_Compiler
(Datei: qtODBC_compiler.mod
mit compiler = CVF, 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 des Quellcode Ihrer ODBC Applikationen i.A. nicht nötig.

Der ODBC 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 die zugehörige Importbibliothek (ODBC32.LIB).** Sie ist sowohl für 32-Bit Systeme als auch für 64-Bit Systeme verfügbar. Beide Varianten tragen den gleichen Namen. Entsprechend ist beim Binden darauf zu achten, daß der Verzeichnisname der ODBC32.LIB korrekt angegeben wird (so er denn verwendet wird).
Der Treiber Manager verwaltet und leitet den Zugriff auf den Treiber und somit die Verbindung zu der dem Treiber zugehörigen .DLL.

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 (pointer) 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 ODBC Funktion 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.
- **ZurückgegebeneZeichenketten (CHARACTER) sind immer null-terminiert.** Es findet kein Blank-Padding statt (d.h. CHARACTER Variablen werden nicht mit Leerzeichen aufgefüllt, wie das sonst in Fortran üblich ist).

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 (es gibt keine Variablen ohne Wert). Man behilft sich oft durch die Verwendung eines bestimmten Variablenwerts, der den Zustand des "fehlenden Werts" markiert. Z.B. wird ein Wert zu -999999.9 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 in der Datenbanktabelle entspricht ein bei der Programmierung zu verwendender 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 (statement) 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 oder SQLHANDLE (ODBC v3.x) - alle 3 Typen basieren übrigens auf dem KIND INTEGER(LP) 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 oder SQLHANDLE (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 oder SQLHANDLE (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 qtODBCDefs 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

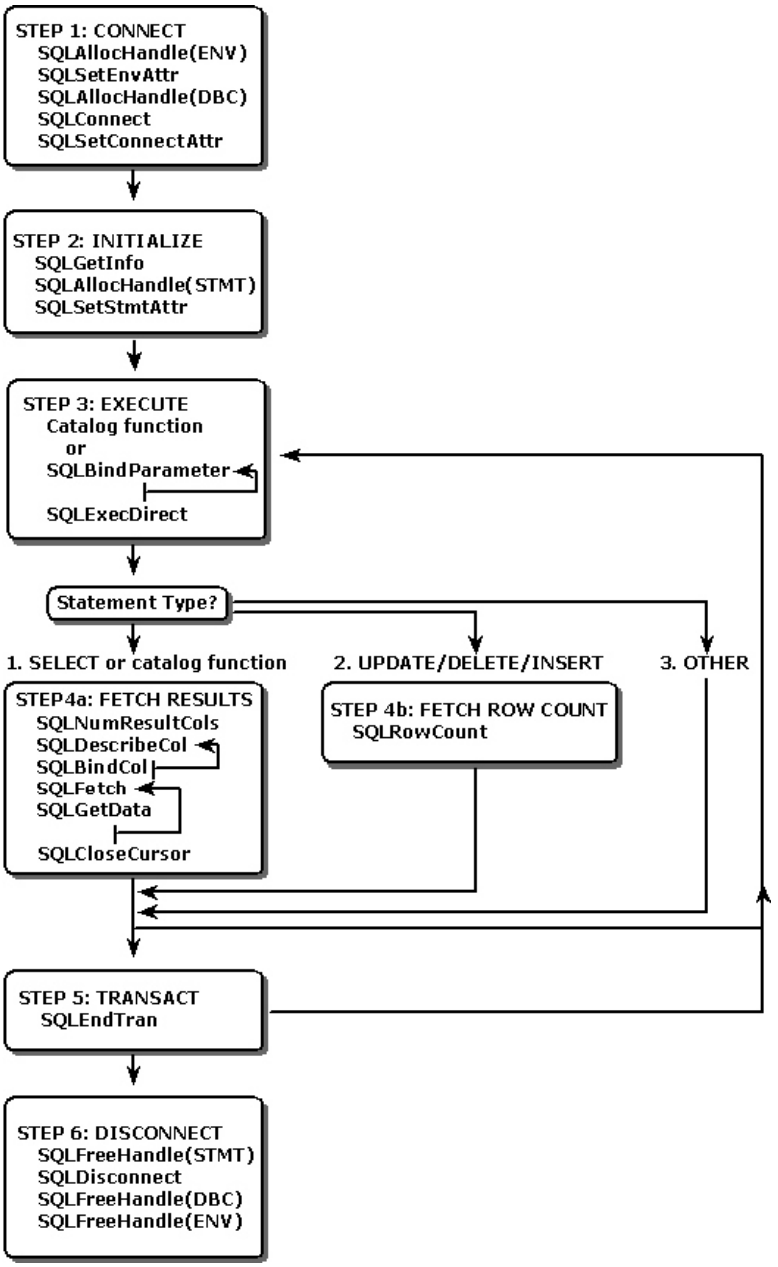

Abb. 7: ODBC Programmstruktur (ODBC v3.x), nach [ODBC08]

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.

Das Diagramm oben 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. Nachfolgend werden die Schritte für ODBC v1.x/v2.x gezeigt.

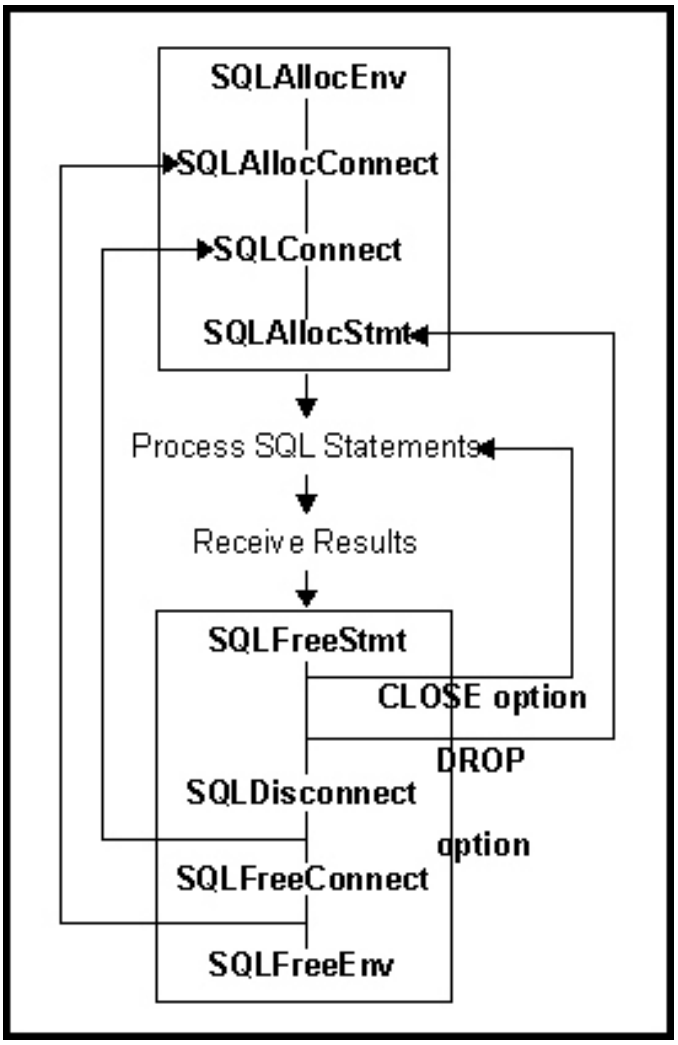

Abb. 8: ODBC Programmstruktur (ODBC v1.x/v2.x)

Zu bevorzugen ist die Programmstruktur für ODBC v3.x, da die meisten der im letzterem Diagramm gezeigten Funktionen bereits als veraltet ("deprecated") markiert sind.

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=SQLHANDLE) :: env = SQL_NULL_HANDLE
INTEGER rtc
```

erfolgt der Aufruf zur Erzeugung der ODBC Umgebung:

```
rtc = SQLAllocHandle(SQL_HANDLE_ENV,  &
                     SQL_NULL_HANDLE, env )
```

Die Funktion `SQLAllocHandle` gibt im Erfolgsfall (rtc = SQL_SUCCESS) die Umgebungsidentifikationsnummer im Argument `env` zurück.

Die Initialisierung einer ODBC Umgebung darf in einer Applikation nur dann mehrmals stattfinden, nachdem die jeweils zuvor geöffnete ODBC Umgebung geschlossen wurde (mittels `SQLFreeHandle`).

5.5.2 Der Verbindungsaufbau

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

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

Für die Verbindung wird die Funktion `SQLAllocHandle` aufgerufen, die als Argument die zuvor angelegte Umgebungsidentifikationsnummer (environment handle) benötigt.

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

Falls kein Fehler auftritt (rtc = SQL_SUCCESS), wird im letzten Argument der Funktion die Verbindungsidentifikationsnummer zurückgegeben.

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 (registry) 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] bzw. den aktuellen ODBC Beschreibungen 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.

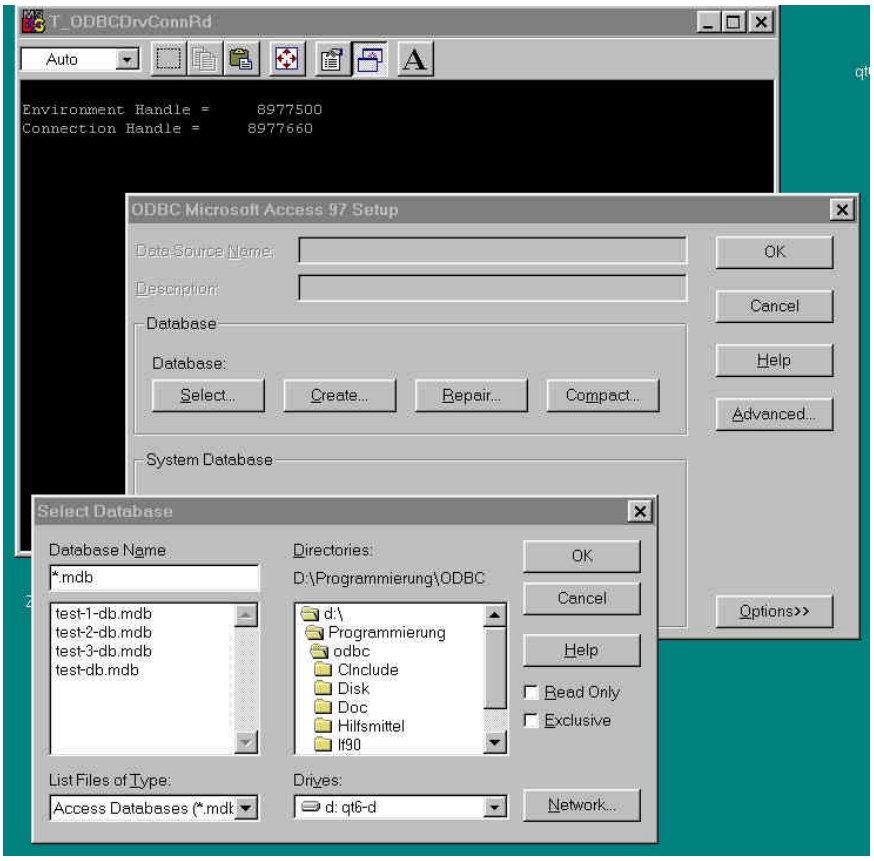

Abb.9: Auswahl der Datenquelle während der Laufzeit - vgl. Beispielprogramm T_ODBCDrvConnRd.f90.

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 SQLHANDLE und kann bspw. wie folgt deklariert werden.

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

Sie wird dann durch den Aufruf von `SQLAllocHandle` erzeugt und benötigt die zuvor generierte Verbindungsidentifikationsnummer (im Beispiel: dbc).

```
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 der Funktion 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.

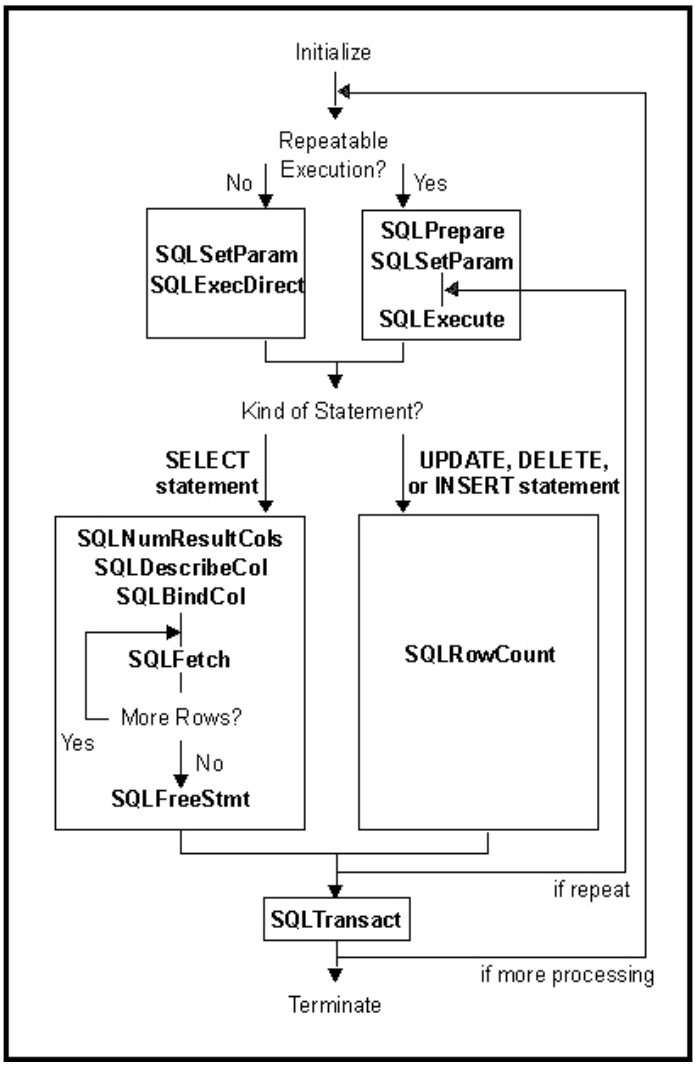

Abb. 10: Programmstruktur zur Ausführung von ODBC Befehlen

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, ggf. die Genauigkeit, Länge und Dezimalbereich an und assoziiert die Spalte mit dem Parameter. Danach kann durch Zuweisung des gewünschten Wertes an diesen Speicherplatz (d.h. der Variablen) der Parameter gesetzt werden. Z.B.:

```
CHARACTER szName*30
INTEGER (SQLULEN) :: LENszName = 30
INTEGER (SQLLEN) :: cbName
...
rtc = SQLPrepare( stmt,   &
            "INSERT INTO adressbuch (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 )
...
szName = 'Mustermann'; cbName = SQLNTSL;
rtc = SQLExecute( stmt )
! INSERT des Datensatz ("'Mustermann',...")
```

Erläuterung: An die erste Spalte (zweites Argument = 1) vom Typ (SQL_CHAR) der Tabelle "adressbuch" 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 Werts vorgegeben. Die Angabe der Puffergröße (vorletztes Argument) für den Textpuffer (szName) ist hier nicht notwendig (und ist 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 (sofern der Treiber diese Konvertierung unterstützt).

Die Zuordnungen der Speicherplätze (d.h. C/Fortran Variablen zu ODBC/SQL Input Parameter) bleiben solange erhalten, bis die Funktion `SQLFreeHandle(SQL_HANDLE_STMT, ...)` aufgerufen wird. 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.

In den ForDBC Beispielen wird nur der Auto-Commit-Modus verwandt, da weder MS/Access noch MS/Excel das Transaktionsprinzip unterstützten.

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 Ihrem Programm deklarierte 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_NTSL )
rtc = SQLBindCol( stmt, 1, SQL_C_CHAR, wName,  &
                  LENwName, cbwName)
```

Erläuterung: Der erste Spaltenname (zweites Argument von `SQLBindCol` ist =1) des SELECT Befehls wird an den Speicherplatz der Variable wName vom Typ SQL_C_CHAR der Länge LENwName gebunden. Wenn der SELECT Befehl erfolgreich ausgeführt wird, wird *später* (genauer beim Aufruf von `SQLFetch`) 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.

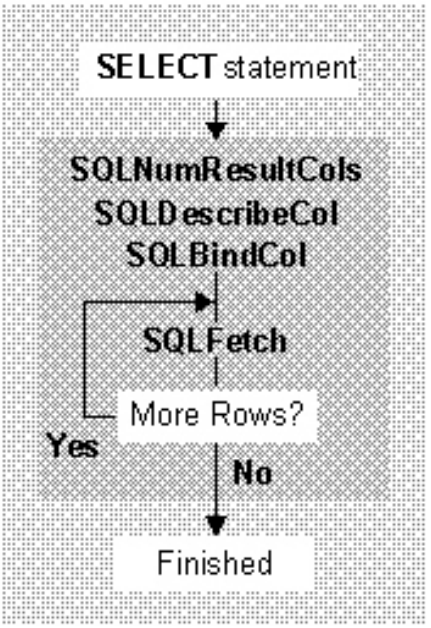

Abb. 11: Abholen von Ergebnissen

Beispiel:

```
rtc = SQLExecDirect( stmt,   &
     "SELECT waehrungname FROM waehrungen"//CHAR(0),  &
     SQL_NTS )
rtc = SQLBindCol(  stmt, 1, SQL_C_CHAR,   &
                   wName, LENwName, cbwName )
DO
   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 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 oder hinter 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 drei inzwischen als veraltet gekennzeichneten Funktionen SQLFreeStmt, SQLFreeConnect und SQLFreeEnv oder die nun zu benutzende Funktion SQLFreeHandle (ab ODBC v3.0).

- SQLFreeStmt (ODBC v1/v2) 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.

SQLFreeHandle(SQL_HANDLE_STMT, ...) (ODBC v3) faßt die verschiedenen Funktionen von SQLFreeStmt in einem Aufruf zusammen (die Verwendung von SQLFreeStmt ist damit hinfällig).

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

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

Anschließend ist die Funktion

- `SQLFreeConnect` (ODBC v1/v2) oder die neuere Funktion `SQLFreeHandle( SQL_HANDLE_DBC, ... )` aufzurufen, die die Ressourcen der Verbindung und die Verbindungsidentifikationsnummer (connection handle) freigeben.

Zuletzt bleibt noch mittels

- `SQLFreeEnv` oder `SQLFreeHandle( SQL_HANDLE_ENV, ... )` 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 (nur für 32-Bit Excel ODBC Treiber).
- 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, weshalb vorherige Aufzählung auch nicht vollständig sein mag, da diese Hinweise mehr auf Erfahrungen im Gebrauch mit Excel ODBC fußen, als aus der Dokumentation zu den EXCEL ODBC Treibern..

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 (.zip Datei oder selbst-extrahierende .exe Datei) zur Verfügung.

Installation von CD-ROM

Im Stammverzeichnis der CD-ROM befindet sich ein komplettes ForDBC Verzeichnis, das Sie an eine beliebige Stelle Ihrer Festplatte kopieren. Sie können ForDBC dann sofort nutzen.

Installation mit komprimiertem oder selbst-extrahierendem Archiv

Wenn Sie ForDBC als komprimiertes Archiv (.zip Datei) oder als selbst-extrahierende Datei (.exe) erhalten, wählen Sie einen beliebigen Ort auf Ihrer Festplatte und entpacken dort das ForDBC Archiv. ForDBC kann dann sofort genutzt werden.

6.1 ForDBC Software und Testprogramme

Um ForDBC schnell und effizient einsetzen zu können, machen Sie sich bitte mit dem Inhalt des ForDBC Verzeichnisses vertraut und probieren die Testprogramme aus.

6.1.1 ForDBC Module

ForDBC besteht aus diversen compilierten Fortran 90 Modulen, z.B.

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

die in den Compiler-spezifischen Verzeichnissen

```
...\ForDBC\Bindings\CVF
```

bzw.

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

bzw.

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

zu finden sind ("..." kürzt das von ihnen gewählte übergeordnete Verzeichnis ab, in dem Sie ForDBC entpackt haben). Die Kürzel CVF und IVF stehen für:
CVF = Compaq Visual Fortran
IVF = Intel Visual Fortran

Wie leicht zu erraten ist, sind für den IVF zwei Varianten vorhanden, eine für 32-Bit Applikationen, die andere für 64-Bit Applikationen.

Auf diese compilerspezifischen Moduldateien muß Ihr Compiler beim Übersetzen Ihres ODBC Programms, welches das Module

ForDBC verwendet, zugreifen können. Sowohl CVF als auch IVF erlauben dazu den Modulpfad anzugeben (siehe unten Kapitel "Compilerspezifische Anmerkungen").

6.1.2 ForDBC Beispielprogramme

Die Beispielprojekte im CVF Workspace

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

bzw. in der Visual Studio Solution

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

zeigen dies in den Einstellungen ("Settings" bzw. "Properties") für den Fortran Compiler. Die Visual Studio 2008 Solution ForDBCExamples.sln kann auch von Visual Studio 2010 und evt. auch 2012 (nicht getestet) geladen werden.

Die Beispielprojekte enthalten Testprogramme, die die Verwendung von ForDBC demonstrieren:

T_ODBCDrivers.f90	listet ODBC Treiber
T_ODBCDataSources.f90	listet ODBC Datenquellen
T_ODBCAccessInfo.f90	informiert über Datenquelle test-db.mdb
T_ODBCAccessGetRows.f90	ermittelt die Anzahl der Sätze in der Tabelle "Tabelle" der Datenquelle test-db.mdb
T_ODBCAccessRd.f90	liest Datenquelle test-db.mdb
T_ODBCExcelGetRows.f90	ermittelt die Anzahl der Sätze in der Tabelle "Table_1" der Datenquelle ODBCTestExcel.xls
T_ODBCDrvConnRd.f90	liest Access- oder Excel-Dateien (test-db.mdb bzw. TestODBCDrvConnRd.xls, die während der Laufzeit auszuwählen sind)
T_ODBCAccessWr.f90	schreibt in Datenquelle test-db.mdb
T_ODBCExcelRd.f90	liest Datenquelle ODBCTestExcel.xls (zeigt auch Tabelleninformation, wie Spaltennamen etc. an)
T_ODBCExcelWr.f90	schreibt in Datei TestODBCExcelWr.xls (die Datei ist während der Laufzeit auszuwählen)

Diese Fortran Quellcodedateien befinden sich alle im Verzeichnis

...\ForDBC\Examples

Bei der Durcharbeitung dieser Beispiele empfiehlt es sich, diese mithilfe des Debuggers durchzugehen, um den Ablauf im Einzelnen zu verstehen und in Relation zu der in den vorherigen Kapiteln dargelegten "Theorie" zu setzen. Obige Liste der Testprogramme gibt in etwa die Reihenfolge wieder, in der sich die Abarbeitung aus didaktischer Sicht empfiehlt. Falls Sie ForDBC nicht für den Zugriff auf Excel verwenden wollen, genügt es, sich mit den Beispielen für die Access-Datenbank auseinanderzusetzen. Für alle anderen relationalen Datenbanken mit ODBC Treibern wird die

Programmierung sehr ähnlich sein. Die meisten Schwierigkeiten hat man anfangs mit dem sog. Connect. Hier hilft es, die Kommentare in den Beispielprogrammen aufmerksam zu lesen (insbesondere in T_ODBCAccessGetRows.f90). Sollte es Probleme beim Column- bzw. Parameter-Binding geben, so hilft oft mittels der "Kataloginformation" (siehe auch `SQLColumns`) herauszufinden, mit welchen Tabellen, SQL Typen etc. man es zu tun hat (vgl. T_ODBCAccessWr.f90).

6.1.3 ODBC32.LIB und compilerspezifische ForDBC Bibliothek

Die ForDBC Beispielprogramme sind zu compilieren und **in Abhängigkeit vom verwendeten Fortran Compiler/Linker mit der ODBC32.DLL oder mit der entsprechenden Import-Library ODBC32.LIB und der compilerspezifischen ForDBC Bibliothek zu binden**.
Die ODBC32.LIB ist nicht Bestandteil der Lieferung von ForDBC, sondern eine von Microsoft bereitgestellte Windows Systembibliothek - und es sollte dann genügen, lediglich ihren Namen dem Linker als zusätzliche Library ohne Verzeichnisangabe bekanntzugeben (mitunter wird sie aber auch automatisch gefunden). Wenn Sie die ODBC32.LIB explizit als Datei Ihrem CVF oder IVF Projekt hinzufügen wollen, so finden Sie sie bspw., wenn Sie den CVF besitzen, vermutlich im Verzeichnis

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

Besitzer von Visual Studio 2008 und IVF werden vermutlich in

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

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

fündig. Der Suchdienst des Windows File Explorer kann Ihnen hier ggf. helfen, die ODBC32.LIB auf Ihrem PC aufzuspüren.

6.1.4 Erstellen der Testprogramme (.exe) in Entwicklungsumgebungen

Wenn Sie die Testprogramme in einer der Entwicklungsumgebungen Developer Studio oder Visual Studio compilieren und binden, finden Sie eine Datei namens

`ForDBCHints.txt`

im Verzeichnis

...\ForDBC\Examples

mit Anweisungen, welche Einstellungen für die jeweiligen Compiler vorzunehmen sind.

Mit IVF

Für IVF laden Sie bitte in die Entwicklungsumgebung (Visual Studio) die Datei

```
ForDBCExamples.sln
```

(für Visual Studio 2008), die im Unterverzeichnis

...\Examples\IVF\IVF_VS08

zu finden ist. Führen Sie dann "Build Solution" aus, um alle Projekte der "Solution" zu "bauen".

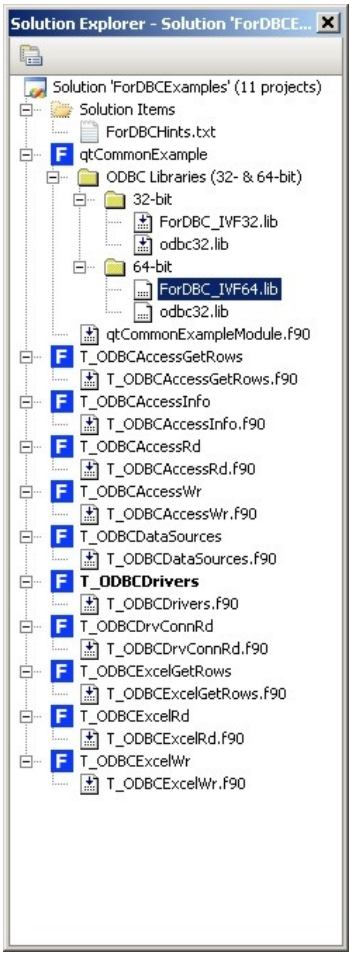

Abb. 12: ForDBC Beispielprojekte in Visual Studio

Mit CVF

Für CVF steht eine Workspace-Datei

```
ForDBCExamples.dsw
```

bereit, im Unterverzeichnis

...\Examples\CVF

die alle Beispielprojekte (.dsp) bündelt. Ein "Batch Build" in der Entwicklungsumgebung (Developer Studio) erzeugt alle Testprogramme.

Einzelheiten zu den compiler- und linkerspezifischen Einstellungen finden Sie später im Kapitel "Compilerspezifische Anmerkungen".

Datenquellen (DSNs)

Einige der ForDBC Testprogramme sind erst nach Anlegen der Test-Datenquellen fehlerfrei lauffähig. Bitte lesen Sie dazu das nachfolgende Kapitel "Anlegen der Test-Datenquellen".

6.1.5 Addendum

Im ForDBC Verzeichnis ist noch eine Datei

```
Addendum.txt
```

zu finden, die Hinweise auf ältere Versionen und Nachrichten enthält, die erst nach Veröffentlichung dieses Dokuments erschienen.

6.2 Anlegen der Test-Datenquellen

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

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 für 32-Bit Applikationen; für 64-Bit Applikationen ist bspw. der Treiber für Excel 12.0 zu verwenden

ODBCTestAccess für die Datei test-db.mdb mit MS Access Treiber

zu definieren. Erst dann können die meisten der o.g. Testprogramme ausgeführt werden. Man beachte, daß der ODBC Administrator auf 64-Bit Windows mitunter als 32- und als 64-Bit Applikation vorhanden ist. Die Datenquellen sind entsprechend dann in beiden anzulegen (mit den jeweiligen Treibern).

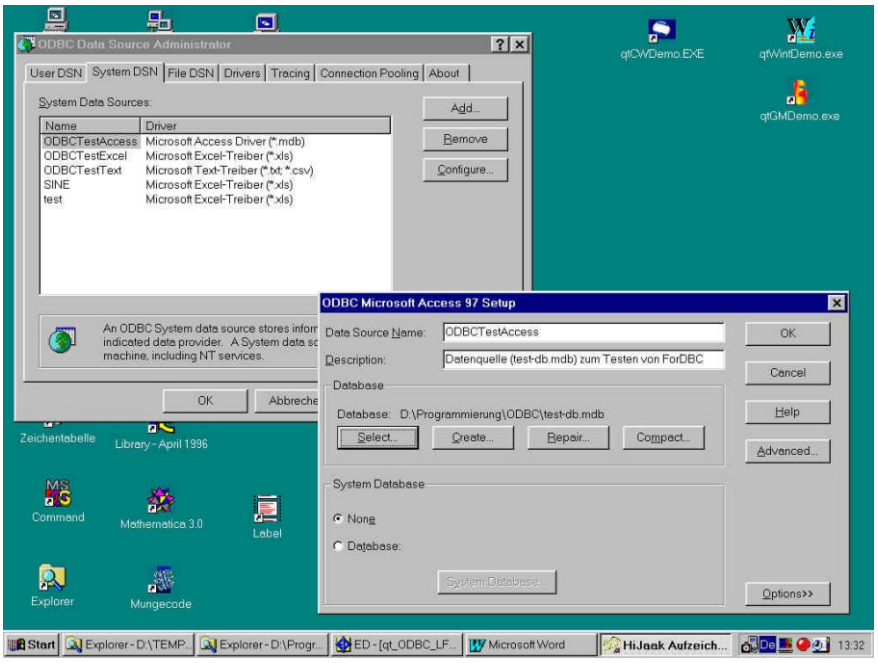

Abb. 13: Anlegen einer Datenquelle mit dem ODBC Administrator.

Die anderen beiden Excel-Dateien

TestODBCDrvConnRd.xls

und

TestODBCExcelWr.xls

brauchen nicht als Datenquellen angelegt zu werden.

6.3 Compilerspezifische Anmerkungen

6.3.1 Compaq Visual Fortran (CVF)

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>; vgl. nachfolgende Abb.). Oder generell für CVF über Tools | Options, über Directories unter "Include files": <Modulpfad hinzufügen>.

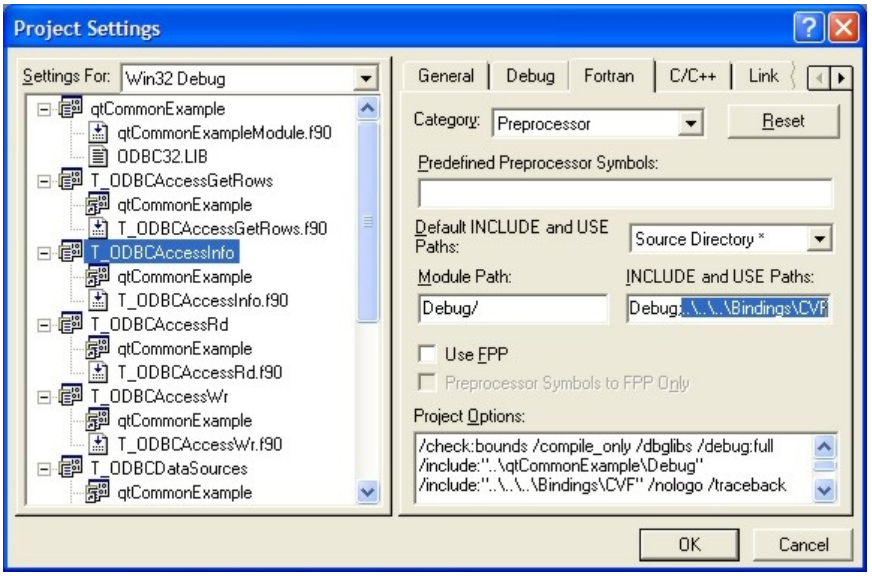

Abb. 14: Angabe des Modulpfads in CVF

Beim Binden (Link) ist darauf zu achten, daß die

```
ForDBC_CVF.LIB
```

und die

```
ODBC32.LIB
```

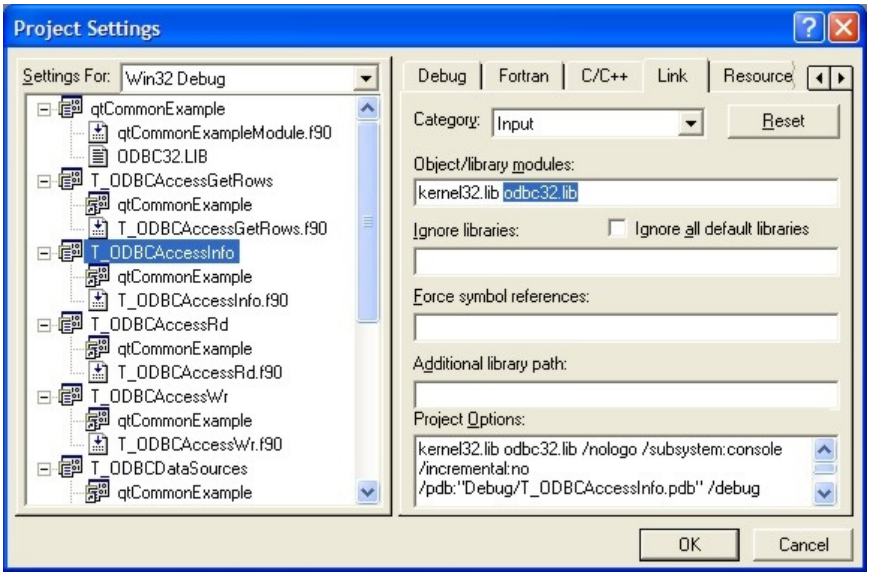

Abb. 15: Angabe der ODBC32.LIB in CVF

dazugebunden werden (Project Settings, für „All Configurations“ | Registerkarte Link, Eingabefeld "Object/library modules": kernel32.lib odbc32.lib; vgl. nachfolgende Abb.) - oder die beiden Dateien dem Projekt explizit hinzufügen. Letzteres empfiehlt sich für die ForDBC_CVF.LIB, da man sonst in den Settings noch den Pfad dieser Bibliothek anzugeben hätte.

6.3.2 Intel Visual Fortran (IVF)

Auch hier gilt: Wenn Sie ForDBC in Visual Studio verwenden wollen, müssen die Module im Modulpfad zu finden sein. Ggf. geben Sie diesen in den Einstellungen (Properties) des Projekts an (Configuration Properties|Fortran, Preprocessor, Eingabefeld "Additional Include Directories": <Modulpfad angeben>; vgl. Abbildung).

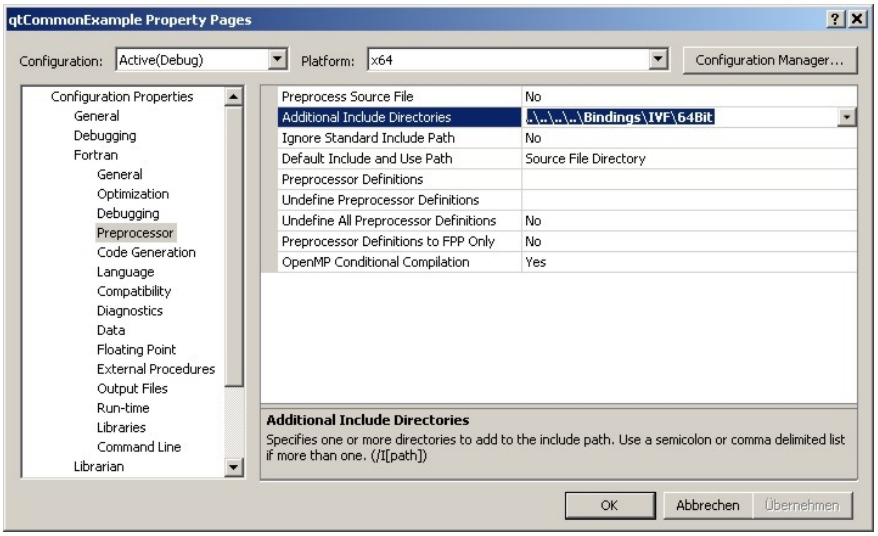

Abb. 16: Angabe des Modulpfads in IVF

Oder generell für IVF über Tools | Options: Intel(R) Fortran|Compilers, Eingabefeld "Includes": <Modulpfad hinzufügen>.

Beim Binden (Link) ist darauf zu achten, daß die

```
ForDBC_IVF32.LIB oder ForDBC_IVF64.LIB
```

(für 32-Bit oder 64-Bit Programme) und die passende 32- oder 64-Bit Variante der

```
ODBC32.LIB
```

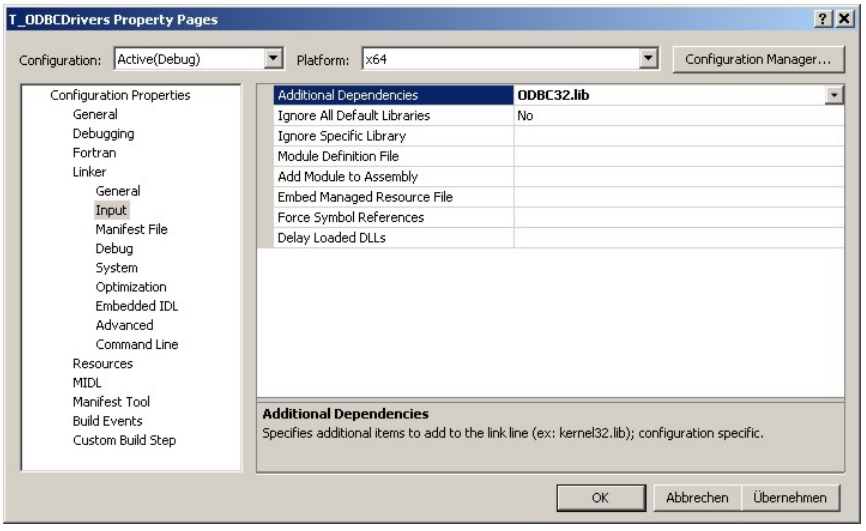

Abb. 17: Angabe der ODBC32.LIB in IVF

sowie möglicherweise die

```
advapi32.lib
```

dazugebunden werden (Properties, Configuration Properties|Linker|, Eingabefeld "Input", z.B.: odbc32.lib fordbc_IVF32.lib advapi32.lib) - oder die Libraries Dateien dem Projekt explizit hinzugefügt werden. Die advapi32.lib ist ab v3.25 entbehrlich. Letzteres empfiehlt sich aber nur für die ForDBC_IVF32.LIB bzw. ForDBC_IVF64.LIB, da man sonst in den Settings noch den Pfad der Bibliothek anzugeben hätte. Die beiden anderen Systembibliotheken (odbc32.lib advapi32.lib) sollte der Linker selbst finden.
In den Beispielprogrammen werden die beiden Systembibliotheken über die Additional Options der "Command Line" des Librarian für das Library-Projekt "qtCommonExample" spezifiziert.

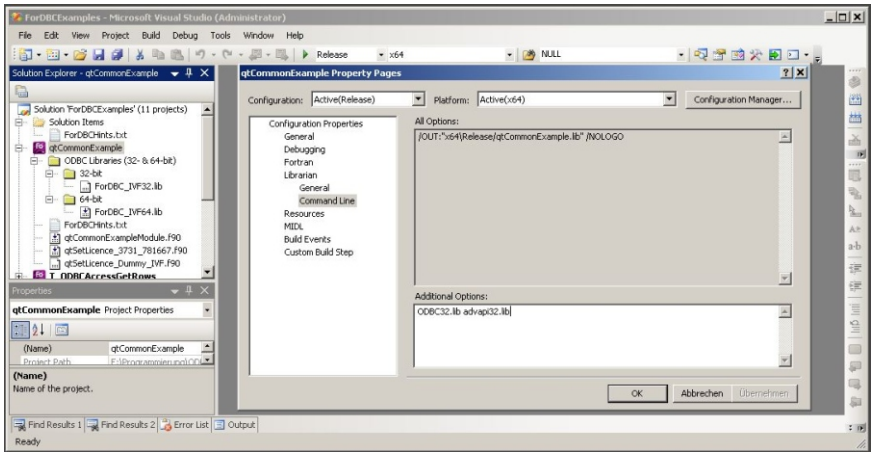

Abb. 18: Angabe der odbc32.lib und advapi32.lib für den "Librarian" in den Additional Options der "Command Line" (Projekt "qtCommonExample") Die Angabe der advapi32.lib ist ab v3.25 entbehrlich.

6.3.2.1 Initialisierung und Lizensierung von ForDBC - qtSetLicence_####_######.f90

Ab der ForDBC Version 3.25 ist eine Initialisierung von ForDBC nötig. Dazu ruft man die Funktion `ForDBC_Initialize` mit der erhaltenen Lizenznummer (Format: L0889-###### für 32-Bit Programme, L3731-##### für 64-Bit Programme; mit # = 0, 1, 2, ..., 8 oder 9) auf. Z.B.

```
! für 32-Bit Programme
iRet = ForDBC_Initialize('L0889-123456')
```

Wenn die Lizenz gefunden wurde und gültig ist, liefert die Funktion den Wert 0 (iRet = 0) zurück (ansonsten einen Wert /= 0). Wenn man noch keine Lizenz besitzt, kann man ForDBC auch im Demonstrations-Modus betreiben:

```
! Betrieb im Demo-Modus
iRet = ForDBC_Initialize('evaluation')
```

In `ForDBC_Initialize` wird die Routine `qtSetLicence_ForDBC` (im 32-Bit-Fall) bzw. `qtSetLicence_ForDBC64` (im 64-Bit-Fall) gerufen. Diese Routine wird als "Dummy" zum Testen (für den Demo-Modus) bereitgestellt:

```
qtSetLicence_Dummy_IVF.f90
```

Wenn Sie eine ForDBC-Lizenz erworben haben, ersetzen Sie sie durch die erhaltene Lizenzroutine

```
qtSetLicence_####_######.f90
(# = eine Ziffer 0 bis 9)
```

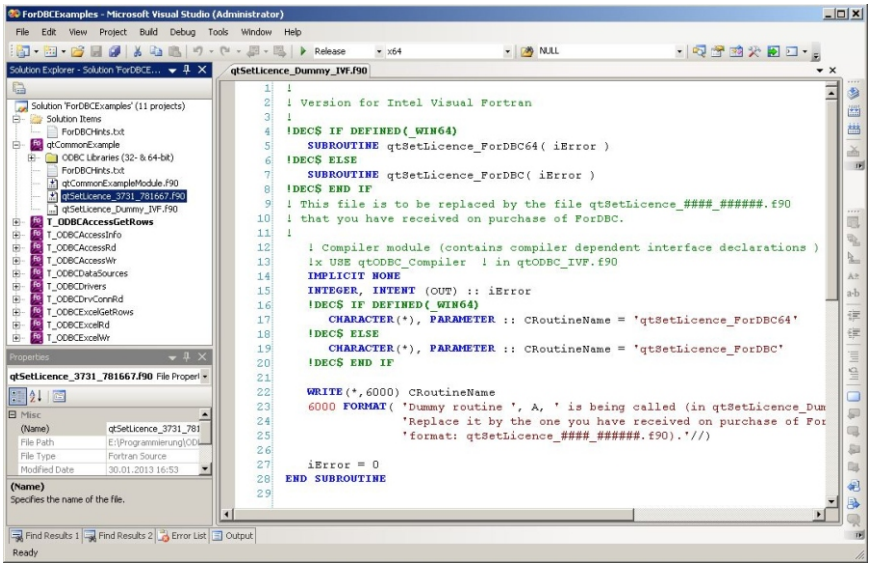

Abb. 19: Die Lizenzroutine ersetzt die "Dummy-Lizenzroutine", welche dann vom "Build " ausgeschlossen wurde ("Exclude File From Build= Yes").

D.h., statt der Datei qtSetLicence_Dummy_IVF.f90 binden Sie die erhaltene qtSetLicence_####_######.f90 in Ihren Projekten ein und geben die Lizenznummer beim Aufruf von ForDBC_Initialize an.

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

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

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

8. Quellen

Referenzen zu [ODBC..] beziehen sich auf:

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

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

[ODBC98] Microsoft Developer Network, Library Visual Studio 6.0, 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

qtODBCInterfaces

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Index

S

T

U

V

W

Z