Architecture of Pervasive.SQL

There are two configurations for Pervasive.SQL, depending on whether the underlying engine configuration is a client/server or a workstation engine configuration.

ODBC With a Client/Server Configuration

ODBC Interface uses two different client/server configurations. In these configurations, IBM PCs and compatibles running Windows, Windows 95, or Windows NT operating systems communicate with MicroKernel Database Engine database servers across a network such as Windows NT or NetWare. The two configurations differ in where the SQL statements are actually processed, as follows:

- Applications can access data in a MicroKernel database through Pervasive.SQL, which makes calls across the network to the Pervasive.SQL client/server engine. Pervasive.SQL communicates over the network through the Scalable SQL client Requester. SQL statement processing occurs on the server. Of the different ODBC Interface configurations, this approach uses the smallest amount of client-side CPU, disk, and memory resources. Figure 1-1 illustrates this configuration.

- Applications can also access server-based data in a MicroKernel database through ODBC Interface using the Pervasive.SQL workstation engine. This engine parses SQL statements locally, but accesses data in a MicroKernel database on the server using the Btrieve client Requester. Figure 1-2 illustrates this configuration.

Figure 1-1
Application Interface Architecture Using a Pervasive.SQL Server Engine

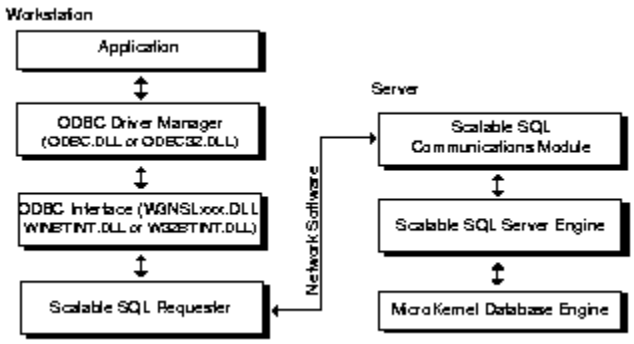
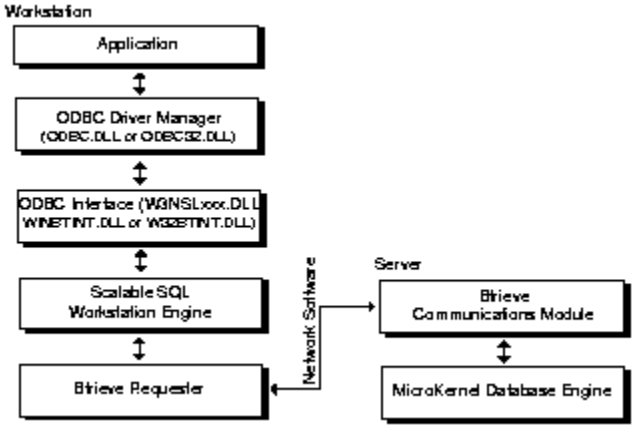


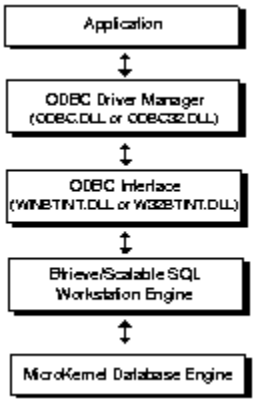
Figure 1-2
Application Interface Architecture Using a Pervasive.SQL Workstation Engine



ODBC and Workstation Engine Configurations

In workstation engine configurations, IBM PCs and compatibles running Windows, Windows 95, or Windows NT communicate with a MicroKernel engine running on the same machine. The ODBC Interface enables applications to access data in a MicroKernel database.
Figure 1-3 shows the application interface architecture for a workstation engine configuration:

Figure 1-3
Application Interface Architecture for Workstation Engines



Both the client/server and workstation engines rely on the underlying MicroKernel technology to provide fast, reliable access to data stored using the MicroKernel.

ODBC Interfaces for Windows, Windows 9X, and Windows NT

The ODBC Interface v2.5 for Windows NT/Windows 9X (W32BTINT.DLL) supports both 16- and 32-bit Windows NT and Windows 9X applications. The ODBC Driver Manager for Windows NT and Windows 9X is called ODBC32.DLL. 16-bit applications are supported via a thunking layer that is supplied as part of ODBC.
Use the ODBC Interface v2.5 for Windows (WINBTINT.DLL) for 16-bit Windows applications. The ODBC Driver Manager for Windows is called ODBC.DLL.

Borland Delphi

With the ODBC Interface 2.5, you must recreate Delphi aliases to ODBC data sources, because of a problem in the way Delphi handles ODBC data sources. When using the Delphi Refresh call to refresh the displayed rowset contents of a table in a grid, duplicate copies of some rows may appear. Future releases of Delphi should contain a fix.

Cognos Impromptu

When Impromptu performs an SQLDescribeCol call to determine the column name for a created column, the name is truncated at 20 characters. If the returned name needs a larger buffer, a “data truncated” error is returned. A workaround is to add the following line to the Cognos system file (COGDMOD.INI):

[Maximum Identifiers Name Length].dll=256

This change allows Impromtu to pass up to 256 characters to the ODBC Interface for retrieval. An alternate workaround is to edit the created name in the Impromtu expression editor.

Cognos Impromptu 3.5 is a 16-bit application and limits users to using only 16-bit ODBC data sources in 32-bit Windows environments. Cognos Impromptu 4.0 is a 32-bit application.

Compiling C/C++ Applications

To compile ODBC.C, perform the following steps:

1. Refer to your compiler's documentation for details about the particular compiler you are using.

2. Set up your project as a standard input/output application (a QuickWin application under Microsoft Visual C++, or an EasyWin application under Borland C++).

3. Choose the Large memory model. Although ODBC.C supports all memory models, Windows protected-mode programs operate best using the large memory model.

Sample make files are included for Microsoft Visual C++ v1.52 (Win16) and v4.0 (Win32) and for Watcom C v10.5 (Win16 and Win32 samples). These files are found in suitably named subdirectories under the samples directory. In the subdirectory, you may find a Readme file containing further instructions specific to that compiler.

Linking C/C++ Applications

Include the ODBC import library file, ODBC.LIB, in the list of libraries to be compiled into your application.

Configuring ODBC Interface

The following sections describe how to configure ODBC Interface and how to configure data sources that ODBC-enabled applications can use to access the information stored in your MicroKernel databases.

Configuration Issues

In Win16 environments, the ODBC Interface installs its configuration information in the following files:

- BTI.INI, the Pervasive Software product configuration information file
- ODBCINST.INI, the shared ODBC driver configuration information file
- ODBC.INI, the shared data source configuration information file

In Windows 9X and Windows NT, shared configuration information regarding ODBC drivers and data sources is stored in the Registry.

Pervasive.SQL

In Windows 9X and Windows NT client environments, the Microsoft Client for NetWare does not support the Novell APIs used by the Pervasive.SQL Database Names browser. If you are using the Microsoft Client for NetWare, Pervasive.SQL database names do not appear in the database names list box. An edit box has been added to the ODBC Administrator so users can target a specific Pervasive.SQL database name. (The Novell Client32 for Windows 95/Windows NT provides the necessary support and a list of available Pervasive.SQL remote database names appears in the ODBC Administrator Data Source Name setup dialog.)

Scalable SQL

NetWare

If you are using Scalable SQL v3.01 for NetWare and want to run the 32-bit ODBC driver, you must have the Scalable SQL v3.01.100 update. This patch includes an updated NetWare communication component and a 32-bit workstation interface.

On NetWare servers, you might encounter problems with the buffer size being too small when calling SQLTables. This problem commonly appears in list boxes that do not show all the tables that exist in the viewed database. To correct this problem, use the Scalable SQL Setup utility to increase either the Maximum Message Length (v3.01) or the Communications Buffer Size (v4.0).

Scalable SQL 4

If you are using Scalable SQL v4.x, the ODBC driver is unable to create ODBC data sources using long path names when connecting to Scalable SQL v4.x remote tables.

Applications

When attempting to update a table using a third-party front-end application (such as Microsoft Access) via ODBC Interface, you can receive Status Code 849. To correct this problem, use the Setup utility to increase the Pervasive.SQL engine's Communications Buffer Size.

ODBC Interface Reference - Contents

About This Manual

1 ODBC Overview

2 Getting Started

3 Creating DDFs for Btrieve Files

4 ODBC SQL Grammar

5 Programming via ODBC

A Extensions to ODBC

B Sample Program

C Programming Considerations

Conventions

Unless otherwise noted, command syntax and code examples use the following conventions:

Case	Commands and reserved words typically appear in uppercase letters. Unless the manual states otherwise, you can enter these items using uppercase, lowercase, or both. For example, you can type MYPROG, myprog, or MYprog.
[ ]	Square brackets enclose optional information, as in [*log_name*]. If information is not enclosed in square brackets, it is required.
|	A vertical bar indicates a choice of information to enter, as in [*filename* | @*filename*].
< >	Angle brackets enclose multiple choices for a required item, as in /D=<5|6|7> .
variable	Words appearing in italics are variables that you must replace with appropriate values, as in *filename* .
...	An ellipsis following information indicates you can repeat the information more than one time, as in [*parameter* ...].
::=	The symbol ::= means one item is defined in terms of another. For example, a::=b means the item *a* is defined in terms of *b* .

Creating DDFs for Btrieve Files

This section describes how to create DDFs for existing Btrieve files using ODBC. If you have existing Btrieve files, you cannot connect to them using ODBC unless you have created appropriate Data Dictionary Files (DDFs) for them. Without DDFs, ODBC does not know what tables, columns, and fields are contained in the data files.

Note: Scalable SQL data files already have DDFs created, so you do not need to read this section if you wish to connect to a Scalable SQL database.

You must perform two major steps to build proper DDFs for your Btrieve files. First, you must create a set of default ("empty") DDFs. Then you must either define a brand new database and populate the DDFs with those definitions, or else analyze a set of existing Btrieve files and modify the DDFs to correctly describe the structure of those files. DDF Ease can assist you with both of these tasks.

This section requires a basic understanding of SQL and possibly some programming expertise. It consists of the following topics:

Creating Default DDFs

There are several ways to create a set of default DDFs. The first way is to follow the instructions for "Adding a Data Source" . Creating a new ODBC data source causes a default set of DDFs to be created, if DDFs do not already exist in the specified **DDF Directory** .

You can also use DDF Ease to create DDFs. When you create a new database using DDF Ease, you create a set of default DDFs. See the *User's Guide* for more information on using DDF Ease.

The other ways to create a set of DDFs require that you have installed Scalable SQL or Pervasive.SQL. If you have Scalable SQL or Pervasive.SQL, you can create the database using the "CREATE DICTIONARY" SQL statement. However, to issue this call you must already be logged into an existing database, so there may be situations where you cannot use it.

Note: You can run SQLScope, a utility included with Scalable SQL and Pervasive.SQL, and log into the 'demodata' sample database, also included with these products. After logging into the 'demodata' database, you can create a new database with the "CREATE DICTIONARY" SQL statement. For more information on SQL statements, see the documentation provided with your software.

Alternatively, you can use the xDD API as documented in the Pervasive.SQL *Programmer's Reference* .

After creating the empty database, you should connect to it using an ODBC application, such as Microsoft's ODBC Test. You can also connect using DDF Ease. For further information about DDF Ease, refer to the *User's Guide* . If you have not defined a data source using the ODBC Administrator, then use the function SQLDriverConnect. If the new empty database is defined in the ODBC Administrator, then you can use the function SQLConnect.

Crystal Reports

Crystal Reports does not display TIME data fields correctly.

Data Types

ODBC Interface maps Scalable SQL and Btrieve data types to ODBC SQL data types. The following table lists Pervasive.SQL data types and shows the corresponding ODBC data types to which they are mapped.

Data Type Support is Dependent on the SQL Engine Version

The ODBC Interface supports all the data types supported by the SQL engine in use on a given connection. When using the Scalable SQL v3.01 engine as the back end, all Btrieve data types are supported except for UNSIGNED BINARY; with Scalable SQL v4.0 and Pervasive.SQL back ends, all Btrieve data types are supported. The Scalable SQL v4.0 and Pervasive.SQLengine supports several new data types in addition to those supported by the v3.01 engine.

ODBC and Data Type Masks

The Microsoft ODBC specification contains no concept of a display mask. When accessing data via ODBC, the application must take on the burden of displaying the data to meet end user expectations. For example, Microsoft Access neither sends nor receives masked data. Instead, as an ODBC application, it handles any display mask at the table view layer.

The ability to create a display mask for data returned by ODBC depends on the application that is formatting the data for screen or print presentation. Many applications, such as Microsoft Access, allow you to define a display mask for a column. Just keep in mind the mask only applies to display and is never communicated to the underlying database.

Nullability Varies By Data Type

Some Pervasive.SQL data types are not nullable. This means, for example, that if you use a conditional clause such as WHERE COL1 IS NULL and COL1 is a non-nullable type, the test evaluates to false. Non-nullable types should be tested for specific values or ranges of values instead of using IS NULL or IS NOT NULL in WHERE and HAVING clauses.

Table 4-2
Data type Characteristics

Pervasive.SQL Data Type	ODBC Data Type	CREATE TABLE Parameters	Nullable?
AUTOINC(2)	SQL_SMALLINT	Use as shown in column 1	No
AUTOINC(4)	SQL_INTEGER	Use as shown in column 1	No
BFLOAT(4)	SQL_REAL	Use as shown in column 1	No
BFLOAT(8)	SQL_DOUBLE	Use as shown in column 1	No
BIT	SQL_BIT	N/A	No
CURRENCY**1	SQL_BIGINT	Use as shown in column 1	Yes
LOGICAL(1) LOGICAL(2)	SQL_BIT SQL_SMALLINT	Use as shown in column 1	No
CHAR (STRING in Btrieve)	SQL_CHAR	Max Length	Yes
DATE	SQL_DATE	N/A	Yes
DECIMAL	SQL_DECIMAL	Precision, Scale	Yes
FLOAT(4)	SQL_REAL	Use as shown in column 1	No

FLOAT(8)	SQL_FLOAT or SQL_DOUBLE	Use as shown in column 1	No
INT(1)	SQL_TINYINT	Use as shown in column 1	No
INT(2)	SQL_SMALLINT	Use as shown in column 1	No
INT(4)	SQL_INTEGER	Use as shown in column 1	No
INT(8)*	SQL_BIGINT	Use as shown in column 1	No
LSTRING	SQL_VARCHAR	Max Length	Yes
LVAR	SQL_LONGVARBINARY	Max Length	No
MONEY	SQL_DECIMAL	Precision	Yes
NOTE	SQL_LONGVARCHAR	Max Length	No
NUMERIC	SQL_NUMERIC	Precision, Scale	Yes
NUMERICSA (NUMERIC SIGNED ASCII in Btrieve)	SQL_NUMERIC	Precision, Scale	Yes
NUMERICSTS (NUMERIC SIGNED TRAILING SEPARATE in Btrieve)	SQL_NUMERIC	Precision, Scale	Yes
TIME	SQL_TIME	N/A	No
TIMESTAMP*	SQL_TIMESTAMP	Timestamp precision	Yes
UNSIGNED(1)*	SQL_TINYINT	Use as shown in column 1	No
UNSIGNED(2)*	SQL_SMALLINT	Use as shown in column 1	No
UNSIGNED(4)*	SQL_INTEGER	Use as shown in column 1	No
UNSIGNED(8)*	SQL_BIGINT	Use as shown in column 1	No
ZSTRING	SQL_VARCHAR	Max Length	Yes

***1** Types marked with an asterisk in this table are available only on connections supported by a Scalable SQL v4.0 or Pervasive.SQL engine.

Note: All conversions in Appendix D of the *Microsoft ODBC 2.0 Programmer's Reference* are supported for the ODBC SQL data types listed in this table.

Implementation of the AUTOINC Data Type In ODBC

Although AUTOINC is mapped to the SQL_INTEGER and SQL_SMALLINT data types, AUTOINC fields require special handling under Pervasive.SQL with regard to INSERT and UPDATE statements. On insert, a zero value causes Pervasive.SQL to insert a unique value in the row for that field. A non-zero value is inserted as is, unless it causes a duplicate value to appear in that field in the table. Pervasive.SQL only inserts a unique value for the AUTOINC field if it is defined as a key.

Although treated as a signed INTEGER, AUTOINC types collate as absolute values, acting in a way as a 15- or 31-bit

unsigned INTEGER for purposes of collation. Therefore, a database user can use negative values as a bit-flag on the rows for selection purposes. To change the value to a negative value, use the UPDATE statement.

Using the NOTE and LVAR Data Types

Pervasive.SQL allows only a single field per table or view of type NOTE or LVAR, and that field must always be the last field defined in the table in CREATE TABLE statements and the last field listed in SELECT field lists.

Defining a New Database

If you are not trying to connect to an existing Btrieve file, then you can simply use an ODBC application (such as DDF Ease or Microsoft's ODBC Test (32-bit)) to execute CREATE statements to build the new database. Refer to the online or printed document *SQL Language Reference* for more information on CREATE.

Development Software Requirements

Programming to the ODBC API requires the following software:

- The ODBC Software Development Kit v2.10 for Win16 development, or v2.5 for Win32 development, available from Microsoft Corporation.

- A C or C++ compiler such as Watcom C/C++, Microsoft Visual C++, Symantec C++, or Borland C++. For the ODBC Interface for Windows NT/Windows 9X, the compiler must be capable of developing applications targeted to the Windows NT or Windows 95 environment.

Extensions to ODBC

This appendix describes using the ODBC Interface to perform certain Pervasive.SQL operations that do not have direct equivalents in the Open Database Connectivity Applications Programming Interface (ODBC API). The following topics are covered in this appendix:

General Programming Notes

This section provides information that may be useful in general when programming with ODBC Interface.

PRIMARY KEY not supported with Btrieve

ODBC 2.5 only supports the PRIMARY KEY keyword if you are using Pervasive.SQL and are logged into a named database.

If you attempt to use this keyword only with the Btrieve engine, Status Code 877 is generated. This limitation is fully explained in the *SQL Language Reference* .

INI settings for debugging

The following excerpt from BTI.INI shows the settings that can be used to generate a trace log for debugging ODBC operations:

[ODBC Interface]

ShowParserOutput=1

ShowParamsSent=1

LogFileName=c:\odbc.trc

ShowGetDataReturns=1

ShowCvtMaskFailures=1

ShowCvtMaskSuccesses=1

ShowConnects=1

The log file, as specified by LogFileName, contains a trace of the actual SQL statements that are sent to the engine.

Note: There is system overhead associated with these trace settings. They can be turned off by inserting a semi-colon ";" at the front of the line.

Optimizing "Order By" Client/Server Query Performance

When connecting as a Btrieve client to a remote Btrieve server, the ODBC Interface relies on included SQL engine components to process SQL statements.

When a query contains an order by clause using a non-indexed column, the engine creates a temporary external sort file. The ODBC Interface creates this file at the "external sort path" location, using the available Btrieve engine.

The external sort path is configured as follows:

For 32-bit applications, in the Win32 registry:

HKEY_LOCAL_MACHINE\SOFTWARE\Pervasive Software\Scalable SQL Engine\Version 3.01\Workstation Edition\Settings\external sort path = "D:\Winnt"

For 16-bit applications, in the BTI.INI file (located in the C:\WINDOWS directory):

[Scalable SQL Engine]

external sort path=D:\WINNT

The default location for the external sort path is the client machine's C:\WINDOWS directory.

The following query requires an external sort file since the order by column (“state”) is not indexed.

select * from patients order by state *[from the ODBC ‘demodata’ sample]*

If no Btrieve engine is available locally, the above query will return Status Code 234 “Scalable SQL cannot create an external sort file.”

If a local Btrieve engine is available on the client, then the local Btrieve engine creates the external sort file.

To correct the Status Code 234 and to ensure optimum performance when executing order by queries using non-index columns, you should change the external sort path directory to the location of the remote Btrieve server engine, by using a mapped drive to the server volume, as shown by drive “L:” in the example below. (In this usage, UNC paths are not supported).

In the Win32 registry:

HKEY_LOCAL_MACHINE\SOFTWARE\Pervasive Software\Scalable SQL Engine\Version 3.01\Workstation Edition\Settings\external sort path = "L:\Winnt"

In the BTI.INI (located in the C:\WINDOWS directory):

[Scalable SQL Engine]

external sort path=L:\WINNT

Directing the external sort path to the same location as the Btrieve server can increase order by query performance by 100%.

Remember that you can achieve the best performance by adding an index to all columns that are named in an order by query, thus eliminating the need for an external sort file.

For example, the following query against the sample database does not require an external sort file because the column “Last^Name” is indexed.

select * from patients order by Last^Name

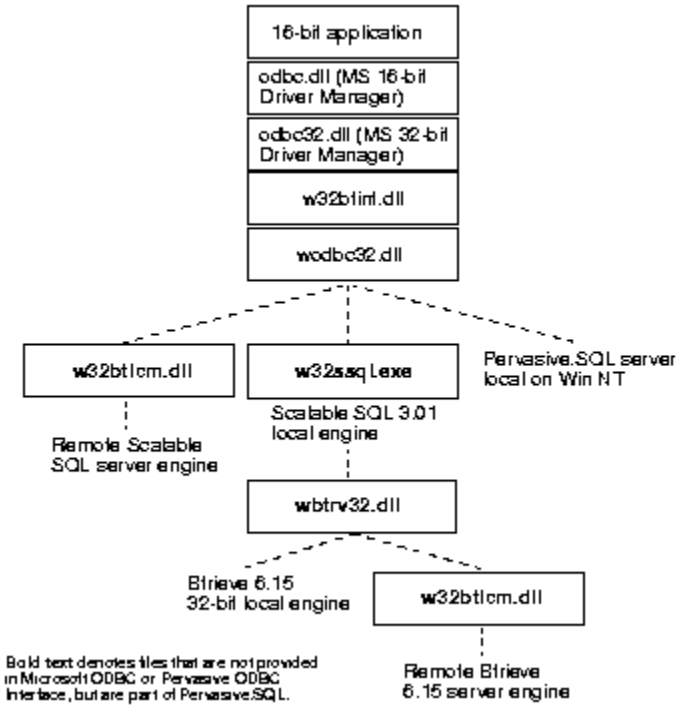
Thunking with Pervasive ODBC 2.04

Pervasive ODBC provides several execution paths for thunking. The recommended path, thunking at the Microsoft ODBC Driver Manager level, is described below. Also described below are alternate paths that permit thunking to occur.

Recommended Thunking Path

The recommended thunking path requires installation of both Microsoft drivers, 16-bit and 32-bit. If both drivers are properly installed, then thunking occurs automatically by default and you do not need to change any configuration attributes.

Figure 5-1
ODBC 2.04 Thunking through the Driver Manager



Thunking to Pervasive.SQL without the 32-bit ODBC Driver Manager

If you cannot or do not wish to install Microsoft ODBC-32, and you wish to use the SQL engine provided in Pervasive.SQL, you can specify thunking without the Microsoft ODBC-32 components. The following settings must be specified in your BTI.INI file:

[Scalable SQL]

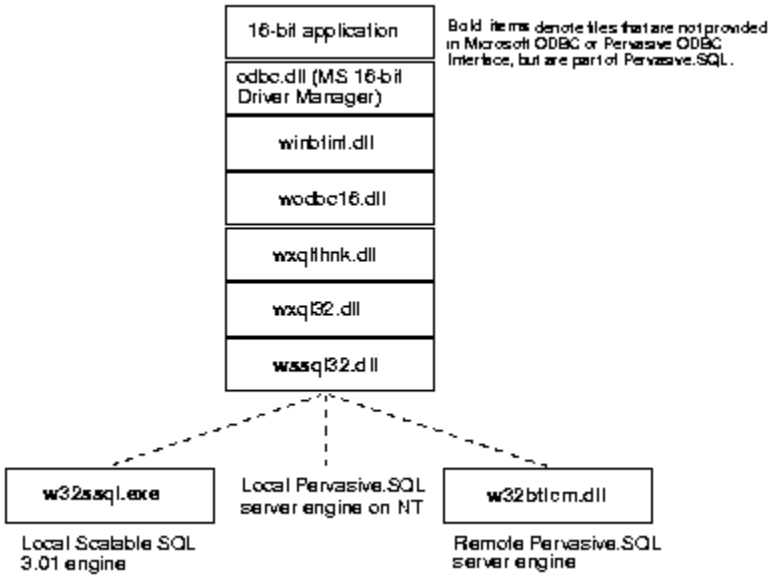
local=no

requester=no

thunk=yes

In the diagram below, note that component attributes are controlled by Registry keys from wssql32.dll to the bottom of the diagram. See “Windows 3.x Applications on Windows 9X/NT Systems” on page 2-12 for instructions on how to specify the path to your selected target engine.

Figure 5-2
ODBC 2.04 SQL Thunking without 32-bit Driver Manager



Thunking to Btrieve without the 32-bit ODBC Driver Manager

If you cannot or do not wish to install Microsoft ODBC-32, and you do not wish to use the SQL engine provided in Pervasive.SQL, you can specify thunking without the Microsoft ODBC-32 components. The following settings must be specified in your BTI.INI file:

[Scalable SQL]

local=yes

requester=no

thunk=no

[Btrieve] (BTRV 6.15 component)

local=no

requester=no

Thunk=yes

[Microkernel Router] (BTRV 7.0 / SSQL 4 component)

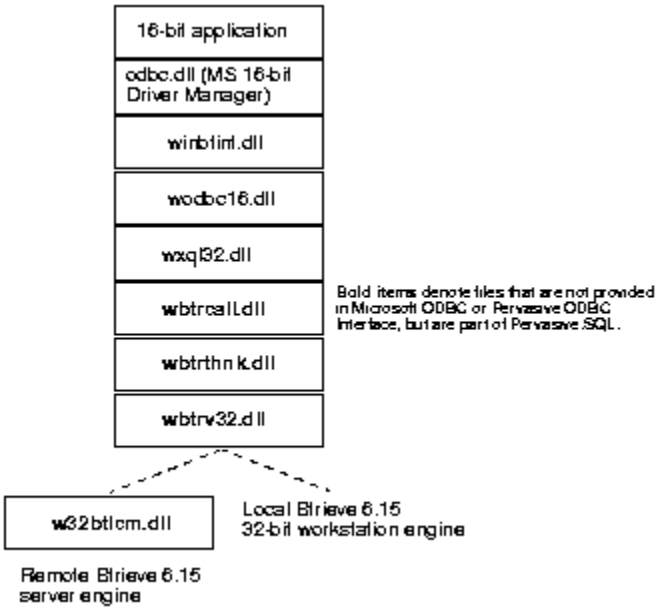
local=no

requester=no

thunk=yes

In the following diagram, note that component attributes are controlled by Registry keys from wbtrv32.dll to the bottom of the diagram. See “Windows 3.x Applications on Windows 9X/NT Systems” on page 2-12 for instructions on how to specify the path to your selected target engine.

Figure 5-3
ODBC 2.04 Btrieve Thunking without the 32-bit Driver Manager



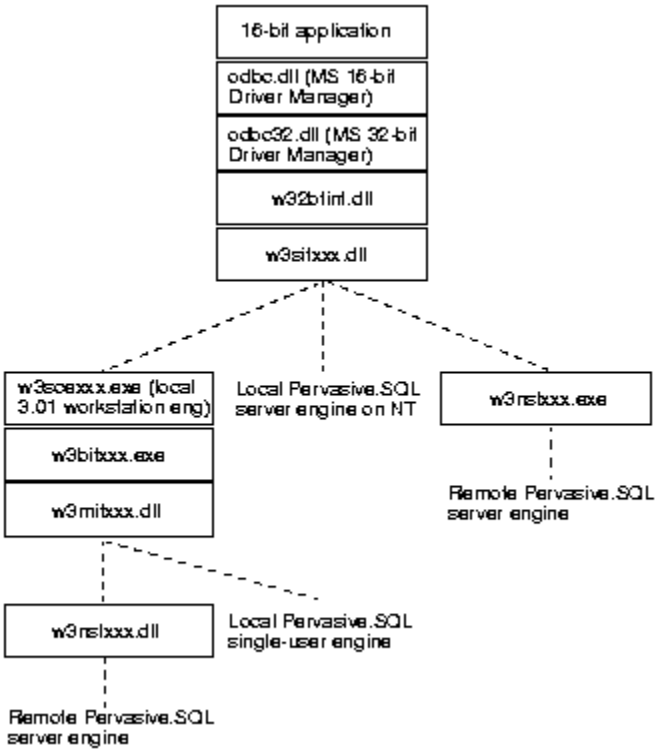
Thunking with Pervasive ODBC 2.5

Pervasive ODBC provides several execution paths for thunking. The recommended path, thunking at the Microsoft ODBC Driver Manager level, is described below. Also described below are alternate paths that permit thunking to occur.

Recommended Thunking Path

The recommended thunking path requires installation of both Microsoft drivers, 16-bit and 32-bit. If both drivers are properly installed, then thunking occurs automatically by default and you do not need to change any configuration attributes.

Figure 5-4
ODBC 2.5 Thunking through the ODBC Driver Managers



Thunking to Pervasive.SQL without the 32-bit ODBC Driver Manager

If you cannot or do not wish to install Microsoft ODBC-32, and you wish to use the SQL engine provided in Pervasive.SQL, you can specify thunking without the Microsoft ODBC-32 components. The following settings must be specified in your BTI.INI file:

[Scalable SQL]

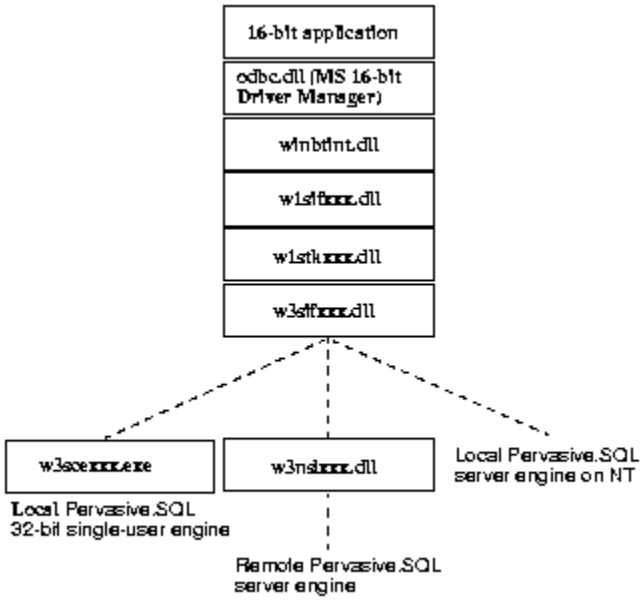
local=no

requester=no

thunk=yes

In the diagram below, note that component attributes are controlled by Registry keys from w3sifxxx.dll to the bottom of the diagram. See “Windows 3.x Applications on Windows 9X/NT Systems” on page 2-12 for instructions on how to specify the path to your selected target engine.

Figure 5-5
ODBC 2.5 SQL Thunking without the 32-bit Driver Manager



Thunking to Btrieve without the 32-bit ODBC Driver Manager

If you cannot or do not wish to install Microsoft ODBC-32, and you do not wish to use the SQL engine provided in Pervasive.SQL, you can specify thunking without the Microsoft ODBC-32 components. The following settings must be specified in your BTI.INI file:

[Scalable SQL]

local=yes

requester=no

thunk=no

[Btrieve] (BTRV 6.15 component)

local=no

requester=no

Thunk=yes

[Microkernel Router] (BTRV 7.0 / SSQL 4 component)

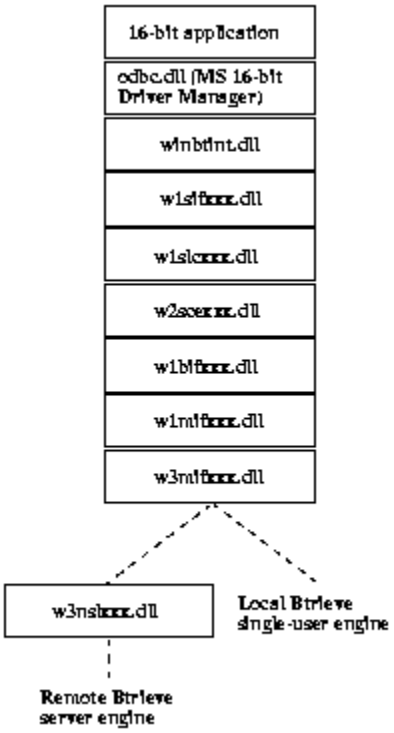
local=no

requester=no

thunk=yes

In the diagram below, note that component attributes are controlled by Registry keys from w3mifxxx.dll to the bottom of the diagram. See "Windows 3.x Applications on Windows 9X/NT Systems" on page 2-12 for instructions on how to specify the path to your selected target engine.

Figure 5-6
ODBC 2.5 Btrieve Thunking without the 32-bit Driver Manager


16-bit application
odbc.dll (MS 16-bit Driver Manager)
w1nbtint.dll
w1slfxxx.dll
w1slcxxx.dll
w2scexx.dll
w1blfxxx.dll
w1nlfxxx.dll
w3nlfxxx.dll
w3nslxxx.dll
Local Btrieve single-user engine
Remote Btrieve server engine

Getting Started

This chapter describes using the Open Database Connectivity (ODBC) Application Programming Interface via the ODBC Interface, an alternative API for Pervasive.SQL. The following topics are covered in this chapter:

Hardware and Software Requirements

This section describes the hardware and software that the ODBC Interface requires.

ODBC Interface

The ODBC Interface requires the following hardware:

- An 80386 or higher microprocessor.
- At least 8 MB of RAM (12 MB RAM recommended) for the Win16 ODBC Interface.
- At least 16 MB of RAM (20 MB RAM recommended) for the Win32 ODBC Interface.
- A fixed disk drive and 5 MB of hard disk space for a complete installation.

The ODBC Interface running on DOS or Windows 3.x (16-bit) requires the following software:

- PC-DOS or MS-DOS v5.0 (or later).
- Windows 3.1 or later.
- For client/server engine configurations, one of the following:
    - Btrieve server engine 6.15 or later for NetWare or Windows NT
    - Scalable SQL server engine 3.01 or later for NetWare or Windows NT
    - Scalable SQL Windows NT 3.01 OEM release
    - Scalable SQL 3.01 or later for NetWare or Windows NT
    - Pervasive.SQL 7 or later (server) for NetWare or Windows NT
- For workstation engine configurations, one of the following:
    - Btrieve 6.15 for Windows or Windows 9X/NT
    - Scalable SQL workstation engine 3.01
    - Scalable SQL server engine 4.0 or later for Windows NT, configured for use as a workstation engine on Windows NT
    - Pervasive.SQL 7 or later (workstation) for Windows 9X or Windows NT

The ODBC Interface running on Windows NT or Windows 9X (Win32) requires the following software:

- Windows NT 4.0 or Windows 9X.
- If you are using a client/server configuration, you must have the following installed:
    - Btrieve 6.15 client Requester, Scalable SQL 3.01 or later client Requester, or Pervasive.SQL client Requester(s), on the same machine where you wish to install the ODBC Interface.
    - Btrieve 6.15 server engine, Scalable SQL 3.01 or later server engine, or Pervasive.SQL server engine(s), accessible over the network.
- For workstation engine configurations, Scalable SQL 3.01 or 4.0 for Windows NT/Windows 95 or Btrieve

6.15 for Windows NT/Windows 95 or Pervasive.SQL Workstation.

Network Software

In client/server configurations, a network is required to connect the client machine running the application using the ODBC Interface with its database server.

With these configurations, the ODBC Interface and Pervasive.SQL use a common network interface. You can use the ODBC Interface on any machine that is capable of a connection (for example, when using SQLScope or the Pervasive.SQL 4.0 Monitor utility) from a computer running Windows v3.x, Windows 9X, or Windows NT to a Pervasive.SQL database server engine running on a network server.

Installing on Windows 3.x

This section describes the steps to follow in order to install ODBC Interface on Windows 3.1 or Windows for Workgroups.

Note: ODBC Interface (16-bit) is provided on the CD with Pervasive.SQL Server, and it is included in the Typical client installation process. You only need to install ODBC Interface separately if:

you chose a Custom client install and did not install ODBC Interface, or
you chose a Minimum client install, or
you do not have Pervasive.SQL, and/or you have purchased an ODBC Interface license or upgrade.

Customers using older versions of Pervasive products can obtain ODBC Interface on diskette. You may also use the diskette install if you are upgrading from a previous version of ODBC Interface.

Note: You must have Pervasive.SQL, Btrieve, or Scalable SQL installed on a server or your computer before you can use ODBC Interface.

To install ODBC Interface

1. **Insert the first diskette or the CD.**

   Insert the diskette labelled “Disk 1”, or the Pervasive.SQL CD.

2. **Start the installation.**

Diskette installation:

   a. On your desktop, double-click “My Computer” and then double-click on the disk drive icon where the ODBC Interface software is.

   b. Double-click on “Setup.exe.”

   c. Click **Next** to bypass the Welcome screen. Now proceed to the next step.

CD installation:

   a. On your desktop, double-click “My Computer” and then double-click on the CD drive icon where the Pervasive.SQL CD is located.

   b. Double-click on the folder labeled “Clients.” Then double-click on the folder labeled “ODBC.” Then double click on the folder labeled “16.” Then double-click on “setup.exe” to start the installation.

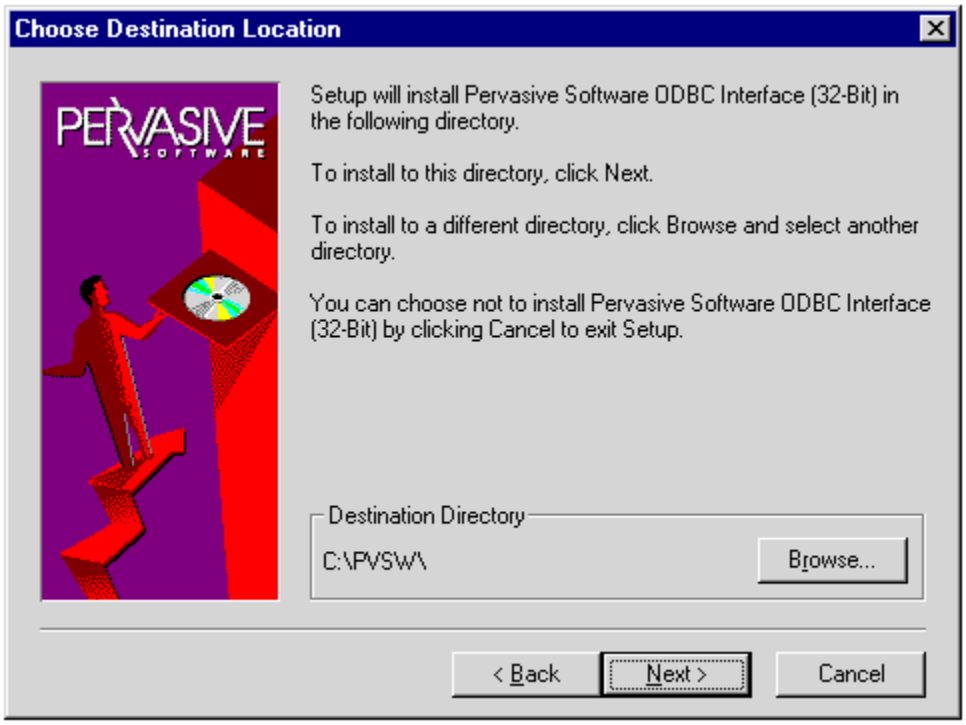
   c. Click **Next** to bypass the Welcome screen.

3. Choose the installation directory.

a. As shown in Figure 2-2, you must choose the directory in which to install ODBC.

b. If you do not want to install in the default location, click **Browse** to select a different location.

Figure 2-2
Choosing the ODBC Installation Directory



c. When you have selected your desired destination directory, click **Next** to continue the installation of the ODBC files.

Note: Please wait while the ODBC Interface files are installed onto your hard disk.
Users : The Installation utility attempts to copy a file called CTL3DV2.DLL to the default Windows SYSTEM directory. If you are currently using a Windows program (such as MS Office) that accesses this DLL, the Installation utility displays a message box indicating that it cannot copy this file. Choose the IGNORE option and continue the install process.

Your installation is now complete.

Installing on Windows NT or Windows 9X

This section describes the steps for installing ODBC Interface on Windows NT or Windows 9X.

Note: If you have installed Pervasive.SQL Workstation, you do not need to follow these instructions. ODBC Interface is automatically installed with Pervasive.SQL Workstation.
Interface is also provided on the CD with Pervasive.SQL Server, and it is included in the Typical client installation process. You only need to install ODBC Interface separately if:

you chose a Custom client installation and did not install ODBC Interface, or
you chose a Minimum client installation, or
you do not have Pervasive.SQL, and/or you have purchased an ODBC Interface license or upgrade.

Customers using older versions of Pervasive products can obtain ODBC Interface on diskette. You may also use the diskette install if you are upgrading from a previous version of ODBC Interface.

Note: You must have either Pervasive.SQL Workstation installed locally or Pervasive.SQL Server installed on a server before you can use ODBC Interface.

To install ODBC Interface

1. **Insert the first diskette or the CD.**

   Insert the diskette labelled “Disk 1”, or the Pervasive.SQL CD.

2. **Start the installation.**

Diskette installation:

a. On your desktop, double-click “My Computer” and then double-click on the disk drive icon where the ODBC Interface software is.

b. Double-click on “Setup.exe.”

c. Click **Next** to bypass the Welcome screen. Now proceed to the next step.

CD installation:

a. On your desktop, double-click “My Computer” and then double-click on the CD drive icon where the Pervasive.SQL CD is located.

b. Double-click on the folder labeled “Clients.” Then double-click on the folder labeled “ODBC.” Then double click on the folder labeled “32.” Then double-click on “setup.exe” to start the installation.

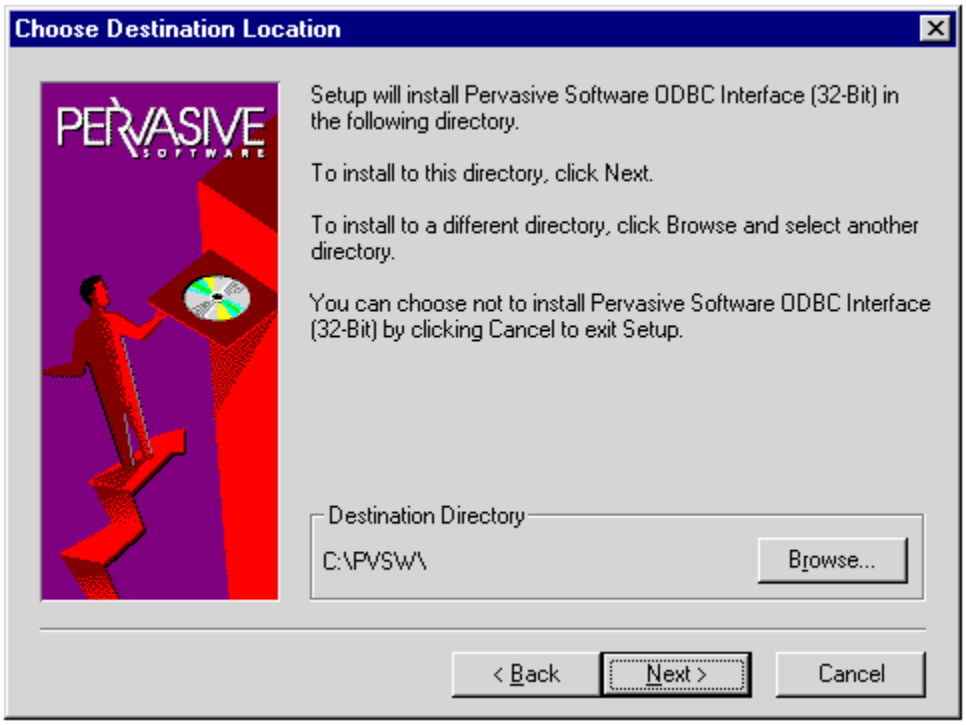
c. Click **Next** to bypass the Welcome screen.

3. Choose the installation directory.

a. As shown in Figure 2-1, you must choose the directory in which to install ODBC.

b. If you do not want to install in the default location, click **Browse** to select a different location. If you plan to install both the Win16 and the Win32 ODBC drivers, DO NOT install them in the same directory. If they are installed in the same directory, uninstalling one of them later will remove components required by the other.

Figure 2-1
Choosing the ODBC Installation Directory



c. When you have selected your desired destination directory, click **Next** to continue the installation of the ODBC files.

Note: Please wait while the ODBC Interface files are installed onto your hard disk.

Your installation is now complete.

Interface Modules

Following are the interface modules. You need the include files for both Win16 and Win32 environments and either ODBC.LIB or ODBC32.LIB, depending on whether you are targeting a Win16 or Win32 platform for the application.

SQL.H andSQLEXT.H

These files are part of the Microsoft ODBC Software Development Kit (ODBC SDK). This developer kit is included in the Microsoft Developer Network subscription service. It is also included with Microsoft Visual C++ and most other Windows-hosted C/C++ compiler packages.

Together, these files declare the ODBC API functions, manifest constants, and platform-independent data types.

ODBC.LIB

This file is part of the ODBC SDK from Microsoft and, like the include files, is part of most Windows C/C++ compiler packages. It is the import library for the Win16 ODBC Driver Manager, ODBC.DLL.

ODBC32.LIB

This file is part of the ODBC SDK from Microsoft and, like the include files, is part of most Windows C/C++ compiler packages. It is the import library for the Win32 ODBC Driver Manager, ODBC32.DLL.

About This Manual

This manual contains information about using the Open Database Connectivity Applications Programming Interface (ODBC API) via the ODBC Interface, an alternative API for Scalable SQL and Btrieve.

Using Bookmarks

Bookmarks allow the programmer to navigate quickly within a large result set. A bookmark is a 32-bit value that represents a particular row in the result set.

The implementation of bookmarks in the ODBC Interface is powerful, but also memory-intensive, especially if the bookmarks are obtained by binding column zero in the result set, which results in the retrieval and storage of every bookmark as it is fetched. Care must be taken when using bookmarks in order to avoid consuming large amounts of memory needlessly.

Bookmark values persist for the lifetime of a connection. That means they can be used on a single statement handle even if the statement is closed and re-executed, across transactions, and even on multiple statement handles on the same connection. In cases where the bookmark is used against a different statement handle or on the same handle after a query is closed and another query executed, the second query must have the same row structure as the first and be made against the same table or tables.

Controlling Bookmark Memory Usage

The amount of memory consumed by bookmarks can be calculated as follows:

Number of tables in query * Number of bookmarks fetched * 4

This value is cumulative over the life of a connection.

For each table contributing to a row in the query, a four-byte position value is saved as part of the bookmark. The value returned as the bookmark is not the position value, because multiple tables may be involved in the query, in which case the row is represented by an array of such four-byte values. Instead, the bookmark value is a “handle” that can be used by the application in place of the array of position values.

There is a small amount of additional memory overhead involved in keeping the list of bookmarks, but it is insignificant compared with the accumulation of the bookmarks themselves when many bookmarks are fetched over the lifetime of a connection.

The least memory consumption can be attained when using bookmarks by calling SQLGetStmtOption with theSQL_BOOKMARK option to retrieve bookmarks only when you intend to re-use them. A front-end application might accomplish this by allowing the user the option of marking a row for later retrieval, perhaps using a mechanism such as a check box on the data display dialog.

Programming With Bookmarks

See the *ODBC 2.0 Programmer’s Reference* from Microsoft Press for a discussion on how to use SQLExtendedFetch and bookmarks. The example can be found in Chapter 22, “ODBC Function Reference,” in the section on SQLExtendedFetch. The subsection is entitled “Positioning to a Bookmark.”

Performing Bulk Operations Using Parameter Arrays

The ODBC Interface supports passing arrays of parameters when executing SQL statements with parameters. For example, when performing a SQL INSERT statement, it is possible to insert an arbitrary number of rows rather than a single row in the execution of the statement.

Operation

To take advantage of this feature:

1. Call SQLParamOptions to set the number of elements in each parameter array (all parameters must have the same number of elements).

2. For each parameter, allocate storage for an array of parameter values (which becomes the rgbValue argument to SQLBindParameter for this parameter) and a corresponding array of parameter lengths (which becomes the pcbValue argument to SQLBindParameter).

3. For each parameter, call SQLBindParameter to notify the interface of the location of the parameter data.

4. For each row to be processed, perform the following:

    a. For each parameter in the row, initialize the corresponding elements in the parameter's value array to the parameter's value in that row.

    b. Initialize the corresponding element in the parameter's pcbValue array to the proper length for that parameter value.

5. Call SQLExecDirect (or SQLPrepare and SQLExecute) to execute the statement, causing all rows to be processed.

Error Handling

If an error occurs while processing, the unsigned Win32 integer whose address was passed as the third argument to SQLParamOptions contains the index of the row where the error occurred. This index is one-based rather than zero-based: the first row is row 1, the second is row 2, and so on.

To skip the offending row and continue processing, call SQLBindParameter for each parameter, using the address of the next element in the data array and the length array as the rgbValue and pcbValue argument respectively. Then call SQLExecDirect or SQLExecute to continue processing.

Limitations

The following limitations apply to bulk operations using parameter arrays:

- Data-at-execution parameters are not supported.

- No parameter array may exceed 64KB in size (Win32 ODBC Interface only).

- Stored statements may not be used with parameter arrays.

For a brief coding example showing how to perform bulk insertion, see the function description for SQLParamOptions in Chapter 22 of the *ODBC 2.0 Programmer's Reference* from Microsoft Press.

Configuring Data Sources

Before you can access MicroKernel data with the ODBC Interface, you must use the ODBC Administrator to configure a data source for each MicroKernel database. ODBC-enabled applications use this data source information to connect to the database.

A data source corresponds to one of the following:

- A named database.
- A directory in which Data Dictionary Files (DDFs) reside containing the definition of the database schema, and optionally a data file directory in which the actual data files reside if not in the same directory as the DDFs.

Note: Btrieve data files cannot be accessed by ODBC unless DDFs have been created for them. If there are no DDFs associated with your Btrieve data file, please see "Creating DDFs for Btrieve Files" . Because Btrieve files can be complex, you are not encouraged to create DDFs for existing Btrieve files unless you are an application developer.

You can also change the definition of a data source, or delete a data source.

In order for an ODBC application to access MicroKernel data files, the application must connect to a data source defined for a dictionary. A dictionary consists of at least three files: FILE.DDF, FIELD.DDF, and INDEX.DDF; other DDF files may also be part of a dictionary. The dictionary is a system catalog that contains table layout information such as field and index names, sizes, lengths, and attributes that define each data file in a given database. These DDF files are actually Btrieve data files and have a very specific set of requirements defining how they should be built and populated with table layout information.

If a dictionary is not built properly or has invalid or improperly structured data, ODBC applications will not function correctly. Invalid dictionaries usually result from manually creating the dictionary with direct Btrieve API calls, rather than using the ODBC Interface or a SQL utility or application. Creating a dictionary via ODBC or SQL guarantees the integrity of the DDFs and the definitions in them; creating a dictionary via Btrieve calls does not.

The most common problem is an unexpected Status Code 204, 6, or 4 when accessing the database. For example:

- Status Code 204 often results from a table name index (index number 1 in FILE.DDF) that is defined as case sensitive, when it should be case insensitive. In a properly built dictionary, you can specify table and field names with any case, regardless of the way it was originally defined.
- Status Code 4 often results either from incorrectly defined indexes (other than FILE.DDF), because the ODBC Interface cannot locate the data it needs, or from a MicroKernel data file that does not match the dictionary definition.
- Status Code 6 often results from incompatible INDEX.DDF files. Pre-v3.0 INDEX.DDF files have only two indexes, but v3.0 and later INDEX.DDF files have three indexes. If the ODBC Interface tries to access a pre-v3.0 dictionary, it automatically tries to add the third index. This index has the UNIQUE attribute and is built on three fields in the X$Index table: Xi$File, Xi$Number, and Xi$Part. Improperly structured dictionaries may have invalid data, such as duplicate entries for these fields. In such cases, the ODBC Interface receives Status Code 6 when attempting to add the unique index.

Users with invalid dictionaries should contact the application vendor or developer and request a valid dictionary.

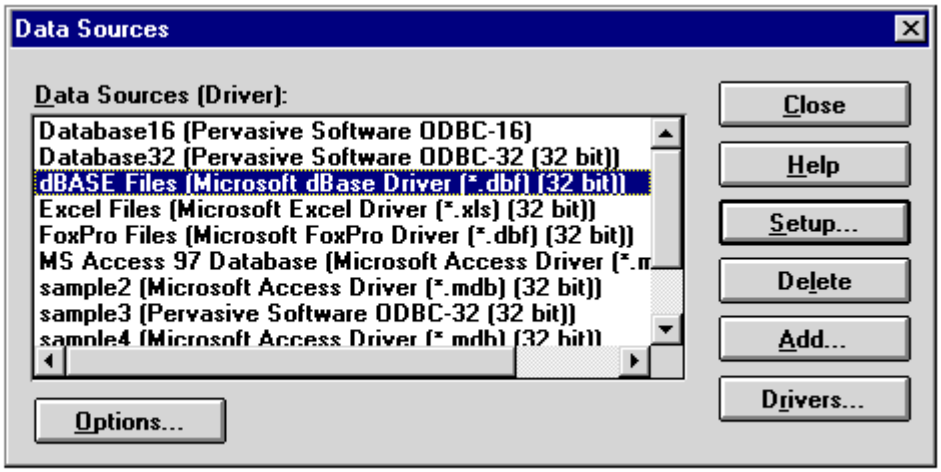
Adding a Data Source

The ODBC Interface uses the information you enter when you add the data source to permit ODBC-enabled applications to access your database. You must perform these steps from the workstation or client where your ODBC-enabled application will run.

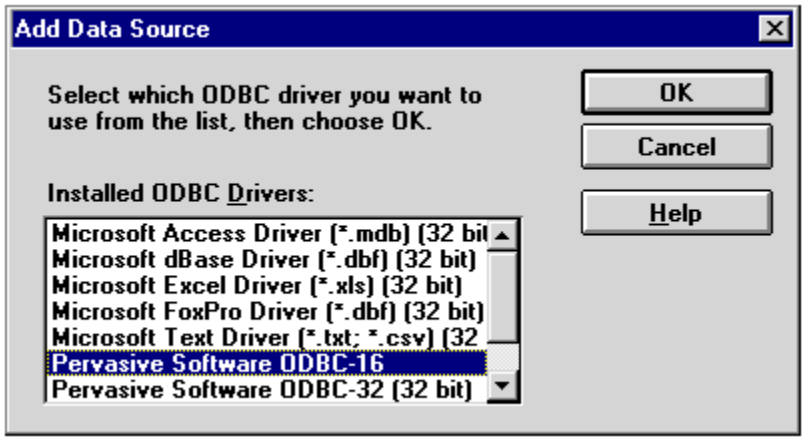
Note: If you specify an owner name in the ODBC Add Data Source dialog and the owner name includes a leading non-alphabetic character or trailing blanks, you must enclose the name in single quotes.

To add a data source in Windows 3.x:

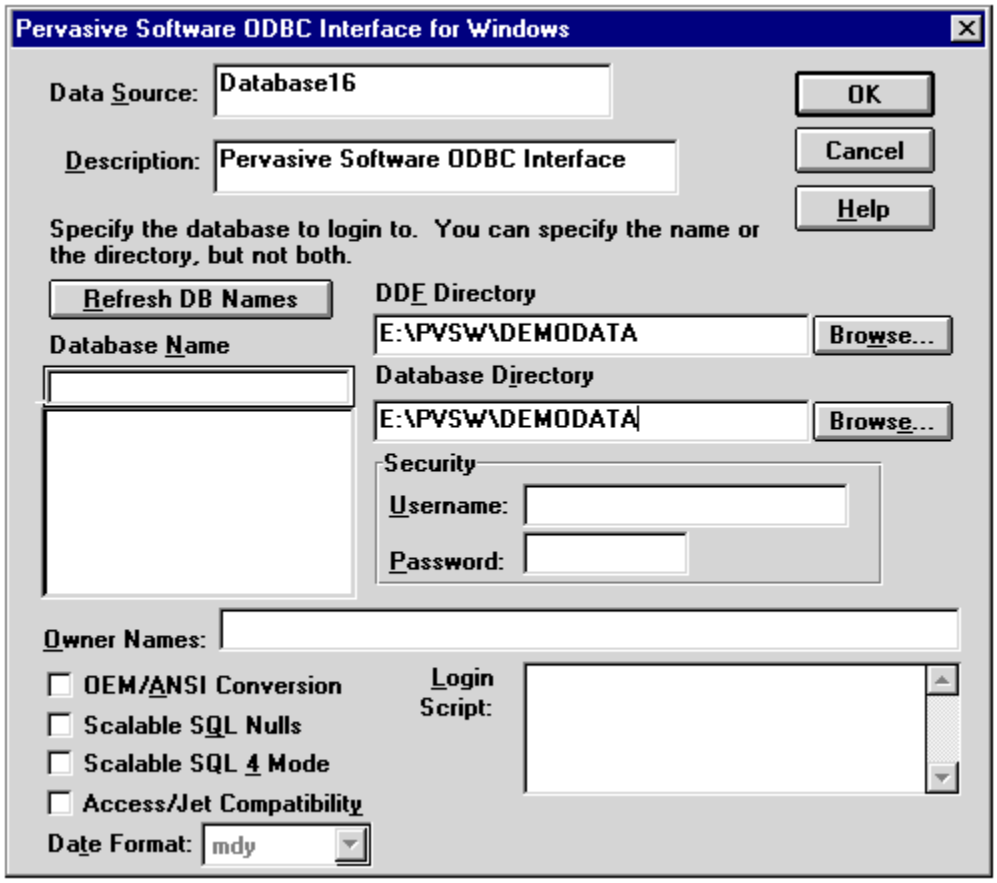
1. Double-click the **Pervasive Software ODBC** program group.
2. Double-click the **ODBC Administrator** icon. You should see the **Data Sources** dialog box:



3. Click **Add** . You should see the **Add Data Source** dialog box:



4. Choose **Pervasive Software ODBC-16** from the list of drivers and click **OK** . You should see the **ODBC Interface** configuration dialog:



5. In the ODBC Interface dialog box, create a name for the data source in the first empty field. In the picture, the ODBC data source is named Database16.

To Select a Named Database:

If you have any Pervasive.SQL databases set up as Named Databases, you can view a list of those available by clicking on **Refresh DB Names** . You may select one of them as the data source by clicking its name in the **Database Names** list box, or by typing a name. If you choose a Named Database, do not enter anything in the **DDF Directory** or **Database Directory** fields. See "Named Databases" for additional information.

Note: Data dictionary files must already exist for named databases. When you create a new data source and you specify a directory that does not contain data dictionary files, you are prompted to allow the ODBC Interface to create them for you. If you select a named database and files do not exist in the directory specified in the definition of the database, the connection cannot be made and the data source addition fails.

To Specify the Data Files:

1. If you do not want to use a Named Database, you must specify a path to the target databases's DDFs in the **DDF Directory** field. In the picture, the DDFs reside in the directory E:\pvsw\demodata.

2. If the data files are not in the same directory as the DDFs, you must enter the location of the data files in the **Database Directory** field.

To Use Security:

1. If you want the ODBC data source to use database engine security, enter a Pervasive.SQL user name in the **Username** field. Enter the appropriate password in the **Password** field only if you want the password to be sent automatically whenever you connect to the database. If you leave the password blank, you will be prompted for the password upon connecting to the data source.

2. If you want to use Pervasive owner names when connecting to this data source, enter them in the **Owner Names** field. You may enter up to 8 names, separated by commas. You will be able to access any database which has an Owner Name matching one or more of the names listed.

To Select Format and Compatibility Options:

1. If your database stores characters in the OEM character set, and your application needs to manipulate data that uses the Windows character set, check the **OEM/ANSI Conversion** box. This selection causes the ODBC driver to translate the characters between the two character sets.

2. If you expect your application to use Scalable SQL-style null handling, check the **Scalable SQL Nulls** box. For more information, see "Scalable SQL-style Null Handling".

3. If you expect your application to log into the database server using the Scalable SQL 4 features available only in Scalable SQL 4 and Pervasive.SQL, check the **Scalable SQL 4 Mode** box. If you want your application to login using only Scalable SQL 3.01 features, even if the server engine is version 4 or later, do not check the box.

4. If you plan to use the new data source exclusively with Microsoft Access, check the **Access/Jet Compatibility** box. Access is known to have problems with certain data types (notably TIME, DECIMAL, and the various NUMERIC types). These problems show up as #deleted errors and reports that a record has been updated by another user when it has actually not been updated by anyone else. DO NOT check this box unless you are encountering #deleted or other unexpected errors in Access.

5. If you wish to change the default formatting of dates, choose the desired format using the **Date Format** list box. This option is only available when Scalable SQL 4 Mode is checked. The options are: mdy, myd, dmy, dym, ymd and ydm.

   If the date format is set to any value other than the default (mdy), SET DATEFORMAT is called using the specified date format after a successful login to the database. You can then select, insert, or update data using the specified format. For example, if Date Format is ymd, a date field can be formatted as in the following SQL statement:

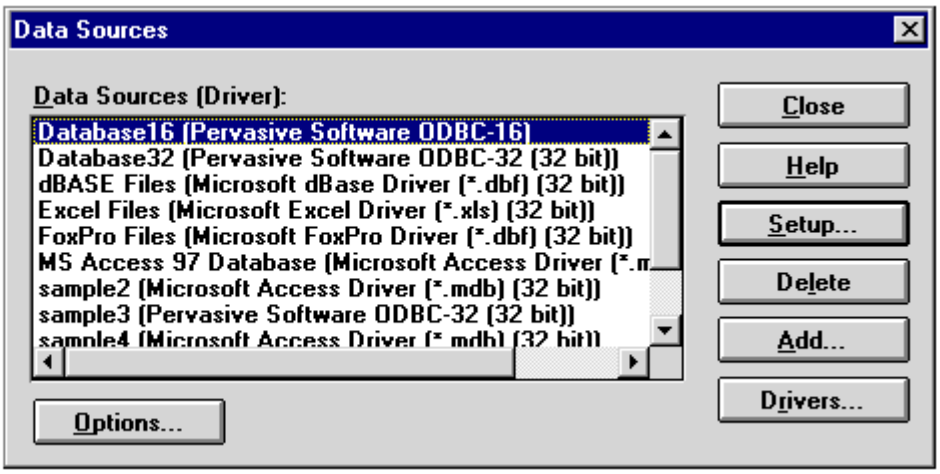
```
Select * from tablename where datecol = '1925/12/25'
```

6. If you wish to have SQL statements executed immediately after you have logged into the database, enter them in the**Login Script** box. See "Login Scripts" for further information.

Note: When adding or modifying a data source, the parameters you supply must allow a successful login to occur.

To Save Your Changes:

When you are finished, click **OK** . You should now see your new data source listed in the **Data Sources** dialog box:



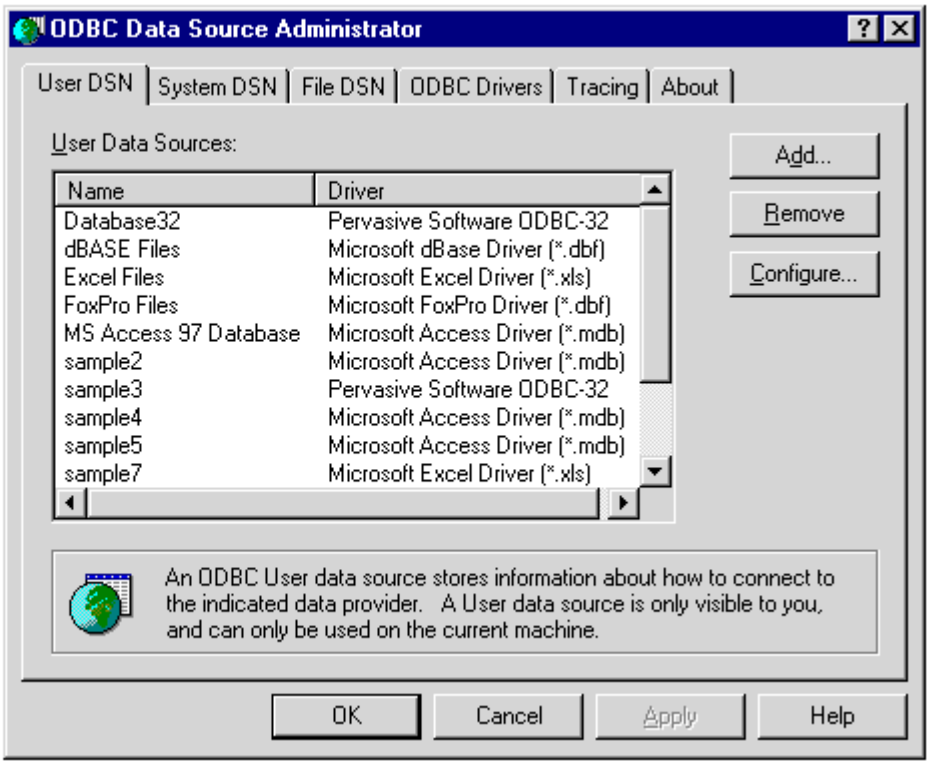
Your ODBC data source configuration is complete. Click **Close** .

To add a data source in Windows 9X or Windows NT:

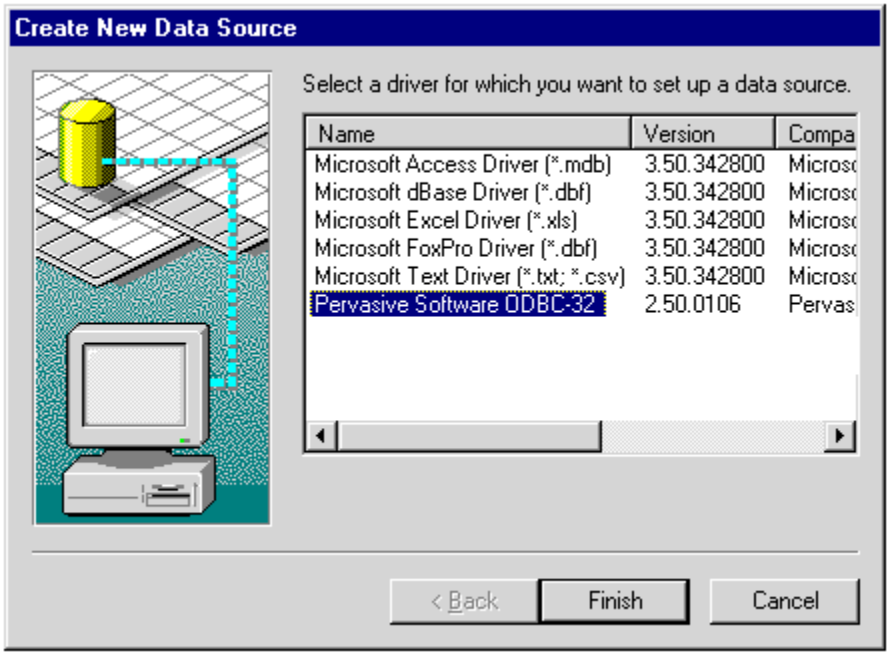
1. Click the **Start** button, then point to **Programs** | **Pervasive SQL 7** and click **32bit ODBC Administrator** .

Note: If you installed ODBC Interface separately, click the **Start** button, then point to **Programs** | **Pervasive ODBC - 32 Bit** .

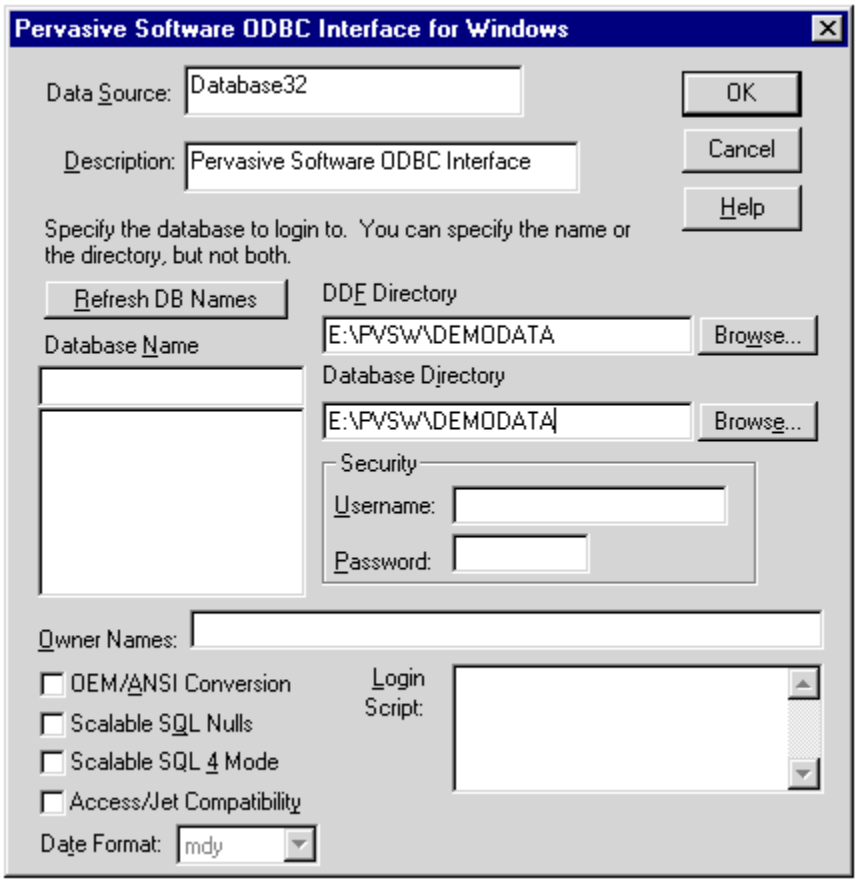
2. You should see the **ODBC Administrator** dialog box, with the **User DSN** tab in front:



3. On the User DSN tab, click **Add** . You should see the **Create New Data Source** dialog box:



4. Select **Pervasive Software ODBC-32** from the list of drivers and click **Finish** . You should see the **ODBC Interface configuration** dialog:



5. In the ODBC Interface dialog box, create a name for the data source in the **Description** field. In the figure, the ODBC data source is named Database32.

To Select a Named Database:

If you have any Pervasive.SQL databases configured as Named Databases, you can view a list of those available by clicking on **Refresh DB Names** . You may select one of them as the data source by clicking its name in the **Database Names** list box, or by typing a name. If you choose a Named Database, do not enter anything in the **DDF Directory** or **Database Directory** fields. See "Named Databases" for additional information.

Note: Data dictionary files must already exist for named databases. When you create a new data source and you specify a directory that does not contain data dictionary files, you are prompted to allow the ODBC Interface to create them for you. If you select a named database and files do not exist in the directory specified in the definition of the database, the connection cannot be made and the data source addition fails.

To Specify the Data Files:

1. If you do not want to use a Named Database, you must specify a path to the target databases's DDF files in

the **DDF Directory** field. In the picture, the DDFs reside in the directory E:\pvsw\demodata.

2. If the data files are not in the same directory as the DDFs, you must enter the location of the data files in the **Database Directory** field.

To Use Security:

1. If you want the ODBC data source to use database engine security, enter a Pervasive.SQL user name in the **Username** field. Enter the appropriate password in the **Password** field only if you want the password to be sent automatically whenever you connect to the database. If you leave the password blank, you will be prompted for the password upon connecting to the data source.

2. If you want to use Pervasive owner names when connecting to this data source, enter them in the **Owner Names** field. You may enter up to 8 names, separated by commas. You will be able to access any database which has an Owner Name matching one or more of the names listed.

To Select Format and Compatibility Options:

1. If your database stores characters in the OEM character set, and your application needs to manipulate data that uses the Windows character set, check the **OEM/ANSI Conversion** box. This selection causes the ODBC driver to translate the characters between the two character sets.

2. If you expect your application to use Scalable SQL-style null handling, check the **Scalable SQL Nulls** box. For more information, see “Scalable SQL-style Null Handling”.

3. If you expect your application to log into the database server using the Scalable SQL 4 features available only in Scalable SQL 4 and Pervasive.SQL, check the **Scalable SQL 4 Mode** box. If you want your application to login using only Scalable SQL 3.01 features, even if the server engine is version 4 or later, do not check the box.

4. If you plan to use the new data source exclusively with Microsoft Access, check the **Access/Jet Compatibility** box. Access is known to have problems with certain data types (notably TIME, DECIMAL, and the various NUMERIC flavors). These problems show up as #deleted errors and reports that a record has been updated by another user when it has actually not been updated by anyone else. DO NOT check this box unless you are encountering #deleted or other unexpected errors in Access.

5. If you wish to change the default formatting of dates, choose the desired format using the **Date Format** list box. This option is only available when Scalable SQL 4 Mode is checked. The options are: mdy, myd, dmy, dym, ymd and ydm.

   If the date format is set to any value other than the default (mdy), SET DATEFORMAT is called using the specified date format after a successful login to the database. You can then select, insert, or update data using the specified format. For example, if Date Format is ymd, a date field can be formatted as in the following SQL statement:

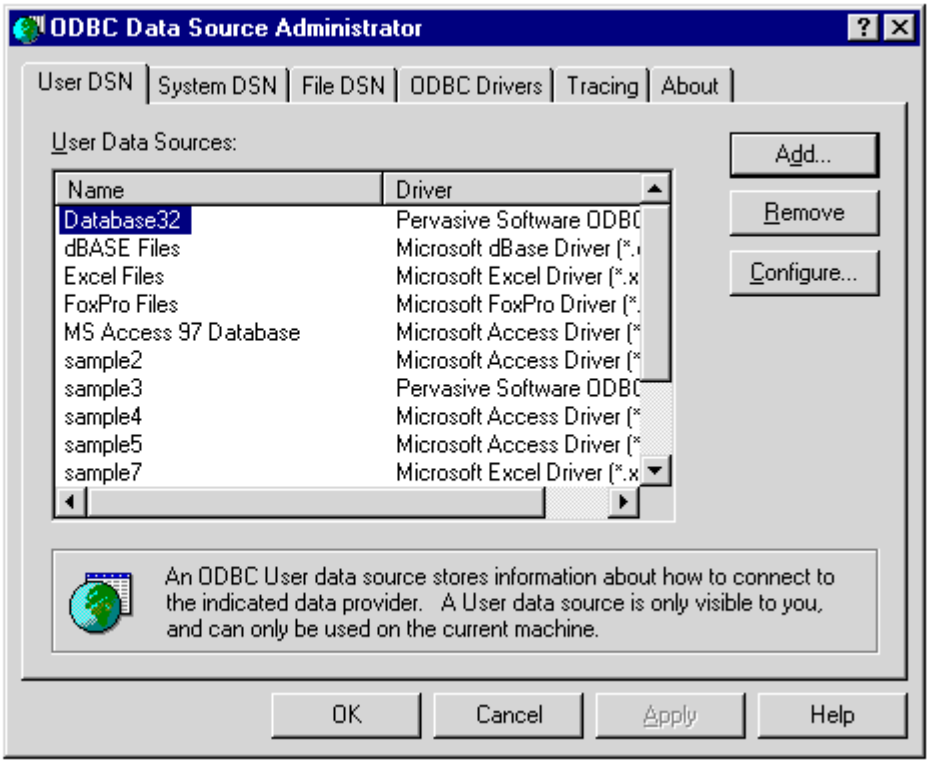
```
Select * from tablename where datecol = '1925/12/25'
```

6. If you wish to have SQL statements executed immediately after you have logged into the database, enter them in the **Login Script** box. See “Login Scripts” for further information.

Note: When adding or modifying a data source, the parameters you supply must allow a successful login to occur.

To Save Your Changes:

When you are finished, click **OK** . You should now see your new data source listed in the **Data Sources** dialog box:



7. Your ODBC data source configuration is complete. Click **OK** .

Modifying a Data Source

To change the characteristics of a data source:

1. Choose the Pervasive ODBC program group.
2. Choose the ODBC Administrator icon in Windows 3.x or the 32-bit ODBC Administrator icon in Windows NT and Windows 9X.
3. Select the desired data source in the Data Sources list and choose **Configure** .
4. In the ODBC Setup dialog, set the option values as necessary and click **OK** .

Deleting a Data Source

To delete a data source:

1. Choose the Pervasive ODBC program group.
2. Choose the ODBC Administrator icon in Windows 3.x or the 32-bit ODBC Administrator icon in Windows NT and Windows 9X.
3. Select the data source to delete in the Data Sources list, then choose **Remove** .

   A confirmation dialog box displays.
4. Click **Yes** to confirm the deletion.

Connecting to an ODBC Data Source

Once data sources are configured, an ODBC-enabled application can connect to the data source and access the data. To connect to an ODBC data source, the ODBC Interface must be installed, and one of the following must be available on your computer:

- The SQL client Requester, configured to process remote requests. Also, the corresponding SQL communications module must be loaded on the Pervasive.SQL database server.

  Refer to your Pervasive.SQL documentation for more information about configuring the Requester.
- A Pervasive.SQL workstation engine.
- The Btrieve client Requester, configured to process remote requests. Also, the corresponding Btrieve communications module must be loaded on the database server.

Many applications are set up to use ODBC data sources for a variety of purposes. Such applications are called ODBC-enabled applications. Examples include Microsoft Access and all members of the Microsoft Office family of products.

Note: Btrieve data files cannot be accessed by ODBC unless accurate Data Dictionary Files (DDFs) have been created for them. It is not enough to generate default DDFs by creating a data source. The DDFs must be customized to describe the internal record structure of your particular data file. If there are no DDFs associated with your Btrieve data file, or if you are unable to connect to your Btrieve database using ODBC, please see “Creating DDFs for Btrieve Files” . Because Btrieve files can be complex, you are not encouraged to create DDFs for existing Btrieve files unless you are an application developer.

As part of the connection process, an ODBC-enabled application may prompt you for information.

If an application prompts you for information about a data source, perform the following steps:

1. In the **User Name** box, enter your name as defined in the database (if security is enabled on the database and a name is required).
2. In the **Password** box, enter your password (if required). If you did not enter a user name, leave **Password**

blank.

3. Choose or enter a database name or enter the DDF path (and optionally, the path for the data files) for the database in which to access data. To choose a database name, click **Refresh** button to fill the list box.

4. Click **OK** .

An ODBC-enabled application must connect to a data source to access its information. Different applications connect to data sources at different times. For example, an application might connect to a data source only at your request, or it might connect automatically when it starts. For information about when an ODBC-enabled application connects to a data source, see the documentation for that application.

ODBC Driver Options

Named Databases

Database names are a feature of Pervasive.SQL. The use of database names insulates the application from the need to know the specific physical location of a database, making the database configuration easy to maintain over time. The ODBC Interface fully supports the use of database names, but provides no means for defining them.

Database names can be defined using the Setup utility that is part of your product.

Login Scripts

A login SQL script consists of one or more SQL statements, separated by semicolons, that are to be executed on connection to the data source after a successful login to the database.

This is useful, for example, to establish the global null values when the underlying data was built using null values other than the default.

This feature is especially useful for data sources to be used with third-party applications such as Borland Delphi, Visual Basic RDO, and Microsoft Access that do not offer easy ways to execute SQL statements on each connection as it is made.

In the Login Script edit control, put the SQL statement or statements you want to have executed during login to the database. If more than one SQL statement is to be executed, separate the statements with a semi-colon (;). For example:

```
SET BINARYNULL = 255;SET DECIMALNULL = ' '
```

In many cases you may wish to avoid the use of a semi-colon as statement separator. For example, you may wish to use a semi-colon within the SQL statement, or you may need to use the connect string returned from SQLDriverConnect, which uses a semi-colon to separate attribute-value pairs. In these instances, you can begin the script with any non-alphabetic character, and that character will be used as the delimiter. Use a dollar sign as the delimiter instead of the semi-colon.

For example, to connect to the sample Patients data source, and then set values for BINARYNULL and DECIMALNULL global null values on this connection, you would use the following connection string:

```
DSN=Patients;DDFPATH=C:\BTI\WIN\DEMODATA;=Smith;
PWD=Sesame;SET BINARYNULL = 255;DECIMALNULL = ';'
```

Test the effect of your script after connecting to the data source.

Scalable SQL-style Null Handling

Pervasive.SQL determines null values by examining the data for a given row and column, looking for a sentinel value in each byte of the column. For example, the default value for a column of type NUMERIC is the space character.

For most data types, there is a valid sentinel value outside the scalar range of the type. However, for a few types, such as INT(2) and INT(4), this is not the case. The default sentinel value for integer types is zero, and an integer with a zero in all bytes has the legal scalar value zero.

Some ODBC-enabled applications construct WHERE clauses that test a column value with an IS NOT NULL construct, resulting in the exclusion of rows where every byte of the test columns value is the sentinel value. This can result in the inadvertent exclusion of rows from a result set.

The ODBC Interface reports as non-nullable any data type that does not have a possible null value outside its scalar range. Nonetheless, some ODBC-enabled applications still construct tests of the type described in the preceding paragraph.

To allow these applications to operate correctly, the ODBC Interface modifies the SQL statement substituting WHERE 0 = 1 for WHERE MY_INT_COLUMN IS NULL and WHERE 1 = 1 for WHERE MY_INT_COLUMN IS NOT NULL . This behavior is applied whenever a test for null status is applied to a column which the ODBC Interface declares as not nullable.

Some programmers may want to expose the native null handling of Pervasive.SQL, which requires that the ODBC Interfaces null handling behavior be disabled. To accomplish this, set the Scalable SQL Nulls check box in the Setup or SQLDriverConnect dialog. When passing a connect string to SQLDriverConnect or SQLBrowseConnect, pass the attribute value pair NULLENABLED=yes .

ODBC and the Web

You can also use ODBC Interface 2 to connect Web applications to Pervasive.SQL via ODBC/Internet Database Connector (IDC) or JDBC-ODBC Bridge. IDC—contained in an ISAPI (Internet Server API) DLL—provides easy access to Pervasive.SQL databases through a simple scripting language on Windows NT platforms. The IDC application uses ODBC APIs to send and retrieve information between the database and the Internet on Windows NT platforms.

The JDBC-ODBC Bridge (from JavaSoft's Web site, www.javasoft.com) provides data access from Java applications and is fully compatible with ODBC Level 2 drivers. The bridge translates JDBC method calls into ODBC function calls and allows JDBC to leverage the database connectivity provided by existing ODBC technology.

Data Type Differences

Some data types are defined differently when using ODBC statements than when using Pervasive.SQL directly, such as within SQL Scope.

String Types

Applicable Types: ZSTRING, LSTRING, NOTE

ODBC definitions for these types must define a length 1 byte shorter than the native Pervasive.SQL definition. Because termination is handled transparently within ODBC, the ODBC definition should allow enough space to store exactly the number of characters that will be in the largest value. In contrast, the Pervasive.SQL definition must allow one additional byte in length for the terminator character. For example, the following two CREATE TABLE statements are equivalent:

ODBC Statement	CREATE TABLE person USING ‘person.mkd’ (FirstName zstring(15) CASE)
SQLScope Statement	CREATE TABLE person USING ‘person.mkd’ (FirstName zstring(16) CASE)

Sample Program

The following table lists the ODBC functions that the sample program calls.

Function	Description
SQLAllocEnv	Allocates an ODBC environment handle.
SQLAllocConnect	Allocates an ODBC connection handle.
SQLConnect	Connects to the data source.
SQLAllocStmt	Allocates an ODBC statement handle.
SQLSetStmtOption	Configures the statement handle.
SQLExecDirect	Executes a SQL statement.
SQLNumResultCols	Returns the number of fields in the result set.
SQLDescribeCol	Provides a description of the attributes of a field in a result set.
SQLBindCol	Binds a field in a result set to a storage location in the application's workspace.
SQLExtendedFetch	Retrieves a block of records from the result set arising from the execution of the SQL statement.
SQLError	Retrieves information about an ODBC error.
SQLFreeStmt	Frees an ODBC statement handle.
SQLDisconnect	Disconnects from the data source.
SQLFreeConnect	Frees an ODBC connection handle.
SQLFreeEnv	Frees an ODBC environment handle.

Example 2-1
C Source Code for ODBCSAMP.C continued

```
/************************************************************
**
**    Copyright 1998 Pervasive Software Inc. All Rights
** Reserved
**
************************************************************/
/************************************************************
   ODBCSAMP.C
This program demonstrates the C/C++ ODBC interface for Scalable SQL and Btrieve under MS Windows.  It uses ODBC functions to fetch records from the 'dental' database that is included with this product. IMPORTANT: Be sure to set up a data source mapped to this database; the data source name must match with the name used in the SQLConnect call. See 'IMPORTANT', below.


The ODBC Software Development Kit is required for developing ODBC-enabled applications.  It can be obtained from
```

Microsoft as part of the Developer Network subscription service. The necessary components are also shipped as part of the Visual C++ product (v1.5 or greater).

```
*************************************************************/
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/*************************************************************
The following files are found in the ODBC SDK, a Microsoft product.
*************************************************************/

#include <sql.h>    /* ODBC SQL function prototypes, part 1 */
#include <sqlext.h>    /* ODBC SQL function prototypes, part 2 */

/*************************************************************
The following pragmas allow Watcom's linker to link with the Win32 library found in the ODBC SDK.

*************************************************************/

#if defined( __WATCOMC__ ) && defined( BTI_WIN_32 )
    #pragma aux   SQLError              "*";
    #pragma aux   SQLAllocEnv           "*";
    #pragma aux   SQLAllocConnect       "*";
    #pragma aux   SQLConnect            "*";
    #pragma aux   SQLAllocStmt          "*";
    #pragma aux   SQLSetStmtOption      "*";
    #pragma aux   SQLExecDirect         "*";
    #pragma aux   SQLBindCol            "*";
    #pragma aux   SQLNumResultCols      "*";
    #pragma aux   SQLDescribeCol        "*";
    #pragma aux   SQLFetch              "*";
    #pragma aux   SQLExtendedFetch      "*";
    #pragma aux   SQLFreeStmt           "*";
    #pragma aux   SQLDisconnect         "*";
    #pragma aux   SQLFreeConnect        "*";
    #pragma aux   SQLFreeEnv            "*";
#endif
/*************************************************************
```

```
   Constants
*************************************************************/

/* ** IMPORTANT **
// You must have a data source called "BTI_ODBC_SAMPLEDB"
// configured for this sample to work. */

#define BTI_SAMPLE_DATA_SOURCE "BTI_ODBC_SAMPLEDB"

#define DOCTOR_LEN      12
#define DOCTOR_COL     1
#define PHONE_LEN     12
#define PHONE_COL     2
#define FIRST_LAST_LEN   33
#define FIRST_LAST_COL   3
#define ROWSET_SIZE     10
#define MAX_FIELD_LENGTH 44


typedef struct ColDescs
   {
      UCHAR szColName[ MAX_FIELD_LENGTH + 1 ];
      SWORD pcbColName;
      SWORD fSqlType;
      UDWORD pcbColDef;
      SWORD pibScale;
      SWORD pfNullable;
   } ColStruct;


/*************************************************************
   generic error handler
*************************************************************/

void      CheckError( RETCODE status, HENV henv, HDBC hdbc, HSTMT hstmt )
{
RETCODE errstat = SQL_SUCCESS ;
char szSqlState[ 8 ];
LONG fNativeError = 0;
static char szErrorMsg[ 256 ];
```

```
short cbErrorMsg = 0;


       /* return directly if no error has occurred. */
if ( status == SQL_SUCCESS )
 ;
else if ( status == SQL_ERROR || status == SQL_SUCCESS_WITH_INFO )
 {
 szErrorMsg[ 0 ] = 0;
 szSqlState[ 0 ] = 0;


 /* if an error or SQL_SUCCESS-with-info occurs, there may be multiple messages associated with the problem; retrieve and
print them all. */


 while ( errstat == SQL_SUCCESS )
   {
   errstat = SQLError(
         henv,
         hdbc,
         hstmt,
         szSqlState,
         &fNativeError,
         szErrorMsg,
         sizeof(szErrorMsg),
         &cbErrorMsg
         );


   printf( "*** Status: %s\nSQLState: %s\nMessage:    "
                                "%s\nNative Error: %ld\n\n",
      status == SQL_ERROR ? "Error" : "SQL_SUCCESS with info",
      szSqlState,
      szErrorMsg,
      fNativeError
      );
   } // end while loop
 } // end else if ( status == SQL_ERROR || status == SQL_SUCCESS_WITH_INFO )
else
 printf( "\nError status = %ld\n", status );


} // end CheckError
```

```
/************************************************************
  main
************************************************************/
void main( void )
{
  char userid   [] = "";
  char password[] = "";
  char header   [] = " DOCTOR          PHONE NUMBER         PATIENT";

  ColStruct * colbuf = NULL;

  SWORD     colPosition = 0;
  UWORD       fetchOption = 0;
  UDWORD   recordIndex = 0;
  char      statement[ 300 ];
  RETCODE    status = SQL_SUCCESS;
  SDWORD      statlen = 0;
  UCHAR       szDoctor[ ROWSET_SIZE ][ DOCTOR_LEN + 1 ];
  SDWORD      cbDoctor[ ROWSET_SIZE ];
  UCHAR       szPhone[ ROWSET_SIZE ][ PHONE_LEN + 1 ];
  SDWORD      cbPhone[ ROWSET_SIZE ];
  UCHAR       szFirst_Last[ ROWSET_SIZE ][ FIRST_LAST_LEN + 1 ];
  SDWORD      cbFirst_Last[ ROWSET_SIZE ];
  UDWORD      bufferLength = 0;
  SWORD       numResultCols = 0;
  UDWORD      rowlen = 0;
  UDWORD      rowsFetched = 0;
  UWORD       rowStatus[ ROWSET_SIZE ];

  /* ODBC variables. */
  HENV         henv   = SQL_NULL_HENV;
  HDBC         hdbc   = SQL_NULL_HDBC;
  HSTMT        hstmt = SQL_NULL_HSTMT;

  printf( "SQL-Level Functions Sample Program Started\n" );

  /* do ODBC standard setup */
  status = SQLAllocEnv( &henv );
  CheckError( status, henv, hdbc, hstmt );
  status = SQLAllocConnect( henv, &hdbc );
```

```
  CheckError( status, henv, hdbc, hstmt );

  /* login to ODBC data source */
  /* Dictionary and data files are in the working directory. */
  status = SQLConnect( hdbc,
         (UCHAR FAR *)BTI_SAMPLE_DATA_SOURCE,
         strlen( BTI_SAMPLE_DATA_SOURCE ),
         userid,
         SQL_NTS,
         password,
         SQL_NTS );
  CheckError( status, henv, hdbc, hstmt );

  /* ODBC driver sets blank character to underscore during connect */

  if ( status != SQL_SUCCESS && status != SQL_SUCCESS_WITH_INFO )
      printf( "SQLConnect failed\n" );

  /* allocate an ODBC statement                                    */
  if ( status == SQL_SUCCESS )
  {
      status = SQLAllocStmt( hdbc, (HSTMT FAR *)&hstmt );
      CheckError( status, henv, hdbc, hstmt );
      if ( status != SQL_SUCCESS )
          printf( "SQLAllocStmt failed, status: %d\n", status );
  } // end if ( status == SQL_SUCCESS )

  /* set appropriate statement options for a read-only block cursor. */

  SQLSetStmtOption( hstmt, SQL_CONCURRENCY, SQL_CONCUR_READ_ONLY );
  SQLSetStmtOption( hstmt, SQL_CURSOR_TYPE, SQL_CURSOR_DYNAMIC );
  SQLSetStmtOption( hstmt, SQL_ROWSET_SIZE, ROWSET_SIZE );

  /* execute SQL select statement */
  if ( status == SQL_SUCCESS )
  {
      strcpy( statement,
          "SELECT Doctor, Phone, First^Name * Last^Name FROM Patients,Appointments WHERE Patients.ID =
Appointments.ID");
```

```
    statlen = strlen( statement );
    printf( "%s\n", statement );
    status = SQLExecDirect( hstmt, statement, statlen );
    CheckError( status, henv, hdbc, hstmt );
    if ( status != SQL_SUCCESS )
        printf( "SQLExecDirect failed, status: %d\n", status );
} // end if ( status == SQL_SUCCESS )

/* allocate a buffer big enough to hold the rowset */
if ( status == SQL_SUCCESS )
{
    rowlen = 0;
    status = SQLNumResultCols( hstmt, &numResultCols );
    CheckError( status, henv, hdbc, hstmt );
    if ( status == SQL_SUCCESS )
      colbuf = malloc( numResultCols * sizeof ( ColStruct ));
    if ( colbuf == NULL )
    {
        printf( "Memory allocation failed\n" );
        status = SQL_ERROR;
    } // end if ( colbuf == NULL )

 /* get column description info into colbuf            */
    for ( colPosition = 0;
          (colPosition < numResultCols) && (status == SQL_SUCCESS);
       colPosition++ )
    {
        colbuf[ colPosition ].pcbColName = MAX_FIELD_LENGTH;
        status = SQLDescribeCol( hstmt,
            colPosition + 1,
            (UCHAR FAR *)&colbuf[ colPosition ].szColName,
            MAX_FIELD_LENGTH,
            &colbuf[ colPosition ].pcbColName,
                    &colbuf[ colPosition ].fSqlType,
                    &colbuf[ colPosition ].pcbColDef,
                    &colbuf[ colPosition ].pibScale,
                    &colbuf[ colPosition ].pfNullable );
    } // end for ( colPosition = 0; . . .
} // end if ( status == SQL_SUCCESS )
```

```
/* bind columns    */
if ( status == SQL_SUCCESS )
    {
    status = SQLBindCol( hstmt,
        DOCTOR_COL,
        colbuf[ DOCTOR_COL - 1 ].fSqlType,
                        szDoctor,
                        DOCTOR_LEN + 1,
                          cbDoctor );
    }

if ( status == SQL_SUCCESS )
    {
    status = SQLBindCol( hstmt,
        PHONE_COL,
        colbuf[ PHONE_COL - 1 ].fSqlType,
                        szPhone,
                        PHONE_LEN + 1,
                          cbPhone );
    }

if ( status == SQL_SUCCESS )
    {
    status = SQLBindCol( hstmt,
        FIRST_LAST_COL,
        colbuf[ FIRST_LAST_COL - 1 ].fSqlType,
                        szFirst_Last,
                        FIRST_LAST_LEN + 1,
                          cbFirst_Last );
    }

/* FETCH UP TO 10 RECORDS WITH A SINGLE SQLExtendedFetch() CALL */
if ( status == SQL_SUCCESS )
{
    printf( "\n%s\n", header );
    fetchOption = SQL_FETCH_FIRST;

    while ( status == SQL_SUCCESS )
    {
       /* Get buffers full of records. */
```

```
        status = SQLExtendedFetch( hstmt,
                fetchOption,
                1,
                &rowsFetched,
                rowStatus );

        if (( status !=   SQL_SUCCESS ) && ( status != SQL_NO_DATA_FOUND ))
        {
         CheckError( status, henv, hdbc, hstmt );
            printf ("SQLExtendedFetch failed, status: %d\n", status);
            break;
        }

        /* Display the records in the column buffers, one record per line. */
        for ( recordIndex = 0;
             ((recordIndex < rowsFetched) && ((status == SQL_SUCCESS) ||
             (status == SQL_SUCCESS_WITH_INFO)));
             recordIndex++ )
        {
         printf( " %-13.13s%-18.18s%s\n",
             szDoctor[ recordIndex ],
             szPhone[ recordIndex ],
             szFirst_Last[ recordIndex ]);
        }

        fetchOption = SQL_FETCH_NEXT;
      } // end while ( status == SQL_SUCCESS )
   } // if ( status == SQL_SUCCESS )

   SQLFreeStmt( hstmt, SQL_CLOSE );
   SQLDisconnect( hdbc );
   SQLFreeConnect( hdbc );
   SQLFreeEnv( henv );

   if (colbuf != NULL)
      free( colbuf );

   exit( 0 );
} // end main
```

OEM Character Translation

When storing data in or retrieving data from databases that are shared with DOS applications, Windows applications should generally use the OEM character set supplied with the underlying operating system. This is especially important in locales outside the United States.

The ODBC Interface v2.0 uses a translator DLL to translate data between the OEM character set and the ANSI character set in use in Windows v3.x, Windows 9X, and Windows NT. Use of the translator DLL is optional. It can be enabled by checking the check box entitled “OEM Character Translation” on either the data source configuration screen displayed by the ODBC Administrator programs or the login screen displayed by SQLDriverConnect.

ODBC Overview

This chapter describes using the Open Database Connectivity Applications Programming Interface (ODBC API) via the Pervasive ODBC Interface, an alternative API for Pervasive.SQL. The following topics are covered in this chapter:

SQLExtendedFetch With Relative and Absolute Positioning

The ODBC Interface v2.5 supports the SQL_FETCH_RELATIVE and SQL_FETCH_ABSOLUTE options in SQLExtendedFetch. Be aware of the following points when using these options:

- Because the underlying cursor model is dynamic, if the tables in the SELECT statement are volatile, relative and absolute positioning cannot guarantee to take you to the same row when executed a second time. This is because rows may move (or disappear from the result set) due to updates, and deleted records will also affect positioning.

- For very large result sets, these operations can be slow if the “distance” to the desired position is relatively large.

Using SQLSetPos

The ODBC Interface implementation of SQLSetPos allows you to perform positioned updates and deletions, lock and unlock rowset rows selectively, and insert rows into the table or view.
SQLSetPos operations work on any result set that does not contain columns representing aggregate functions such as MIN, MAX, and SUM, and AVG, nor any columns whose content is computed.

See the *ODBC 2.0 Programmer's Reference* from Microsoft Press for an example of how to use SQLExtendedFetch and SQLSetPos in combination. The example can be found in Chapter 22, "ODBC Function Reference," in the section on SQLSetPos.

Manual Organization

- Chapter 1—“ODBC Overview”

  This chapter describes using the Open Database Connectivity Applications Programming Interface (ODBC API) via the ODBC Interface, an alternative API for Pervasive.SQL. There’s also a description of new features and enhancements included in the latest version.

- Chapter 2—“Getting Started”

  This chapter explains how to install and configure your ODBC driver for Pervasive products.

- Chapter 3—“Creating DDFs for Btrieve Files”

  This chapter covers how to use ODBC to access existing Btrieve files.

- Chapter 4—“ODBC SQL Grammar”

  This chapter explains how to use the ODBC SQL grammar.

- Chapter 5—“Programming via ODBC”

  This chapter describes how to program using the ODBC Applications Programming Interface (API) with the ODBC Interface.

- Appendix A—“Extensions to ODBC”

  This appendix describes using the ODBC Interface to perform certain Pervasive.SQL operations that do not have direct equivalents in the Open Database Connectivity Applications Programming Interface (ODBC API).

- Appendix B—“Sample Program”

  This appendix shows how to program in the C language using the ODBC API and the ODBC Interface.

- Appendix C—“Programming Considerations”

  This appendix provides specific information on programming with the ODBC Interface in various client development environments.

The manual also includes an index. Information on status codes can be found in the CODE_MSG.HLP Windows Help file provided in the product, or in *Status Codes and Messages* or *Status Codes and Messages Quick Reference Card* . Information on SQL syntax can be found in the SQLREF.HLP (*SQL Language Reference*) Windows Help file provided in the product.

Microsoft Access

Access applications do not allow updating of rows on tables that do not have a unique index. Access 95 and Access 97 allow users to specify which columns constitute a unique identifier if an index does not exist, but a unique index should be created on these columns in advance for best performance on updates.

You must relink your attached tables following any changes to the DSN configuration. Access hard codes these settings in the table properties when linked. Access 2.0 users should apply the Access Service Pack #1 (dated 11/94). This Service Pack fixed numerous problems that affected the behavior of Access with regard to ODBC data sources.

Designing for Access

As an ODBC application connecting to Pervasive Software products, Access works best when certain guidelines are applied during the creation of your database, as follows:

- Column Names. Avoid using Pervasive Software keywords as column names. For example, the words *time* , *year* , and *date* are scalar functions in Pervasive.SQL. Using keywords as column names can result in query problems. In addition, avoid using arithmetic and illegal characters as part of the column name. When Access attaches a column name containing a forward slash (/) or plus sign (+) that has existing column names on each side of the character, it is unclear whether *name/last* is the result of column *name* divided by column *last* .

- Data Types. Access works best with string and integer data types.

- Key Columns. Microsoft recommends a single unique key column. Integer and string columns yield the best results. Access 2.0 and Microsoft Query users should consider making the key column first and ordered alphabetically. Access uses the first unique index it finds (in alphabetical order by name) as the "bookmark" source for building a keyset for the table. In addition, Microsoft recommends that users avoid multi-segmented keys on ODBC tables, because the Jet engine does not handle such keys correctly in certain instances.

- DDFs. It is important that DDFs be created according to specification. When possible, use CREATE TABLE statements through the ODBC Interface to create the data definitions.

Compatibility Problems

Most problems with Access are known limitations of the Microsoft Jet database engine technology in dealing with ODBC drivers. Refer to the Microsoft Tech Notes Q128809 and Q127096 for details. A White Paper (www.microsoft.com/odbc/wpapers/rjetwp.htm) on using ODBC data sources with Access also documents problems you may encounter working with ODBC data sources under Access.

#Name Problems

Access might return #Name in table views when encountering column names containing invalid characters or Pervasive.SQL keywords. Refer to the SSQLREF.HLP file for a list of keywords and reserved words. The following column name characters can cause Access to return a #name: back slash (\), forward slash (/), hyphen (-), NULL characters and NULL terminated strings, arithmetic operators, and unknown characters.

#Name can result when a mismatch exists between the defined column type and the actual data stored in the Btrieve data file. A problem caused by mismatched data is not always obvious. For example, a column of type CHAR filled with NULL bytes can result in a #Name return.

When a MONEY column is used as a key, the datasheet may show #Name for every cell. This is apparently due to an internal error in Access.

LVAR columns containing CHAR data and consisting of only one segment are handled as CHAR fields if the Access/Jet Compatibility mode is turned on. Under these conditions, LVAR fields can be viewed, modified, and

inserted using Access. Otherwise, Access displays *OLE Object* for LVAR columns or a *Not a valid OLE object* error when attempting to display rows from a table containing LVAR columns. Access assumes that any large binary data type (more than 255 bytes) contains an OLE object. This problem is also present when tables with LVAR columns are opened using the Microsoft Visual C++ Foundation Classes' database support or with Visual BASIC.

#Deleted Problems

The Access Jet engine may return a #Deleted for each field returned. The default query used by Access causes an internal comparison of the record set. This internal comparison is used to determine if a record has been deleted or modified from the database. The mechanism is known to work poorly for certain data types, notably TIME, MONEY, NUMERICSA and NUMERICSTS.

To solve this problem, first try setting the value of ACCESSFRIENDLY to 'Yes' in your connection string. If you still have problems, use the steps below.

Microsoft recommends using a SQL pass-through query as a workaround to this problem.

To create a SQL pass-through query:

1. In query design mode, select **SQL Specific** , then **Pass-Through** from the **Query** menu. For Access v7.0, select **Query** , then press the **New** button.
2. Accept the default of Design View.
3. Close the **Show Table** window.
4. Select **SQL Specific** from the **Query** menu and select the **Pass-Through** option.
5. Entered the desired SQL statement.

The query can be saved for future use. This method corrects virtually all display problems, but the resulting grid is not updatable. Updates must be performed using an Update Query when SQL Pass Through is used.

When a front-end ODBC application, such as Access, invokes SQLPassThrough, it means *pass the query directly to the Microsoft ODBC Driver Manager unmodified* . This functionality is often needed because front-end applications always modify their implementation of SQL grammar before handing off to the Microsoft Driver Manager. SQLPassThrough enables the "front-end" user to execute driver or back-end specific grammar constructs, which can increase query performance.

There has been confusion about the term *SQLPassThrough* . As a Pervasive Software user, you might think that *SQLPassThrough* would invoke an ODBC Interface bypass to the Pervasive.SQL interface. However, the ODBC Interface has no pass through mechanism. It simply processes ODBC APIs, mapping them to Pervasive Software product APIs. The Microsoft ODBC API begins all API calls with the letters *SQL* . Typically, when an ODBC application attaches the prefix *SQL* to a function, it intends to call the Microsoft ODBC Driver Manager.

Updating Data

When you try to update records in a table that has a MONEY key, a window displays a message indicating that someone else has changed the data and the update does not occur. In addition, Access exhibits a variety of difficulties with tables that have a unique key containing a TIME, DATE, or BFLOAT field. Problems also exist with Access's handling of NUMERICSA and NUMERICSTS keys.

An *Invalid date, time, or timestamp* error can appear when opening a table in table view. This is often the result of data stored in a null type other than the default. For example, if DATE data were stored with an assumed binary null value of 128, an error appears in Access because binary 128 is not a valid date. To resolve this problem, select the

Null Enabled checkbox, combined with a DSN Login Script of Set Binarynull=128; .

Write Conflict errors have been observed on tables containing columns that are incorrectly defined. Older Btrieve files often contain binary data that is not easily represented in a form that is accessible in relational tables. If the portion of the data record is defined as a character column, accessing the record via ODBC (e.g., as an Access linked table) may produce unexpected results, among them the *Write Conflict* error. This happens because the data in a character field is expected to be text with blank padding to the full width of the column. Problems may also occur when binary data is stored in character type fields. If the data contains a null byte, the ODBC Interface interprets character data as NULL terminated strings. Access pads the remaining data with blanks before presenting it to the user.

Inserting Data

When inserting data into a table that has an AUTOINC column, if you explicitly insert a zero as the column value, the insert succeeds, but Access shows the row as #deleted , indicating that someone else deleted the record immediately after it was inserted. What has happened is that Access queries after the insert to make sure the row is accessible, using all values explicitly entered by the user as part of the WHERE clause in the query. Because Pervasive.SQL assumes the zero AUTOINC value to mean that the column should contain a unique value that is internally generated, a query testing for this value in the column fails.

An alternative is to leave AUTOINC columns untouched when entering data for the new row. If the column value is left blank, Access does not assume a value for the column in either the insert or the subsequent attempt to retrieve the newly inserted row.

Exporting Data

When a table containing a NOTE or LVAR column is created during the Access Export operation, the NOTE or LVAR column in the resulting table has a size of 28,672.

Low Memory Condition

When Access is left open for extended periods of time, you may run out of available memory. In order to prevent this problem, create a special table in your Pervasive.SQL database, which Access can use to configure the way it uses ODBC. Name the table *MSysConf* and define in it the following columns:

Column Name	Data Type
Config	INT(2)
chValue	CHAR(255)
nValue	INT(4)
Comments	CHAR(255)

You should then insert the following two records into the table:

Config = 102 and nValue = a large value (such as 10,000)

Config = 103 and nValue = a small value (such as 5 or 10)

For more information regarding this issue, refer to pages 338 and 339 of the *Microsoft Jet Database Engine Programmer's Guide* by Microsoft Press.

Procedure

If you have not already done so, go into the ODBC Administrator and create a data source to your database using the

Pervasive ODBC Interface.

Use the Microsoft Access option **SQL Specific Pass Through** (described in the following steps) to create the **MSysConf** table in the Pervasive.SQL database.

To create the MSysConf table:

1. In the Access main database screen, from the **Query** menu, select **New** ; a dialog box with types of views is displayed.

2. Select **Design View** ; a list of available tables is displayed.

3. Click **Close** ; a blank query design window is displayed.

4. From the **Query menu** , choose **SQL Specific** , then **Pass-through** ; a blank text screen is displayed. Enter the following (including quotes, commas, etc.) in the text box:

```
CREATE TABLE MSysConf
(Config INT(2), chValue Char(255), nValue INT(4),
 Comments CHAR(255))
```

5. Run the query. You are then prompted to select the data source which you created in the previous step. A message may be displayed stating, 'Pass-through query with ReturnRecords Property set to True did not return any records'. Click **OK** if this occurs.

6. Create another new Pass-Through query using the previous steps and then enter the following statement:

```
INSERT INTO MSysConf (Config, nValue)
 VALUES (102, 10000)
 VALUES (103, 5)
```

7. Run the query, and select the data source as in Step 5.

This process creates and populates the MSysConf table. Save these queries for future reference and repeat this process for any other Pervasive.SQL databases that will be accessed from Microsoft Access. For further information on this table, search the Microsoft Access help for MSysConf or consult your Access documentation.

Status Code 833in ORDER BY Clauses

When using a Scalable SQL 3.x engine (either workstation or client/server), if you create a query using the Access Query Designer in which the results are ordered by a column that is not the unique key for the table, you receive an error message with an underlying Status Code 833.

For example, if you execute the following statements:

```
CREATE TABLE MyTest
(c1 autoinc,
c2 char(10),
c3 char (10))
WITH INDEX (c1 UNIQUE);
INSERT INTO MyTest VALUES (0, 'TEST', 'TEST');
```

And then use Access to execute the following query in the Access Query designer:

```
SELECT c2, c3 FROM Mytest ORDER BY c2;
```

You receive Status Code 833: The columns in the ORDER BY clause must be defined in the selection list if they are not defined as indexes in the database.

The error is actually not returned by processing the above query, but rather by processing a query which Access generates in preparing to execute this query. The underlying query attempts to build a keyset involving the unique key, but ordering its results using the ORDER BY clause from the original query, as in the following:

```
SELECT "MyTest.C1" FROM "MyTest" ORDER BY "C2";
```

The Scalable SQL 3.x engine returns a Status Code 833 from executing this statement. This error is passed on to the user, who assumes that it comes from executing the original query.

Following are suggested workarounds:

- Add indexes to all columns in the ORDER BY clause. For optimal performance, you should perform ORDER BY queries using indexed columns. You can create an index on a column easily, using a statement similar to this example:

```
create index MyIndex on MyTest (c2);
```

  You may run this statement as a Microsoft Access “pass through” query, as shown (for a different statement) in “Procedure” on page C-11.

- In the Access query editor, use a query that contains the DISTINCT keyword on tables that contain a unique key. For example, the following statement succeeds:

```
SELECT DISTINCT c2, c3 FROM Mytest ORDER BY c2;
```

  In contrast, the following statements fail:

```
SELECT DISTINCTROW c2, c3 FROM Mytest ORDER BY c2;
```

```
SELECT c2, c3 FROM Mytest ORDER BY c2;
```

- Perform the SELECT without the ORDER BY and then sort the results in the Access table view.

Microsoft Visual BASIC

Visual BASIC applications do not allow updating of rows on tables that do not have a unique index.

ODBC Interface 2.5 includes support for 32-bit Visual BASIC 4.0 Remote Data Objects (VB4 RDO) features. You can access VB4 RDO from within the Visual BASIC language. VB4 RDO is available only in the Win32 Visual BASIC 4.0 Enterprise Edition. RDO support is not available in the Win16 version of Microsoft Visual BASIC.

Note: Ensure that you are using the latest available version of MSRDO32.DLL (dated 6/7/96), which ships with Microsoft Visual C++ 4.2. A patch file is also available via other Microsoft support sources, including the MSBASIC forum on CompuServe. The older version of MSRDO32.DLL (dated 7/26/95), which ships with Microsoft Visual Basic 4.0, causes MONEY and other decimal data types to display incorrectly.

When using VB4 RDO, the default rowset size of 100 is adequate for most purposes, but must be set to a size that keeps the full rowset buffer from exceeding 32K. The full rowset buffer size can be calculated using the following formula:

buffer size = rowset size * (maximum record length + 2)

RDO manages the rowset as a virtual window on the result set, so as a programmer you appear to be operating on only one row at a time.

Status Code 822 or 210 can be returned if the RDO RecordSet exceeds 32KB.

Modifying DDFs to Describe Existing Data Files

The database you have created is “empty”—it does not contain any information about your Btrieve files. To tell the database about the content and structure of your Btrieve files, you must create a table definition for each Btrieve file and add that table definition to the database.

Determining the Table Definition of an Existing File

This section describes the steps for determining the table definition for an existing Btrieve data file. The next section provides instructions on how to add that table definition to the database.

There are 4 basic steps for determining the table definition for an existing Btrieve file:

- Determine what the fixed record length is and whether or not there is a variable-length field.
- Determine known field definitions from indexes. Determine known index definitions.
- Fill in field definitions for non-indexed fields.
- Translate information obtained from the above steps into a CREATE TABLE statement that can be executed by ODBC to add the table definition to the database.

The Btrieve file PERSON.MKD, which is provided with the 'demodata' sample database, will be used as an example in executing these steps.

Note: You can use DDF Ease to perform these steps. The ‘Help’ menu in DDF Ease contains a tutorial that shows you how to use DDF Ease for this purpose.

Step 1: Determine Fixed Record Length and Whether a Variable-length Field Exists

Records in a Btrieve file have a fixed-length section plus an optional variable-length field. The fixed-length portion of the record contains all of the fixed-length fields (or columns) that make up the data. The minimum size of the fixed-length record is 4 bytes, and the maximum size of the fixed-length record and variable-length field together is 32,765 bytes. A Btrieve file can only have one variable-length field if it has one, and it is always stored after the fixed-length portion of the record.

You can determine the fixed record length and whether the data file has a variable-length field by obtaining the file statistics on the Btrieve file using the Btrieve Maintenance Utility. Pervasive Software offers both a command-line and graphical version of the Btrieve Maintenance Utility. For the examples in this section, we chose to use the command-line Btrieve Maintenance Utility (BUTIL).

The following is a portion of the output produced by running BUTIL -stat on the PERSON.MKD Btrieve file.

```
BUTIL -stat person.mkd
...
Total Number of Records = 1500
Record Length = 312
Data Compression = No
Variable Records = Yes
Variable-Tail Allocation Tables = No
```

Blank Truncation = No

Free Space Threshold = 5%

...

From this output, we can determine that the fixed record length is 312 bytes and that a variable-length field is defined. This information is used in Step 2.

Step 2: Determine Known Fields and Indexes from the Btrieve File

Btrieve files contain some information about field lengths and data types, indexes, and overall record length that must match the corresponding table definition.

You can determine the index definitions, the position (that is, byte position in the record), length, and data type of the fields that are indexed using the BUTIL -stat command.

The following is a portion of the output from running BUTIL -stat on the PERSON.MKD Btrieve file.

BUTIL -stat person.mkd

Total Number of Keys = 3

Total Number of Segments = 5

```
Key         Position        Type            Null Values*        ACS
    Segment           Length          Flags               Unique
  Values
  0    1          1       8  Unsigned            --       1500    --
  1    1         25      26  Zstring     I MD    --       1498    --
  1    2          9      16  Zstring     I MD    --       1498    --
  2    1        113       3  Zstring     I MD    --        896    --
  2    2         82      31  Zstring     I MD    --        896    --
```

Legend:
< = Descending Order
D = Duplicates Allowed
I = Case Insensitive
M = Modifiable
R = Repeat Duplicate
A = Any Segment (Manual)
L = All Segments (Null)
* = The values in this column are hexadecimal.
?? = Unknown
-- = Not Specified

From the above information, we know that the file contains five indexed fields. There may be additional fields that we have not yet determined. We can also tell field characteristics such as data type, case insensitivity, duplicates allowed, and modifiable (among others).

One step in creating your table definition is naming the fields. In our example, we use column names such as: "Student_ID", "FirstName", "LastName", "PermCity", and "PermState". These names do not need to match anything specific. When you define the table, you create them for the first time. Choose names that will be significant to your application. If you do not know the contents of the fields, you can use Field1, Field2, etc. for the purposes of working out your definition terms. Create a worksheet similar to the table below.

Table 3-1
Naming the Indexed Fields

Field Name**1	Position	Length	Data Type	Case Insensitive
Student_ID	1	8	Unsigned	
FirstName	9	16	Zstring	Yes
LastName	25	26	Zstring	Yes

PermCity	82	31	Zstring	Yes
PermState	113	3	Zstring	Yes

***1** This portion was not identified with the BUTIL -stat command.

We also know that some field definitions are MISSING because there are gaps in the record positions and lengths (see Table 3-2 on page 3-7). For example, there is not a gap between the first two fields because the first field ends at 8, and the second field begins at position 9. However, the third field ends at position 50 (25 [position] + 26 [length] - 1 = 50) and the next field should begin at position 51. But the next indexed field begins at 82. From this information, we can tell that there are one or more fields in the record beginning at position 51 and ending at position 81 (total length of 31 bytes).

We also know the fixed record length is 312 and the last indexed field ends at position 115 (113 + 3 - 1 = 115). So one or more field definitions exist for positions 116 to position 312 for a length of 197 bytes. There is also a variable-length field, and the maximum size of the record is the fixed record length plus the variable-length field size. So the maximum size of the variable-length field is the maximum record length, 32765 bytes, minus the defined fixed record length for the PERSON.MKD file, 312 bytes, plus one (32765 - 312 + 1 = 32454 bytes). The field name can be defined as “Comment”. Scalable SQL or only allows a data type of LVAR or NOTE for a variable-length field.

Table 3-2
Non-Indexed Fields

Field Name	Position	Length	Data Type	Case Insensitive
	51	31	??	??
	116	197	??	??
Comment	313	Up to 32456	LVAR or NOTE	N/A

Table 3-3 shows what is known so far about the field definitions for the Btrieve file, keeping in mind that the unnamed field definitions could be broken into one or more fields.

Table 3-3
Fields Known so Far

Field Name	Position	Length	Data Type	Case Insensitive
Student_ID	1	8	Unsigned	
FirstName	9	16	Zstring	Yes
LastName	25	26	Zstring	Yes
	51	31	??	??
PermCity	82	31	Zstring	Yes
PermState	113	3	Zstring	Yes
	116	197	??	??
Comment	313	Up to 32456	LVAR or NOTE	N/A

Note: IMPORTANT: If there are ANY field overlaps in Table 3 you cannot use ODBC or Scalable SQL to create a table definition for your Btrieve file. For example, you would have a field overlap if there were an indexed field at position 5 with length of 1 or more. This would overlap the bytes in the Student_ID field, and possibly the FirstName field as well.
you have overlapping field definitions and you want to use ODBC or Scalable SQL to define a table definition, then you must:
(1) remove indexes from your Btrieve file which has indexes on overlapping positions and lengths, and
(2) modify your Btrieve application program logic to reflect the newly changed index definitions.
you must separate the overlapping fields into multiple fields that do not overlap, and define an index (or segmented index) on these fields.

The index information we know so far from the above field table and BUTIL -stat command is shown in Table 3-4 on page 3-9. Table 3-4 on page 3-9 provides an easier to read format for the index definitions for later steps.

Table 3-4 on page 3-9 uses the following conventions: An index consisting of more than one field is a segmented index. If the index consists of N fields, then segments 1 through (N - 1) will have Yes in the Segmented column. A value of No in the column “Segmented” means that the next index encountered is not part of the current index definition.

Table 3-4
Index Information

Index	Segmented	Field	Case Insensitive	Dup	Mod	NULL	Sort	ACS
0	No	Student_ID						
1	Yes	LastName	Yes	Yes	Yes			
1	No	FirstName	Yes	Yes	Yes			
2	Yes	PermState	Yes	Yes				
2	No	PermCity	Yes	Yes				

The next step is to complete the field definitions for the non-indexed fields in Table 3.

Step 3: Complete the Non-Indexed Field Definitions

In Step 2 we identified the non-indexed portions of the record, which are shown in Table 3-5 on page 3-9.

Table 3-5
Non-indexed Field Definitions

Field Name	Position	Length	Data Type	Case Insensitive
	50	31	??	??
	116	197	??	??

Comment	313	Up to 32456	LVAR or NOTE	N/A

The first two gaps can be split into one or more fields over the interval of the position and length specified in the table because they are part of the fixed record length portion of the record (i.e. the fields are not variable-length fields).

How to divide the field gaps into one or more fields depends on the context and structure of the data in the Btrieve file as defined by your application. If you are not using a DDF tool to assist you with determining how to split these fields, you must rely on your knowledge of your application's use and definition of the Btrieve record structure for the file. It is assumed that you have the latter information if you are creating a table definition using ODBC without the assistance of a DDF tool.

For the PERSON.MKD data file, the application's record structure indicates that the first field should not be split, and the field name and data type are "PermStreet" and "Zstring," respectively.

Table 3-6
Non-indexed Field Definitions for PERSON.MKD file Position 51, Length 31

Field Name	Position	Length	Data Type	Case Insensitive
PermStreet	51	31	Zstring	yes

The second field gap (position 116, length 197) should be split into subfields as follows:

Table 3-7
Non-indexed Field Definitions for PERSON.MKD file Position 116, Length 197

Field Name	Position	Length	Data Type	Case Insensitive
Perm_Zip	116	11	Zstring	Yes
Perm_Country	126	21	Zstring	Yes
Street	148	31	Zstring	Yes
City	179	31	Zstring	Yes
State	210	3	Zstring	Yes
Zip	213	11	Zstring	Yes
Phone	224	10	Numeric	
EmergencyPhone	234	20	Character	Yes
UnlistedNumber	254	1	Bit	
BirthDate	255	4	Date	
EmailAddress	259	31	Zstring	Yes
Sex	290	1	Logical	
Citizenship	291	21	Zstring	Yes
Survey	312	1	Bit	
Smoker	312	1	Bit	
Married	312	1	Bit	

Children	312	1	Bit
Disability	312	1	Bit
Scholarship	312	1	Bit

The "Comment" field is a variable-length field. The length and the data type of the "Comment" field can be defined based on knowledge of the application that uses PERSON.MKD. From our knowledge of the application, this field is defined as:

Table 3-8
Variable-length Field Definition for PERSON.MKD file

Field Name	Position	Length	Data Type	Case Insensitive
Comment	313	200	Note	N/A

Combining the completed field information of the non-indexed fields (Table 3-6 on page 3-10, Table 3-7 on page 3-10, and Table 3-8 on page 3-11) with the field information of the indexed field (Table 3-1 on page 3-7) yields the complete field table definitions:

Table 3-9
Complete Field Definition for PERSON.MKD file

Field Name	Position	Length	Data Type	Case Insensitive
Student_ID	1	8	Unsigned	
FirstName	9	16	Zstring	Yes
LastName	25	26	Zstring	Yes
PermStreet	51	31	Zstring	Yes
PermCity	82	31	Zstring	Yes
PermState	113	3	Zstring	Yes
Perm_Zip	116	11	Zstring	Yes
Perm_Country	126	21	Zstring	Yes
Street	148	31	Zstring	Yes
City	179	31	Zstring	Yes
State	210	3	Zstring	Yes
Zip	213	11	Zstring	Yes
Phone	224	10	Numeric	
EmergencyPhone	234	20	Character	Yes
UnlistedNumber	254	1	Bit	
BirthDate	255	4	Date	

EmailAddress	259	31	Zstring	Yes
Sex	290	1	Logical	
Citizenship	291	21	Zstring	Yes
Survey	312	1	Bit	
Smoker	312	1	Bit	
Married	312	1	Bit	
Children	312	1	Bit	
Disability	312	1	Bit	
Scholarship	312	1	Bit	
Comment	313	200	Note	N/A

Step 4: Translate Index and Field Information to a Create Table Statement

Now that we have the information we need about fields and indexes, we must translate that information into a format that ODBC and the SQL engine can understand. In the correct format, this information is used to create the definition. This format is a CREATE TABLE statement. See *SQL Language Reference* (online or printed) for the complete syntax used when defining fields and indexes with a CREATE TABLE statement.

We will be translating the information listed in Table 3-9 on page 3-11, which lists the field definitions, and Table 3-4 on page 3-9, which lists the index definitions into a CREATE TABLE statement.

Some rules of thumb when performing this translation can be summarized as follows:

- Do not rely on a default length for the field. Specify a length for all fields except DATE, TIME, BIT, CURRENCY, and TIMESTAMP. These fields only have one length.
- You can optionally define the number of decimal places (precision) on the data types DECIMAL, NUMERIC, NUMERICSA, and NUMERICSTS. In the CREATE TABLE statement, the syntax is (length, precision). See *SQL Language Reference* .
- Because you will execute the CREATE TABLE statement in ODBC, you must subtract one from the length of the following data types: ZSTRING, LSTRING, and NOTE. These fields have an internal storage format which adds one byte to the actual data being stored. The length you specify for the field should correspond to the actual number of characters that will be stored without regard to how the data is internally represented or terminated. If you plan to use the CREATE TABLE statement within a native SQL application instead of an ODBC application, you must provide the internal length for the field without subtracting one from the length of these types of fields.
- Only the data types CHARACTER, LSTRING, and ZSTRING can be case sensitive or case insensitive.
- For index definitions, if there is a "No" for Duplicates Allowed, you will need to add an index attribute UNIQUE. If the Sort is descending, add the index attribute DESC. If there is a "Yes" for Mod, Case, Null, or Seg, you will need to add the index attribute MOD, CASE, NULL, or SEG, respectively. The SEG attribute must be specified last.
- For index definitions, if there is an Alternate Collating Sequence (ACS) for an index or index segment, you will need to provide a path (in single quotes) to a valid ACS file after the index attributes. An example ACS file is UPPER.ALT. It does not matter if the ACS file matches the ACS in the Btrieve file because the ACS information is stored in the existing Btrieve file. Providing the ACS path is only used to set the ACS flag in the DDF table definition. You can obtain a copy of UPPER.ALT from Pervasive Software.

The CREATE TABLE statement has a USING <file> clause that allows you to specify the Btrieve file corresponding to your table. Note: You will get a Status Code 59 or a Status Code 257.

The CREATE TABLE statement requires a table name. For the PERSON.MKD file, we chose the name “person.”

Armed with this information, we will now construct the CREATE TABLE statement:

```
CREATE TABLE person USING 'person.mkd'
(
 Student_ID        unsigned(8) ,
 FirstName         zstring(15) CASE ,
 LastName          zstring(25) CASE ,
 PermStreet        zstring(30) CASE ,
 PermCity          zstring(30) CASE ,
 PermState         zstring(2)   CASE ,
 PermZip           zstring(10) CASE ,
 PermCountry       zstring(20) CASE ,
 Street            zstring(30) CASE ,
 City              zstring(30) CASE ,
 State             zstring(2)   CASE ,
 Zip               zstring(10) CASE ,
 Phone             numeric(10),
 EmergencyPhone character(20) CASE,
 UnlistedNumber Bit,
 BirthDate         Date,
 EmailAddress      zstring(30) CASE ,
 Sex               logical(1),
 Citizenship       zstring(20) CASE,
 Survey            bit,
 Smoker            bit,
 Married           bit,
 Children          bit,
 Disability        bit,
 Scholarship       bit,
 Comment           note(199)
) WITH INDEX (
 Student_ID        UNIQUE,
 LastName          CASE MOD SEG,
 FirstName         CASE MOD,
 PermState         CASE MOD SEG,
 PermCity          CASE MOD
)
```

We now have a table definition for a Btrieve data file. The next section explains how to use this statement to add the

table to the database.

How to Add a Table Definition to the Database

In the previous section we determined the table definition in terms of a CREATE TABLE statement.

To use ODBC to execute these statements, you will need to:

- Connect to a data source for the database you have defined.
- Create the table in the database using the CREATE TABLE statement defined in the last section.
- Verify that the definition is correct by calling SQLStatistics.

You can use DDF Ease or Microsoft's ODBC Test (32-bit) to execute ODBC statements. None of the above steps require coding—all can be accomplished through the graphical interface of the ODBC Administrator and/or other tools.

Connecting to a data source

To connect to a data source:

1. Create a data source for the database if one does not exist.

   If ODBC Administrator was used to create the database, a data source is already be defined. Otherwise, you need to define a data source. Define a data source using the ODBC Administrator, as shown in "Adding a Data Source".

2. You may now connect to the data source.

   In ODBC Test, from the **Connect** menu, choose **Full Connect** and select the data source you defined.

   In DDF Ease, from the **File** file menu, choose **Open Database** .

Creating the Table Definition:

To create a table definition using DDF Ease:

From the **Table** menu, choose **Create** . The Table Creation Wizard guides you through the steps to create your table.

To create a table definition using ODBC Test:

1. From the **Statement** menu, choose **SQLAllocStmt** . Click **OK** . This step allocates a statement. The new empty statement will be used by the SQLExecDirect function in the following step.
2. Execute the CREATE TABLE statement.
   a. From the **Statement** menu, choose **SQLExecDirect** to display the SQLExecDirect dialog box.

b. In the szSqlStr edit box, enter the CREATE TABLE statement exactly as you defined it in the previous section. Click **OK** .

c. If you received an Status Code of -1, from the **Misc** menu, choose **SQLError** to see the error code. Look up the error code in *Status Codes and Messages* . You probably have a syntax error in the CREATE TABLE statement or you mis-translated your table definition when analyzing the Btrieve file structure. Make the corrections to the CREATE TABLE statement and go back to step 1 (allocate a new statement). If you receive Status Codes 59 or 257, you have successfully created the table. You should now check that the table definition is correct.

Verifying That the Table Definition is Correct

There are two tasks to perform in this step:

- Verify that the index definitions in the table match the index definitions in the Btrieve file.
- Verify that the field definitions match what you expect for your data.

To verify the definition using DDF Ease:

From the **File** menu, choose **Check Database** . The Check Database Wizard guides you through the steps to check your database definitions.

To verify the index definitions using ODBC Test:

1. From the **Statement** menu, choose **SQLAllocStmt** to allocate a statement. From the **Catalog** menu, choose **SQLStatistics** .

   A dialog appears.

2. In the **szTable name** box, enter the name of the table (in this example, "person").

   For the **fUnique** field, select SQL_INDEX_ALL from the box. Click **OK** .

   If this command succeeds (SQL_SUCCESS = 0), then your definitions match. You can display them by repeating these steps and from the **Results** menu, choose **GetDataAll** .

To verify the field definitions using ODBC Test:

1. From the **Statement** menu, choose **SQLAllocStmt** to allocate a statement.

2. From the **Statement** menu, choose **SQLExecDirect** from the menu. Enter the following statement in the **szSqlStr** box (substitute your table name for “person”):

```
SELECT * from person
```

   To fetch and view all of the data at once, from the **Results** menu, choose **GetDataAll** .

   Fetching and viewing the data all at once will be slow if you have a large amount of data. To view the data one row at a time, alternately choose from the **Results** menu: **SQLFetch** followed by **GetDataRow** .

3. If the table definition is incorrect, you must delete it.

   However, you do not want to delete your data file. You must first move your data file to another directory location so that it is not deleted. You can use the following command to move it to a subdirectory named “bak”:

```
copy person.mkd bak
```

   Next, allocate a statement using the menu item Statement | SQLAllocStmt, and then choose Statement | SQLExecDirect. Enter the following statement to erase the incorrect table definition:

```
DROP TABLE person
```

   Copy your Btrieve data file back to the database directory and make adjustments as necessary to the table definition following the directions in the previous major sections.

As you can see, this can be a time consuming, detailed, and painstaking task. When you are finished, your Btrieve file(s) can be accessed by any ODBC application. Keep in mind that DDF Ease can greatly ease the burden of creating and manipulating DDFs.

New Features in ODBC Interface 2.04 and 2.5

This section outlines the functionality added in this release.

Features Added

- Version 2.5 supports Pervasive Smart Components. Version 2.04 does not support this architecture because it is intended for use with earlier versions of the product that do not use the Smart Components architecture.
- ODBC performs cache fetches if the query is read-only, forward cursor and rowset size is 1.
- Multiple paths are allowed with DATAPATH when using SQLDriverConnect.

With the release of Pervasive.SQL Workstation (ODBC Interface 2.53 and later), ODBC Interface no longer includes an embedded SQL engine. You must either use ODBC Interface with a local Pervasive.SQL Workstation, or with the SQL engine running on your Pervasive.SQL server machine.

ODBC Conformance Levels

This version of the ODBC Interface for Windows, Windows 95, and Windows NT supports ODBC Conformance Level 2. ODBC defines conformance levels for both the API and the SQL grammar. Applications written to conform to a specific level of the ODBC standard can use any interface that supports the conformance level that the application uses. Such applications are often referred to as ODBC-enabled applications. Many ODBC-enabled applications exist, including most report writers, word processors, and spreadsheet programs.

SQL Grammar Conformance

The ODBC core-level SQL grammar is, for the most part, a subset of the ODBC Interface's SQL grammar. SQL statements written to conform to ODBC SQL generally run without modification on a MicroKernel database using Pervasive.SQL.

Microsoft designed ODBC as a truly open standard, allowing full access to the power of the underlying DBMS. Consequently, if a programmer is willing to trade a degree of portability for additional functionality, it is possible to take advantage of the full spectrum of ODBC Interface SQL extensions to the ODBC SQL grammar. SQL statements written using ODBC Interface SQL syntax not supported by the ODBC SQL grammar are passed through without modification to the ODBC Interface SQL engine.

API Conformance

All ODBC Level 1 and Level 2 API functions are fully supported, with minor limitations described in later sections.

ODBC Procedure (Stored Statement) Support

The ODBC Interface supports the concept of ODBC procedures, via the equivalent of Pervasive.SQL stored statements. In ODBC terminology, a procedure is an executable entity that, when executed, can return result sets in the same manner as a SELECT statement.

A procedure, however, is much more powerful than a simple SELECT statement. It can consist of several different kinds of SQL statements, including SELECT, INSERT, UPDATE, and DELETE statements. Because it can contain multiple statements, in circumstances in which it contains multiple SELECT statements, it is capable of returning multiple result sets.

Using SQLRowCount and SQLMoreResults in Procedure Processing

When you execute a SQL INSERT, UPDATE, or DELETE statement, the ODBC Interface stores the number of rows affected and continues processing the next SQL statement, if one is available. When you execute a SELECT statement, a result set is available and execution stops until the caller decides what to do with the result set.

The application may, therefore, have both a row count and a result set available for processing at the same time. Because multiple INSERT, UPDATE, or DELETE statements might be processed in the procedure before control is returned to the caller, multiple row counts might be available. When processing stored procedures that contain a mix of SELECT and non-SELECT statements, use the following algorithm after SQLExecute or SQLExecDirect returns SQL_SUCCESS:

1. Call SQLRowCount.
2. If the value that SQLRowCount returns is zero, a result set may be available. Call SQLNumResultCols; if it returns a non-zero value for the number of columns in the result set, a result set is available.
3. If the value that SQLRowCount returns is greater than zero, or equal to zero and SQLNumResultCols returned a zero for the number of columns in the result set, it is the result of the execution of a statement that did not return a result set.
4. If a result set is available, process it using the usual ODBC functions (for example, SQLDescribeCol, SQLBindCol, and SQLFetch).
5. Call SQLMoreResults to see if another row count or result set is available.
6. If SQLMoreResults returned SQL_NO_DATA_FOUND, processing is complete.
7. If SQLMoreResults returned SQL_SUCCESS, more results are available; proceed to Step 1.
8. If SQLMoreResults returned SQL_ERROR, an error has occurred while processing a stored statement.

Limitations

Following are the limitations to using SQLRowCount and SQLMoreResults in procedure processing:

- You cannot use procedures in conjunction with parameter arrays.
- Procedures can only use input parameters. Output and input/output parameters are not supported.

ODBC SQL Grammar

This chapter explains how to use the Open Database Connectivity (ODBC) SQL grammar. The ODBC Interface supports the core SQL grammar with some extensions. Many of the vendor-specific escape sequences outlined in Appendix C of the ODBC specification are also supported. In accordance with the design of ODBC, Pervasive.SQL passes native SQL grammar to the underlying native SQL engine.

For a detailed description of the ODBC SQL grammar, see the *ODBC 2.0 Programmer's Reference* , available from Microsoft Press.

This chapter discusses the following topics:

Performing Pervasive.SQL Operations Without ODBC Equivalents

Some functions in the SQL API do not have direct equivalents in the ODBC API. These functions include XQLCallback and XQLVersion. Moreover, ODBC does not have an equivalent to the Btrieve concept of owner names on files.

You can still access these functions and enable files with owner names using extensions to the ODBC API functions SQLGetConnectOption and SQLSetConnectOption. Using these functions, you can perform the following tasks:

- Provide a list of owner names to the interface
- Retrieve version information
- Identify the current session ID
- Install or remove a callback function (Windows only)
- Convert data between internal and display formats and vice versa
- Validate a mask
- Obtain the default mask
- Validate data values

Table A-1 shows the correspondence between the ODBC data types and the corresponding C data types.

Table A-1
C Platform-Independent Data Types

ODBC Type	Windows Type
HDBC	long integer
RETCODE	long (32-bit) integer
UDWORD	long integer
UWORD	short (16-bit) integer

Using SQLSetConnectOption and SQLGetConnectOption

Table A-2 shows when to use SQLSetConnectOption and when to use SQLGetConnectOption.

Table A-2
Using SQLSetConnectOption and SQLGetConnectOption

Functionality	ODBC Function	fOption Value
Provide a list of owner names to the interface	SQLSetConnectOption	1000
Retrieve version information	SQLGetConnectOption	1001
Identify the current session ID	SQLGetConnectOption	1002

Install or remove a callback function (Windows only)	SQLSetConnectOption	1003
Convert data between internal and external formats and vice versa	SQLGetConnectOption	1004
Validates a mask for use in entering data for a field	SQLGetConnectOption	1005
Gets the default mask for use in entering data for a field	SQLGetConnectOption	1006
Validates values for a field or group of fields	SQLGetConnectOption	1007

Following is the C/C++ call specification for SQLGetConnectOption.

```
RETCODE SQLGetConnectOption( hdbc, fOption, pvParam)
HDBC    hdbc;
UWORD   fOption;
UDWORD   pvParam;
```

Following is the C/C++ call specification for SQLSetConnectOption.

```
RETCODE SQLSetConnectOption( hdbc, fOption, vParam
HDBC    hdbc;
UWORD   fOption;
UDWORD   vParam;
```

Calling SQLGetConnectOption

To call SQLGetConnectOption, prepare a buffer according to the format required by the option you wish to invoke. Call the function, passing the connection handle obtained from ODBC when allocating the connection, option number, and the address of the buffer (which must be cast to type UDWORD to avoid compiler warnings).

Parameter Summary

The parameters for the SQLGetConnectOption function are described separately for each option, because each option interprets the parameters differently.

Error Conditions

SQLGetConnectOption returns an error if the underlying SQL API function returns an error. Use the ODBC function SQLError to obtain error information, including the status code that the function in the native error code parameter returns.

SQLGetConnectOption may also return an error status if the connection is not active and a cursor ID was needed internally to complete the operation.

Calling SQLSetConnectOption

To call SQLSetConnectOption, follow these steps:

1. Prepare an input buffer according to the format required by the option you want to invoke.
2. Call the function, passing the connection handle obtained from ODBC when allocating the connection, the

option number, and the address of the input buffer, which must be cast to type UDWORD to avoid compiler warnings.

Parameter Summary

The parameters for the SQLSetConnectOption function are described separately for each option, because each option interprets the parameters differently.

Error Conditions

SQLSetConnectOption returns an error if the underlying SQL API function returns an error. Use the ODBC function SQLError to obtain error information, including the status code that the function returns in the native error code parameter.

SQLSetConnectOption may also return an error status if the connection is not active (since a cursor ID may need to be allocated internally to complete the operation).

An error is reported if the name list is incorrectly formatted when passing a list of owner names.

Option 1000: Giving the ODBC Interface a List of Owner Names

Owner names act as passwords and encryption keys in protecting access to sensitive data stored in Pervasive.SQL data files. You can also use SQLSetConnectOption to pass a list of owner names to the SQL engine for opening the underlying tables.

Parameter Summary

Table A-3 summarizes the parameters for the SQLSetConnectOption function when passing a list of owner names.

Table A-3
SQLSetConnectOption parameters

Parameter	Description	Passed by: Value
Ref. hdbc	Value that SQLAllocConnect returns.	u
fOption	The constant value 1000.	u
vParam	Address of a null-terminated string containing individual owner names separated by commas; if an owner name contains spaces, enclose it in single quotes. The contents of this buffer are not modified during the processing of the SQLSetConnectOption function.Cast this address to type UDWORD to avoid compiler warnings on the call.	u

Passing a List of Owner Names

To allow ODBC to open Pervasive.SQL files to which owner names have been applied, invoke SQLSetConnectOption with the parameters described in Table A-3.

Error Conditions

SQLSetConnectOption returns an error if the connection is not active (since a cursor ID must be allocated internally to complete the operation). An error is reported if the name list is incorrectly formatted.

Option 1001: Retrieving Version Information

Use this option to obtain version information concerning the Pervasive.SQL engine or client Requester.

Parameter Summary

Table A-4 summarizes the parameters for the SQLGetConnectOption function when retrieving version information.

Parameter		Description	Passed by: Value
Ref.	hdbc	Value that SQLAllocConnect returns.	u
	fOption	The constant value 1001.	u
	vParam	Address of a buffer. Cast this address to type UDWORD to avoid compiler warnings on the call.	u

Table A-4 describes the structure of the buffer used for obtaining version information.

Table A-4
SQLGetConnectOption Version Parameter Block Format

Size	Type	Description
4	Character	Input: Initialize to the signature “XVER.”Output: If this still contains “XVER,” version information is contained in the following data elements. If it does not contain “XVER,” these 4 bytes contain version information from an older version of the Scalable SQL engine. In this case, the first two bytes contain the version number and the next two bytes contain the revision number.
2	Integer	Set to the size of this buffer, including the 4-byte signature.
2	Integer	Option code. Set to 0 to obtain the client requester version. Set to 1 to obtain the version of the local engine. Set to 2 to obtain the version of the engine serving the session ID given in the next data element. Set to 3 to obtain the version of the engine servicing the database

		whose database name or database directory path is given in the next data element.
4	Integer (first two bytes only)	If option code is 2, pass a valid session ID. If option code is 3, pass the address of a character buffer containing a zero-terminated string that is a database name beginning with “@”, a NetWare server and volume name in the form \\server\volume:, or a database directory path.
2	Integer	Version number.
2	Integer	Revision number.
2	Integer	Product update number. A return of 65535 indicates that the product update level cannot be determined.
4	Character	Code indicating the type of the target engine or client requester. NWSV indicates Scalable SQL for Netware (Server Edition), RQST indicates Scalable SQL Requester, and WIN3 indicates Scalable SQL for Windows.

Retrieving Version Information

Invoke SQLGetConnectOption after formatting the buffer for input as described in Table A-4. The requested version information is returned in the buffer.

Error Conditions

SQLSetConnectOption returns an error if the internal call fails.

Option 1002: Identifying the Current Session ID

Normally, you do not need the session ID when using ODBC, but it is required for one of the options in retrieving version information.

Parameter Summary

Table A-5 summarizes the parameters for the SQLGetConnectOption function when obtaining the current session ID.

Table A-5

Parameter		Description	Passed by: Value
Ref.	hdbc	Value that SQLAllocConnect returns.	u
	fOption	The constant value 1002.	u
		Address of short integer into which the session ID is copied. Cast this address to type UDWORD to avoid compiler warnings on the call.	

vParam

u

SQLGetConnectOption Parameters with Session ID

Retrieving Version Information

Invoke SQLGetConnectOption passing the address of a short integer as the vParam argument. The session ID is returned in the buffer (if a session has been established).

Error Conditions

SQLGetConnectOption returns an error if a session has not been established (no connection has been made to a Pervasive.SQL data source).

Option 1003: Setting or Removing a Callback Function (Windows Only)

You can use this option with the ODBC Interface for Windows. If your application needs to yield control to other applications while making Btrieve calls, your application must define a callback function, register it with the MicroKernel, and then deregister it before terminating. This option is not valid for the ODBC Interface for Windows 95.

Parameter Summary

Table A-6 summarizes the parameters for the SQLSetConnectOption function when setting or removing a callback function.

Table A-6

Parameter		Description	Passed by: Value
Ref.	hdbc	Value that SQLAllocConnect returns.	u
		The constant value 1003.	u
	fOption		
		Address of a buffer. Cast this address to type UDWORD to avoid compiler warnings on the call.	
	vParam		

u

SQLGetConnectOption Parameters with Callback

Table A-7 describes the structure of the buffer used for setting and removing a callback function.

Table A-7
SQLSetConnectOption Callback Parameter Block Format

Size	Type	Description
2	Integer	Action number: 0 to register a callback function; 1 to remove a callback function.
2	Integer	Option; specify zero for this parameter, which is reserved for future use.
4	Pointer to callback function	Address of callback function to register or remove.
4	Pointer to callback function pointer buffer, which must be 4 bytes in size	Address of previous callback function storage buffer. For option 0, initialize the callback function storage buffer to zero and pass its address; on return it contains the address of the previous callback function (if any was registered). For option 1, pass the address of the buffer containing the value returned by the call using option 0.
4	Pointer to character buffer	The address of an application-defined structure that the task needs in the callback.
4	Pointer to character buffer, which must be 4 bytes in size	Address of pointer storage for previous callback function's application-defined buffer. For option 0, initialize this pointer to zero and pass its address, and on return it contains the address of the previous callback function's application-defined structure (if any was registered). For option 1, pass the address of the buffer containing the value returned by the call using option 0.

Installing or Removing a Callback Function

The form of the callback function and the details of its use are described in the entry for XQLCallback in the Pervasive.SQL *Programmer's Reference* . Refer to these manuals for further details.

Error Conditions

SQLSetConnectOption returns an error if the internal call fails.

Option 1004: Converting Data

Use this option to convert data between its internal format and ASCII display format and vice versa.

Parameter Summary

Table A-8 summarizes the parameters for the SQLGetConnectOption function when converting data.

Table A-8

Parameter		Description	Passed by: Value
Ref.	hdbc	Value that SQLAllocConnect returns.	u
		The constant value 1004.	u

fOption

Address of a buffer. Cast this address to type UDWORD to avoid compiler warnings on the call.

vParam

u

SQLGetConnectOption Parameters with Conversion

Table A-9 describes the structure of the data buffer used for converting data.

Table A-9
SQLGetConnectOption Convert Parameter Block Format

Size	Type	Description
2	Integer	Option number: 0 to convert from unformatted to formatted; 1 to convert from formatted to unformatted.
2	Integer	Internal data type of the value to convert.
2	Integer	Internal size of the value to convert.
2	Integer	Number of decimal places for DECIMAL, MONEY, NUMERIC, NUMERICSA, and NUMERICSTS data types. Initialize to 0 for all other data types.
2	Integer	Size of the buffer pointed to by the next data element (for option zero) or the size of the data in the buffer pointed to by the next data element (for option 1).
4	Pointer to character buffer	Address of buffer containing the unformatted value (for option 0) or the formatted value (for option 1).
4	Pointer to character buffer	Address of output buffer; for option 0, contains the formatted value. For option 1, it contains the unformatted value.
4	Pointer to character buffer	Specify the mask to use for conversion. The mask can be up to 30 characters long; if shorter than 30 characters, terminate the mask with a binary zero. To use the default mask, store a binary zero in the first byte.
2	Integer	Option 0: Specify 0 for left justification, 1 for centered, and 2 for right justification.Option 1: Specify 2 to strip leading blanks from a character data type; the data is treated as if right-justified. If you are not stripping leading blanks from the data in a character data type, specify -1. For other types, this value is ignored.

Converting Data

Invoke SQLGetConnectOption after formatting the buffer for input as described in Table A-8. The data is converted as specified.

Error Conditions

SQLGetConnectOption returns an error if the internal call fails.

Option 1005: Validate Mask

Use this option to validate a mask for use in entering data for a field.

Parameter Summary

Table A-10 summarizes the parameters for the SQLGetConnectOption function when validating a mask.

Table A-10

Parameter		Description	Passed by: Value
Ref.	hdbc	Value that SQLAllocConnect returns.	u
	fOption	The constant value 1005.	u
	vParam	Address of a buffer. Cast this address to type UDWORD to avoid compiler warnings on the call.	u

SQLGetConnectOption Parameters with Mask Validation

Table A-11 describes the structure of the buffer used for validating a mask.

Table A-11
SQLGetConnectOption Mask Validation Block Format

Size	Type	Description
2	Integer	Internal data type code.
2	Integer	Internal size of the data type in bytes.
2	Integer	Number of decimal places for the data type.
2	Integer	Initialize to the length in bytes of the mask to be validated.
4	Pointer to character buffer	The address of the character string containing the mask to validate.

Validating a Mask

Invoke SQLGetConnectOption after formatting the buffer for input. The mask is validated and a return of SQL_SUCCESS indicates the mask is valid.

Error Conditions

SQLSetConnectOption returns an error if the mask is not valid.

Option 1006: Get Default Mask

Use this option to obtain the default mask for use in entering data for a field.

Parameter Summary

Table A-12 summarizes the parameters for the SQLGetConnectOption function when obtaining the default mask.

Table A-12
SQLGetConnectOption Parameters for Get Default Mask

Parameter		Description	Passed by: Value
Ref.	hdbc	Value that SQLAllocConnect returns.	u
		The constant value 1006.	u
	fOption		
		Address of a buffer. Cast this address to type UDWORD to avoid compiler warnings on the call.	
	vParam		

u

Table A-13 describes the structure of the buffer used for obtaining the default mask.

Table A-13
SQLGetConnectOption Get Default Mask Block Format

Size	Type	Description
2	Integer	Internal data type code.
2	Integer	Internal size of the data type in bytes.
2	Integer	Number of decimal places for the data type.
2	Integer	Initialize to the number of bytes available in the buffer in which the default mask will be stored. On return, stores the length in bytes of the returned default mask.
4	Pointer to character buffer	The address of the buffer in which the character string containing the default mask should be stored.

Obtaining the Default Mask

Invoke SQLGetConnectOption after formatting the buffer for input as described in Table A-13. The returned mask

indicates the default format for the indicated data type.

Error Conditions

SQLGetConnectOption returns an error if the input parameters are not valid.

Option 1007: Validate Values

Use this option to validate a value or values for a given field or group of fields.

Parameter Summary

Table A-14 summarizes the parameters for the SQLGetConnectOption function when validating a value.

Table A-14

Parameter	Description	Passed by: Value
Ref. hdbc	Value that SQLAllocConnect returns.	u
fOption	The constant value 1007.	u
vParam	Address of a buffer. Cast this address to type UDWORD to avoid compiler warnings on the call.	u

SQLGetConnectOption Parameters when Validating

Table A-15 describes the structure of the buffer used for validating a value.

Table A-15
SQLGetConnectOption Validation Param. Block Format

Size	Type	Description
4	Pointer to short (2-byte) integer	Input: number of field names in the buffer pointed to by the next data element.Output: if an error occurs, the number of fields that were validated correctly.
4	Pointer to character buffer	Specify the names of the fields for which to validate values; terminate each field name with a binary 0. If the dictionary contains duplicate field names, qualify each duplicate field name with the appropriate table name.
2	Integer	Length in bytes of the following data element.
4	Pointer to character	Specify the field values to validate in internal format. A one-to-one

buffer	correspondence must exist between the field names in the second data element and the values in this buffer, and the size of each value must match the internal size of the field definition.

Validating a Value

Invoke SQLGetConnectOption after formatting the buffer for input as described in Table A-15. The value is validated for the indicated data type.

Error Conditions

SQLGetConnectOption returns an error if the input parameters are not valid. Use SQLError to obtain more detailed information; in particular, the native error code indicates the return from the internal call, giving the specific nature of the error.

Positioned Updates and Deletes Using SQL Statements

The ODBC Interface SQL preprocessor allows you to perform positioned updates and deletions using SQL statements with a variation on the extended SQL grammar for the verbs UPDATE and DELETE.

Rows must be selected using a select statement of the following form:

SELECT [ALL | DISTINCT] select-list FROM table-nameUPDATE OF [column-name [, column-name]...]

The FOR UPDATE OF clause is supported for compliance with the specification, but is otherwise ignored.

Before performing the select, the cursor type should be set to SQL_CURSOR_DYNAMIC and concurrency should be set to SQL_CONCUR_ROWVER using calls to SQLSetConnectOption. You may set the rowset size if you want to use a rowset larger than one row.

Fetch rows into the rowset buffer using SQLExtendedFetch. If a multi-row rowset is used and the row to be updated or deleted is not the first row in the rowset, call SQLSetPos with the SQL_POSITION option to set the current row number.

Obtain the driver-assigned cursor name using SQLGetCursorName, or set the name to a name of your own choosing using SQLSetCursorName.

On a separate statement handle, prepare and execute (using SQLPrepare and SQLExecute) or execute directly (using SQLExecDirect) one of the following types of SQL statements:

DELETE FROM table-name WHERE CURRENT OF cursor-name

UPDATE table-name column-identifier =
{constant-value | dynamic-parameter}
[, column-identifier =
{constant-value | dynamic-parameter}]...CURRENT OF cursor-name

The table name must match that used in the SELECT ... FOR UPDATE OF statement.

Limitations

- Only a single table should be used in the SELECT ... FOR UPDATE OF statement.
- The result set should not contain columns representing aggregate functions such as MIN, MAX, SUM, and AVG, nor any columns whose content is computed.
- The SET clause in the UPDATE statement should set columns in the select-list of the select statement to either a constant value or a dynamic parameter. Expressions are not supported.
- Parameter arrays are not supported.
- Data-at-execution parameters are not supported.

See the *ODBC 2.0 Programmer's Reference* from Microsoft Press for a discussion on how to use positioned updates and deletes via SQL statements. The example can be found in Chapter 22, "ODBC Function Reference," in the section on SQLSetPos.

Powersoft PowerBuilder

PowerBuilder applications do not allow updating of rows on tables that do not have a unique index.

If you are using PowerBuilder 4.x or 5.x, insert the following sections in the PBODB040.INI or PBODB050.INI file:

Under the "DBMS Driver / DBMS Settings" heading, insert:

Figure C-1
PowerBuilder INI Settings - Main

```
; Pervasive Software ODBC Interface
[BTRIEVE WINBTINT]
PBNoCatalog='YES'
PBSyntax='BTRIEVE_ODBC_SYNTAX'
PBDateTime='DEFAULT_DATETIME'
PBFunctions='MS_BTRIEVE_FUNCTIONS'
; Pervasive Software ODBC Interface
[BTRIEVE W32BTINT]
PBNoCatalog='YES'
PBSyntax='BTRIEVE_ODBC_SYNTAX'
PBDateTime='DEFAULT_DATETIME'
PBFunctions='MS_BTRIEVE_FUNCTIONS'
```

Under the "Pattern Matching Syntax" heading, insert:

Figure C-2
PowerBuilder INI Settings - Pattern Matching

```
[BTRIEVE_ODBC_SYNTAX]
AddColumn='ALTER TABLE &TableName ADD ::AddColElement[, ::AddColElement ]...'
AddColElement='&ColumnName &data type '
CreateIndex='CREATE &UNIQUE INDEX &IndexName ON &TableName (::ColumnIndex[,::ColumnIndex]...)'

ColumnIndex='&ColumnName &Descending'
CreateTable='CREATE TABLE &TableName (::ColumnElement[, ::ColumnElement]...)'
ColumnElement='&ColumnName &data type &NotNull'
DropIndex='DROP INDEX &IndexName'
DropTable='DROP TABLE &TableName'
DropView='DROP VIEW &TableName'
GrantTablePrivilege='GRANT &Privilege[,&Privilege]... ON &TableName TO
&UserName[,&UserName]...'
RevokeTablePrivilege='REVOKE &Privilege[,&Privilege]... ON &TableName FROM
&UserName[,&UserName]...'
```

Managing Rowsets

Powerbuilder returns records by using SQLExtendedFetch. Prior to fetching records, Powerbuilder applications set a rowset size of 1 to 1000 records.

The rowset size Powerbuilder applications choose varies, based on a calculation internal to Powerbuilder. Frequently, the calculated rowset size causes an overflow in one or more Pervasive data buffer settings. If this situation occurs, the application may return one of the following errors, depending on which Pervasive product is in use:

[Pervasive Software][ODBC Interface][Pervasive Software SQL Engine]Status Code: 2106

[Pervasive Software][ODBC Interface][Pervasive Software SQL Engine]Status Code: 822

[Pervasive Software][ODBC Interface][Pervasive Software SQL Engine]Status Code: 210

[Pervasive Software][ODBC Interface]Error in row. (100)

To avoid these failures, you can set the Powerbuilder rowset size so that it fits better within the Pervasive data buffer limits. The default Pervasive data buffer setting is 16KB, the maximum 32KB. You can change the client buffer setting by using Setup utility to modify the Communications Buffer Size within the Scalable SQL Requester component or the Btrieve Communications Manager component. You can change the server buffer setting by using Setup to modify the Communications Buffer Size within the appropriate database engine component. The client and server settings must be the same.

Once you have chosen your Pervasive data buffer size, you must ensure that Powerbuilder does not use rowsets too large for the data buffer. In Powerbuilder, the BLOCK parameter specifies the rowset size, or the number of rows returned in a rowset. See the Powerbuilder online help for more information about BLOCK.

You can use the formula below to calculate the largest value of BLOCK (the largest rowset size) you can use:

BLOCK = Data buffer size / (max record length + 2)

Maximum record length is the length in bytes of the longest possible record in your database.

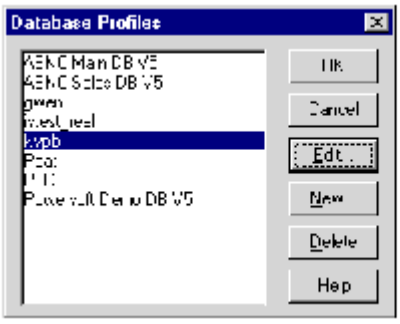
Setting the BLOCK Parameter

Before connecting to a database, you must configure the database profile in Powerbuilder.

To change the BLOCK parameter within the database profile:

1. Click the **DB Profile** icon on the Powerbuilder toolbar. You should see the **Database Profiles** window as shown in Figure C-3 on page -16.

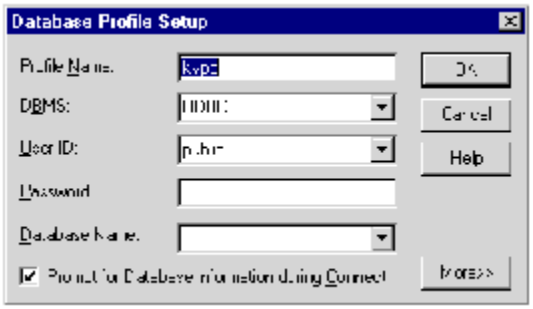
Figure C-3



Database Profiles window

2. Choose the database profile you wish to edit. Then click **Edit** to open the **Database Profile Setup** window, as shown in Figure C-4 on page -17.

Figure C-4
Database Profile Setup window

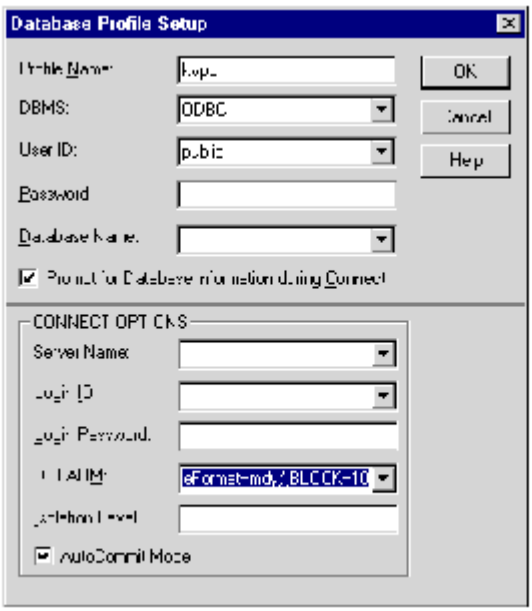


3. Click **More** . Add the following text to the **DBPARM** value: **,BLOCK=** x where *x* is the desired rowset size. See Figure C-5 on page -18.

   For example, the resulting string for DBPARM, specifying that no more than 10 rows should be returned in a data buffer, might look like this:

ConnectString='DSN=DEMODATA;=C:\PVSW\DEMODATA;DDFPATH=C:
\PVSW\DEMODATA;=no;FeaturesUsed=no;AccessFriendly=no;=mdy;',BLOCK=10

Figure C-5



DBPARM setting

Programming Considerations

The ODBC Interface 2.5 has been tested with Borland Delphi 2.0; Microsoft Visual BASIC 4.0 and 5.0; Microsoft Access 2.0, 7.0, and 8.0; Microsoft Visual FoxPro 3.0; and Powersoft PowerBuilder 5.0 (Enterprise Edition). This appendix discusses issues that developers may encounter when developing applications that use the ODBC Interface. The following topics are covered:

Programming via ODBC

This chapter describes how to program using the Open Database Connectivity Applications Programming Interface (ODBC API) with the ODBC Interface. The following topics are covered in this chapter:

SQL Preprocessor

The ODBC Interface uses a SQL preprocessor to modify ODBC-compliant SQL statements into a form that can be correctly interpreted by the underlying proprietary SQL engine.

For example, you can use quoted identifiers in accordance with the ODBC specification, and if the underlying engine is earlier than Scalable SQL 4.0, the preprocessor removes the quotes and substitutes the appropriate blank substitution character in the identifier. The NOT keyword as supported in ODBC SQL is slightly different from the support in Pervasive.SQL. The preprocessor transforms ODBC-compliant statements containing the NOT keyword into Pervasive.SQL-compliant statements.

The preprocessor also handles SQL grammar constructs used by some non-compliant front ends. For example, Microsoft Access uses the keyword DISTINCTROW in SELECT statements. This keyword is not supported either by the ODBC specification or Scalable SQL engines. The preprocessor substitutes the word DISTINCT for DISTINCTROW.

The SQL preprocessor handles ODBC-compliant grammar and a few extensions required by popular front ends, but does not recognize the full range of Pervasive.SQL extensions. If the preprocessor detects unrecognized grammar constructs, it passes the original, unmodified SQL statement to the underlying SQL engine. For this reason, you should choose between ODBC compliance and Pervasive.SQL compliance when submitting SQL statements to the ODBC Interface. If you choose Pervasive.SQL compliance, you are responsible for ensuring that all the rules of the Pervasive.SQL grammar are followed, including blank substitution in identifiers.

SQLDriverConnect Connection Strings

You can connect to a Pervasive.SQL database using the information provided during the definition of the data source, using a database name, or by specifying the path to the DDF and data files for the database. When calling SQLDriverConnect, you pass a connection string that contains the name of the data source, and optionally, additional information such as the user's login name and password, the DDF and data paths, or the database name.

Connection strings passed to SQLDriverConnect consist of keyword-value pairs joined by equal signs, with the keyword-value pairs separated by semicolons. The connection string uses the keywords listed in Table 5-2.

Table 5-2
SQLDriverConnect Connection String Keywords

Keyword	Description
DSN	The name of the data source (required).
DB	A Pervasive.SQL database name.
DDFPATH	Directory path containing the dictionary files for a Pervasive.SQL database (if this is not a named database).
DATAPATH	Directory path containing the data files for a Pervasive.SQL database (if different from the dictionary path, and if this data source does not use a named database). You may specify:- NetWare-style paths - A single long path - Multiple short paths separated by commasNOTE: Multiple long paths are not supported.
UID	The user login ID.
PWD	The user-specified password.
LOGINSCRIPT	Specific SQL statements that should be executed upon connection to the database, immediately following a successful login. See below for more information.
NULLENABLED	Range: Yes|No. Default: No.Set this variable to 'Yes' if you wish to use Pervasive.SQL-style null handling. The default value of 'No' specifies traditional ODBC-style null handling. See below for additional information.
ACCESSFRIENDLY	Range: Yes|No. Default: No.Set this variable to 'Yes' if you wish to enable the Access/Jet Compatibility mode.
FEATURESUSED	Range: Yes|No. Default: No.Set to 'Yes' to enable Pervasive.SQL 4 mode connection (allowed for Scalable SQL 4 and Pervasive.SQL databases only). The default value of 'No' causes the connection to be made in Scalable SQL 3-compliant mode regardless of the version of the target engine.
DATEFORMAT	Range: mdy|myd|dym|dmy|ymd|ydm. Default: mdy.Set to the date format desired. This keyword is only available if using a Scalable SQL 4or Pervasive.SQL data base and FEATURESUSED is set to 'Yes'. See below for more information.
CREATEDDF	Range: 0|1|2. Default: empty.This keyword allows you to create, replace, or remove a DDF. Zero '0' signifies Create, '1' specifies Replace, and '2' specifies Remove. See below for more information.

Named Database Example

To connect to the Patients data source (which is the named database 'Patients') with the login ID Smith and the password Sesame using Pervasive.SQL, use the following connection string. (The '@' character prepended to the database name is optional; if missing, it is added dynamically before the internal call to XQLLogin.)

DSN=Patients;DB=@Patients;UID=Smith;PWD=Sesame

DDF Path Example

To connect to the Patients data source (which is located in the directory C:\PVSW\DEMODATA for the ODBC Interface for Windows, and in the directory C:\ProgramFiles\BTI\WIN32\DEMODATA for the ODBC Interface for Windows NT/Windows 95) with the login ID Smith and the password Sesame using Pervasive.SQL, use the following connection strings.

For Windows 3.x:

DSN=Patients;DDFPATH=C:\PVSW\DEMODATA\ODBC; UID=Smith;PWD=Sesame

For Windows NT and Windows 95:

DSN=Patients;DDFPATH=C:\PVSW\DEMODATA\ODBC;UID=Smith;=Sesame

Note: Spaces are not supported as part of long path names under Windows NT or Windows 95.

DATAPATH Keyword

This variable specifies the directory path containing the data files for a Pervasive.SQL database, if the data files path is different from the dictionary path and if this data source does not use a named database.

To provide multiple data file paths, use a comma separated list, as in the following example:

DATAPATH=d:\data1,d:\data2,d:\data3;

Note: Although the ODBC Administrator uses semicolons to separate data paths, the semicolon separator cannot be used with the DATAPATH keyword, because it already has meaning as the statement separator.

LOGINSCRIPT Keyword

This variable specifies whether a login script should be run on successful connection to the database. A login SQL script consists of one or more SQL statements, separated by semicolons. You specify the SQL statements using the Login Script edit control.

This feature can be used, for example, to establish global null values when the underlying data was built using null values other than the default.

This feature is especially useful for data sources to be used with third-party applications (such as Borland Delphi, Visual Basic RDO, and Microsoft Access) that do not offer easy ways to execute SQL statements on each connection.

In the Login Script edit control, enter the SQL statement or statements you want to have executed during login to the database. If more than one SQL statement is to be executed, separate the statements with a semi-colon (;). For example:

LOGINSCRIPT=SET BINARYNULL = 255;SET DECIMALNULL = ' '

In some cases you may wish to use a different statement separator. For example, you may need to use a semi-colon within the SQL statement, or you may need to use the connect string returned from **SQLDriverConnect** , which uses a semi-colon to separate attribute-value pairs. In these instances, you can begin the Login Script with any non-alphabetic character, and that character will be used as the delimiter.

For example, to connect to the Patients data source as described above and then set values for BINARYNULL and DECIMALNULL global null values on this connection, you could use the following connection string (which sets the Login Script delimiter to he dollar sign ($):

DSN=Patients;DDFPATH=C:\BTI\WIN\DEMODATA;UID=Smith;=Sesame;LOGINSCRIPT=$SET BINARYNULL = 255$SET DECIMALNULL = ';'

Test the effect of your script after connecting to the data source.

NULLENABLED Keyword

Pervasive.SQL determines null values by searching for a sentinel value in each byte of the column. For example, the default NULL value for a column of type NUMERIC is the space character.

For most data types, there is a valid sentinel value outside the scalar range of the type. However, for a few types, such as INT(2) and INT(4), this is not the case. The default sentinel value for integer types is zero, and an integer with a zero in all bytes has the legal scalar value zero.

Some ODBC-enabled applications construct WHERE clauses that test a column value with an IS NOT NULL construct, resulting in the exclusion of rows where every byte of the test columns value is the sentinel value. This can result in the inadvertent exclusion of rows from a result set.

The ODBC Interface reports as non-nullable any data type that does not have a possible null value outside its scalar range. Nonetheless, some ODBC-enabled applications still construct tests of the type described previously.

To allow these applications to operate correctly, the ODBC Interface modifies the SQL statement substituting WHERE 0 = 1 for WHERE MY_INT_COLUMN IS NULL and WHERE 1 = 1 for WHERE MY_INT_COLUMN IS NOT NULL. This behavior is applied whenever a test for null status is applied to a column which the ODBC Interface declares as not nullable.

Some programmers may want to expose the native null handling of Pervasive.SQL, which requires that the ODBC Interfaces null handling behavior be disabled. To accomplish this, set the **Scalable SQL Nulls** check box in the Setup dialog. When passing a connect string to SQLDriverConnect or SQLBrowseConnect, pass the attribute value pair ‘NULLENABLED=yes’.

ACCESSFRIENDLY Keyword

Setting this flag to ‘Yes’ turns on a special Access/Jet Compatibility mode that provides additional interoperability with Microsoft Access. This flag is normally set by checking the Access/Jet Compatibility box on the Setup Data Sources screen for a data source to be used with MS Access.

Access and the Jet engine behave best with String datatypes (CHAR, ZSTRING) and INT.

Access is known to have problems with certain data types:

- TIME (used as a key)
- Numerics (NUMERIC, NUMERICSA, NUMERICSTS) with precision greater than 9.

- DECIMAL (used as a key)
- LVAR
- Float & Bfloat (with precision greater than 7)

These Access/Jet engine problems show up as '#deleted' errors, and Access reports that a record has been updated by another user when it has actually not been updated by anyone else.

When ACCESSFRIENDLY mode is set, the Pervasive ODBC Interface takes special steps to ensure proper interpretation of the data types that can cause problems with Access. This flag should only be set for data sources that display '#deleted' for each field, and should only be used when Access or the Jet engine is used to view or update the data source.

This Jet compatibility mode may require some workarounds at the front-end level. In the case of Excel, you can multiply strings by calling a "string to Number" function. The same is true with Access. A front-end mask can be applied so that the "string" with a number value will display and behave as a number.

Microsoft has not indicated any plans to fix this problem.

DATEFORMAT Keyword

If the date format is set to any value other than the default (mdy), SET DATEFORMAT is called for the specified date format after a successful login to the database. This supports dates using the date format specified for updates and inserts. For example, if Date Format is ymd, a date field can be formatted as in the following SQL statement:

```
Select * from tablename where datecol = '1925/12/25'
```

CREATEDDF Keyword

The CREATEDDF keyword is used to create, replace or remove a Data Dictionary File. This keyword can only be used with SQLDriverConnect and SQLBrowseConnect and requires the DRIVER and DDFPATH keywords. Note that a data source is not used with this keyword. When specifying CREATEDDF=2 to remove a data dictionary, the DDF files are deleted, no connection is made, and the return code is -1, because no connection occurred. (Similarly, SQLError returns a native code of 0.) An error is returned if the directory specified by DDFPATH does not exist, or if CREATEDDF is used to create a data dictionary and DDF files already exist in the specified directory.

To create a data dictionary in the d:\datadict directory, using the Win32 Pervasive Software ODBC Interface, you would use the following connection string:

```
DRIVER={Pervasive Software ODBC-32};=d:\datadict;CREATEDDF=0;
```

To create a data dictionary in the d:\datadict directory, with data in the d:\data directory, using the Win32 Pervasive Software ODBC Interface, you would use the following connection string:

```
DRIVER={Pervasive Software ODBC-32};DDFPATH=d:\datadict; DATAPATH=d:\data;CREATEDDF=0;
```

Invoking Driver-specific Features using SQLDriverConnect

You can use SQLDriverConnect to invoke features specific to Pervasive's ODBC Interface.

Braces "{}" are mandatory when used with SQL_DRIVER_NONPROMPT.

The semi-colon ";" is also necessary.

Following are some examples of invoking various features.

Basic set-up

DRIVER={Pervasive Software ODBC-32};=USER;PWD=PASSWORD;DATAPATH=L:\Bench;DDFPATH=L:\bench;

Providing a Database Name

DRIVER={Pervasive Software ODBC-32};=USER;PWD=PASSWORD;DB=@DBASE;

Invoking Other Features

This example invokes Pervasive.SQL null-handling, Microsoft Access compatibility, and Pervasive.SQL features.

DRIVER={Pervasive Software ODBC-32};=D:\btissql;DDFPATH=D:\btissql;NullEnabled=yes;=yes;featuresused=yes;

For more information on the context of using these features, refer to the *ODBC 2.0 Programmer's Reference* , p. 274.

To activate the translation DLL, you can call SQLSetConnectOption after SQLDriverConnect, as shown in the following example.

- SQLDriverConnect:
    - In:

hdbc=0x00821AE8, hwnd=VALID,szConnStrIn="DRIVER={Pervasive Software ODBC-32};DATAPATH=D:\btissql;...", cbConnStrIn=127, szConnStrOut=VALID, cbConnStrOutMax=300, pcbConnStrOut=VALID, fDriverCompletion=SQL_DRIVER_NOPROMPT=0

    - Return:

SQL_SUCCESS=0

    - Out:

szConnStrOut="DRIVER={Pervasive Software ODBC-32};DATAPATH=D:\btissql...", *pcbConnStrOut=127

- SQLSetConnectOption:
    - In:

hdbc=0x00821AE8,fOption=Conn: SQL_TRANSLATE_DLL=106, vParam="", Parameter Type=SQL_C_CHAR=1

    - Return:

SQL_SUCCESS=0

For more information on this programming context, refer to the *ODBC 2.0 Programmer's Reference* , p.450.

SQLGetInfo Return Values

Table 5-1 lists the C language #define directives for the fInfoType argument and the corresponding values that SQLGetInfo returns as Pervasive.SQL processes them. An application can retrieve this information by passing the constants (listed in the first column in Table 5-1) to SQLGetInfo in the fInfoType argument.

Note: For some fInfoType values listed below that refer to maximum values (such as for SQL_MAX_TABLES_IN_SELECT), the value zero is returned when the actual value is unlimited or variable depending on system resource availability. Consult the *ODBC 2.0 Programmer's Reference* from Microsoft for explanations of the various fInfoType values.

Table 5-1
SQLGetInfo Return Values

fInfoType Value (#define)	Returned Value
SQL_ACTIVE_CONNECTIONS	Returns a number that depends on Pervasive.SQL Setup and available memory, or that depends on BTI.INI settings and available memory.
SQL_ACTIVE_STATEMENTS	Always returns 0 as there is no specified limit.
SQL_DATA_SOURCE_NAME	Returns the current data source name.
SQL_DRIVER_HDBC	Handled by the Driver Manager.
SQL_DRIVER_HENV	Handled by the Driver Manager.
SQL_DRIVER_HSTMT	Handled by the Driver Manager.
SQL_DRIVER_NAME	Returns "WINBTINT.DLL" for the ODBC Interface for Windows or "W32BTINT.DLL" for the ODBC Interface for Windows NT/Windows 95.
SQL_DRIVER_VER	Returns the current version string.
SQL_FETCH_DIRECTION	Returns the following: SQL_FD_FETCH_NEXT | SQL_FD_FETCH_FIRST | SQL_FD_FETCH_LAST | SQL_FD_FETCH_PRIOR | SQL_FD_FETCH_ABSOLUTE | SQL_FD_FETCH_RELATIVE | SQL_FD_FETCH_BOOKMARK.
SQL_ODBC_API_CONFORMANCE	Returns SQL_OAC_LEVEL1 to indicate level 1 conformance.
SQL_ODBC_VER	Handled by the Driver Manager.
SQL_ROW_UPDATES	Returns N.
SQL_ODBC_SAG_CLI_CONFORMANCE	Returns SQL_OSCC_COMPLIANT to indicate that a driver is SAG compliant.

SQL_SERVER_NAME	Returns an empty string.
SQL_SEARCH_PATTERN_ESCAPE	Returns \.
SQL_ODBC_SQL_CONFORMANCE	Returns SQL_OSC_CORE for core SQL.
SQL_DATABASE_NAME	Returns an empty string.
SQL_DBMS_NAME	Returns “BTRIEVE.”
SQL_DBMS_VER	Returns a string representing the current version of Pervasive.SQL on this connection (e.g., "03.01.0935"). The actual version is determined dynamically on request
SQL_ACCESSIBLE_TABLES	Returns N.
SQL_ACCESSIBLE_PROCEDURES	Returns N.
SQL_PROCEDURES	Returns Y.
SQL_CONCAT_NULL_BEHAVIOR	Returns SQL_CB_NULL to indicate that the result is a concatenation of non-NULL value columns.
SQL_CURSOR_COMMIT_BEHAVIOR	Returns SQL_CB_PRESERVE.
SQL_CURSOR_ROLLBACK_BEHAVIOR	Returns SQL_CB_PRESERVE.
SQL_DATA_SOURCE_READ_ONLY	Returns N.
SQL_DEFAULT_TXN_ISOLATION	For the ODBC Interface for Windows, returns a value based on the transaction isolation level specified in Pervasive.SQL Setup. For the ODBC Interface for Windows NT/Windows 95, returns a value based on the transaction isolation level specified in BTI.INI [Scalable SQL Engine] “isolation level” section. A value of “cs” in that section causes the return value SQL_TXN_REPEATABLE_READ; a value of “ex” (exclusive) results in a return of SQL_TXN_SERIALIZABLE.
SQL_EXPRESSIONS_IN_ORDERBY	Returns N.
SQL_IDENTIFIER_CASE	Returns SQL_IC_MIXED.
SQL_IDENTIFIER_QUOTE_CHAR	Returns a double-quote character.
SQL_MAX_COLUMN_NAME_LEN	Returns 20.
SQL_MAX_CURSOR_NAME_LEN	Returns 18.
SQL_MAX_OWNER_NAME_LEN	Returns 0 to indicate that owner names are not supported.
SQL_MAX_PROCEDURE_NAME_LEN	Returns 30.
SQL_MAX_QUALIFIER_NAME_LEN	Returns 0 to indicate that qualifiers are not supported.
SQL_MAX_TABLE_NAME_LEN	Returns 20.
SQL_MULT_RESULT_SETS	Returns Y.
SQL_MULTIPLE_ACTIVE_TXN	Returns Y.

SQL_OUTER_JOINS	Returns Y.
SQL_OWNER_TERM	Returns an empty string.
SQL_PROCEDURE_TERM	Returns “Stored Statements”.
SQL_QUALIFIER_NAME_SEPARATOR	Returns “.”
SQL_QUALIFIER_TERM	Returns an empty string.
SQL_SCROLL_CONCURRENCY	Returns the following: SQL_SCCO_OPT_ROWVER | SQL_SCCO_OPT_VALUES | SQL_SCCO_READ_ONLY | SQL_SCCO_LOCK.
SQL_SCROLL_OPTIONS	Returns the following: SQL_SO_KEYSET_DRIVEN | SQL_SO_DYNAMIC | SQL_SO_FORWARD_ONLY.
SQL_TABLE_TERM	Returns “Table”.
SQL_TXN_CAPABLE	Returns SQL_TC_DML to indicate that transactions are supported, but for DML statements only.
SQL_USER_NAME	Returns the login name of the current user, or an empty string if the user is not available.
SQL_CONVERT_FUNCTIONS	Returns 0.
SQL_NUMERIC_FUNCTIONS	Returns the following: SQL_FN_NUM_ROUND | SQL_FN_NUM_TRUNCATE.
SQL_STRING_FUNCTIONS	Returns the following: SQL_FN_STR_CONCAT | SQL_FN_STR_LEFT | SQL_FN_STR_LTRIM | SQL_FN_STR_LENGTH | SQL_FN_STR_LCASE | SQL_FN_STR_RIGHT | SQL_FN_STR_RTRIM | SQL_FN_STR_SUBSTRING | SQL_FN_STR_UCASE.
SQL_SYSTEM_FUNCTIONS	Returns 0.
SQL_TIMEDATE_FUNCTIONS	Returns the following: SQL_FN_TD_CURTIME | SQL_FN_TD_HOUR | SQL_FN_TD_MINUTE | SQL_FN_TD_SECOND | SQL_FN_TD_DAYOFWEEK | SQL_FN_TD_CURDATE| SQL_FN_TD_MONTH | SQL_FN_TD_DAYOFMONTH | SQL_FN_TD_YEAR.If the Scalable SQL 4 or Pervasive.SQL Engine is in use on the connection, SQL_FN_TD_NOW is also returned.
SQL_CONVERT_BIGINT	Returns 0.
SQL_CONVERT_BINARY	Returns 0.
SQL_CONVERT_BIT	Returns 0.
SQL_CONVERT_CHAR	Returns 0.
SQL_CONVERT_DATE	Returns 0.

SQL_CONVERT_DECIMAL	Returns 0.
SQL_CONVERT_DOUBLE	Returns 0.
SQL_CONVERT_FLOAT	Returns 0.
SQL_CONVERT_INTEGER	Returns 0.
SQL_CONVERT_LONGVARCHAR	Returns 0.
SQL_CONVERT_NUMERIC	Returns 0.
SQL_CONVERT_REAL	Returns 0.
SQL_CONVERT_SMALLINT	Returns 0.
SQL_CONVERT_TIME	Returns 0.
SQL_CONVERT_TIMESTAMP	Returns 0.
SQL_CONVERT_TINYINT	Returns 0.
SQL_CONVERT_VARBINARY	Returns 0.
SQL_CONVERT_VARCHAR	Returns 0.
SQL_CONVERT_LONGVARBINARY	Returns 0.
SQL_TXN_ISOLATION_OPTION	Returns the following: SQL_TXN_READ_COMMITTED | SQL_TXN_SERIALIZABLE.
SQL_ODBC_SQL_OPT_IEF	Returns Y.
SQL_CORRELATION_NAME	SQL_CN_ANY.
SQL_NON_NULLABLE_COLUMNS	SQL_NNC_NULL.
SQL_DRIVER_ODBC_VER	Returns the version of ODBC under which the Interface was compiled.
SQL_LOCK_TYPES	Returns SQL_LCK_NO_CHANGE | SQL_LCK_EXCLUSIVE | SQL_LCK_UNLOCK.
SQL_POS_OPERATIONS	Returns SQL_POS_POSITION | SQL_POS_REFRESH | SQL_POS_UPDATE | SQL_POS_DELETE | SQL_POS_ADD.
SQL_POSITIONED_STATEMENTS	Returns SQL_PS_POSITIONED_DELETE | SQL_PS_POSITIONED_UPDATE | SQL_PS_SELECT_FOR_UPDATE.
SQL_GETDATA_EXTENSIONS	Returns SQL_GD_ANY_COLUMN | SQL_GD_ANY_ORDER | SQL_GD_BLOCK.
SQL_BOOKMARK_PERSISTENCE	Returns SQL_BP_CLOSE | SQL_BP_DROP | SQL_BP_SCROLL | SQL_BP_TRANSACTION | SQL_BP_UPDATE | SQL_BP_OTHER_HSTMT.
SQL_STATIC_SENSITIVITY	Returns 0.

SQL_FILE_USAGE	Returns SQL_FILE_NOT_SUPPORTED.
SQL_NULL_COLLATION	Returns SQL_NC_LOW.
SQL_ALTER_TABLE	Returns the following: SQL_AT_ADD_COLUMN | SQL_AT_DROP_COLUMN.
SQL_COLUMN_ALIAS	Returns N.
SQL_GROUP_BY	Returns SQL_GB_GROUP_BY_EQUALS_SELECT.
SQL_KEYWORDS	Returns the current list of reserved keywords for the SQL engine (See Appendix B, “Pervasive.SQL Keywords” for a complete listing).
SQL_ORDER_BY_COLUMNS_IN_SELECT	Returns Y.
SQL_OWNER_USAGE	Returns 0.
SQL_QUALIFIER_USAGE	Returns 0.
SQL_QUOTED_IDENTIFIER_CASE	Returns SQL_IC_MIXED.
SQL_SPECIAL_CHARACTERS	Returns #$^.
SQL_SUBQUERIES	Returns SQL_SQ_CORRELATED_SUBQUERIES.
SQL_UNION	Returns the following: SQL_U_UNION | SQL_U_UNION_ALL.
SQL_MAX_COLUMNS_IN_GROUP_BY	Returns 0, meaning no maximum is specified.
SQL_MAX_COLUMNS_IN_INDEX	Returns 0, meaning no maximum is specified.
SQL_MAX_COLUMNS_IN_ORDER_BY	Returns 0, meaning no maximum is specified.
SQL_MAX_COLUMNS_IN_SELECT	Returns 0, meaning no maximum is specified.
SQL_MAX_COLUMNS_IN_TABLE	Returns 32767.
SQL_MAX_INDEX_SIZE	Returns 255.
SQL_MAX_ROW_SIZE_INCLUDES_LONG	Returns N.
SQL_MAX_ROW_SIZE	Returns 4088. The actual value may vary depending on configured sizes for the communications buffer for both the client requester and the server engine.
SQL_MAX_STATEMENT_LEN	Returns 32767.
SQL_MAX_TABLES_IN_SELECT	Returns 20 for Scalable SQL 3.01 connections, and zero for Scalable SQL 4.0 or Pervasive.SQL connections. Zero is used to indicate that there is no fixed limitation, but the actual number is dependent on available memory.
SQL_MAX_USER_NAME_LEN	Returns 30.
SQL_MAX_CHAR_LITERAL_LEN	Returns 255.
SQL_TIMEDATE_ADD_INTERVALS	Returns 0.

SQL_TIMEDATE_DIFF_INTERVALS	Returns 0.
SQL_NEED_LONG_DATA_LEN	Returns N.
SQL_MAX_BINARY_LITERAL_LEN	Returns 255.
SQL_LIKE_ESCAPE_CLAUSE	Returns Y.
SQL_QUALIFIER_LOCATION	Returns 0.

Sample Program

This appendix shows how to program in the C language using the ODBC API and the ODBC Interface. The following topics are covered in this appendix:

Specifying your Configuration

This section covers how to set up ODBC Interface for your particular client/server configuration. All configuration changes are performed using the Setup Utility, which you can start from the Pervasive.SQL program group. You may need to use the Win16 version or the Win32 version, as noted in the following section.

Note: The default installation of ODBC is properly configured with the most commonly used setup parameters. Do not use the procedures outlined here unless you are certain you need to change your existing configuration.

It may help you to remember that the ODBC driver is, itself, a Pervasive.SQL application, so if you know how to configure Pervasive.SQL applications, you know how to configure the ODBC driver.

Windows 3.x Applications on Windows 9X/NT Systems

If you want to run Win16 ODBC applications on Win32 systems, there are a few guidelines to take into account.

Note: This section applies to Pervasive.SQL Server only.

Thunking

Pervasive recommends that you use “thunking” at the Microsoft ODBC driver manager level. “Thunking” refers to the translation of Win16 calls to Win32 calls. To do so, you should have both Microsoft Win16 and Microsoft Win32 ODBC installed. Once this step is complete, thunking will occur transparently and automatically.

If you do not install both Microsoft ODBC products on your Windows 9X or Windows NT computer, you must use a different method of thunking.

To turn on Pervasive thunking:

1. Using Setup (Win16), choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.

2. Choose **Use Thunk** under Settings. Set the value to “On.”

3. Make sure the **Local Usage** and **Remote Usage** settings are both “Off.”

   These settings are preferred if thunking is on, because thunking means you want to use the Win32 requester, not the Win16 requester. By disabling Win16 Remote and Local Usage, you are ensuring that only the Win32 requester gets loaded into memory.

4. Using the other Setup program, Setup (Win32), choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.

5. Under Settings, set **Local Usage** to “On” if you wish to use a local Win32 workstation engine. Or, set **Remote Usage** to “On” if you wish to connect to a remote server engine.

   If you have both settings set to “On,” the Requester will always try to find the server first, and then fall back to the local engine if no server can be found.

6. Click **Save** , and click **Exit** .

Using a local Btrieve workstation engine

This section describes how to configure your client if your 16-bit ODBC application uses the Btrieve workstation engine.

If you wish to run a Win16 ODBC application directly against a local Btrieve workstation engine, perform the following steps:

1. Using Setup (Win32), choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories. If this Component is not available, skip to step 4.
2. Choose **Local Usage** under Settings. Set the value to “Off.”
3. Choose **Remote Usage** under Settings. Set the value to “Off.”
4. Under Component, choose **MicroKernel Router (Win32)** and under Categories, choose **Access Control** .
5. Under Settings, choose **Local** . Set the value to “On.”
6. Under Settings, choose **Requester** . Set the value to “Off.”
7. Click **Save** , and click **Exit** .
8. Using Setup (Win16), choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.
9. Choose **Local Usage** under Settings. Set the value to “On.”
10. Choose **Remote Usage** under Settings. Set the value to “Off.”
11. Choose **Use Thunk** under Settings. Set the value to “Off.”
12. Under Component, choose **MicroKernel Router (Win16)** , and under Categories, choose **Access Control** .
13. Under Settings, choose **Local** . Set the value to “Off.”
14. Under Settings, choose **Requester** . Set the value to “Off.”
15. Choose **Use Thunk** under Settings. Set the value to “On.”
16. Click **Save** , and click **Exit** .

Selecting Your Remote Access Path

Note: With the release of Pervasive.SQL Workstation and Pervasive.SQL Server Service Pack 2 (ODBC Interface 2.53 and later), ODBC Interface no longer includes an embedded SQL engine. You must either use ODBC Interface with a local Pervasive.SQL Workstation, or with the SQL engine running on your Pervasive.SQL server machine. If you are using either of these product versions,

this section does not apply to you

By default, the Typical client installation configures your ODBC client to connect to a remote Pervasive.SQL server engine. If you are not running the SQL server engine, the default configuration “falls back” to the Btrieve server engine. In some cases you may wish to prevent this fallback, oryou may want to require that an ODBC application works directly with Btrieve without a SQL server engine. This section explains how you can specify these configurations.

ODBC Interface includes a built-in SQL workstation engine. If you do not have a remote Scalable SQL 4 server installed, the built-in workstation engine translates the ODBC calls to Btrieve calls, allowing the ODBC application to communicate directly with the remote Btrieve server engine. You should use this built-in local SQL engine only if you do not want to have (or cannot have) Scalable SQL 4 running on the server.

If you wish to run a Win16 ODBC application directly against a remote Btrieve 7 server engine (remote Scalable SQL access disabled), perform the following steps:

1. Using Setup (Win32), choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.
2. Choose **Local Usage** under Settings. Set the value to “Off.”
3. Choose **Remote Usage** under Settings. Set the value to “Off.”
4. Under Component, choose **MicroKernel Router (Win32)** and under Categories, choose **Access Control** .
5. Under Settings, choose **Local** . Set the value to “Off.”
6. Under Settings, choose **Requester** . Set the value to “On.”
7. Click **Save** , and click **Exit** .
8. Using Setup (Win16), choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.
9. Choose **Local Usage** under Settings. Set the value to “On.”
10. Choose **Remote Usage** under Settings. Set the value to “Off.”
11. Choose **Use Thunk** under Settings. Set the value to “Off.”
12. Under Component, choose **MicroKernel Router (Win16)** and under Categories, choose **Access Control** .
13. Under Settings, choose **Local** . Set the value to “Off.”
14. Under Settings, choose **Requester** . Set the value to “Off.”
15. Choose **Use Thunk** under Settings. Set the value to “On.”
16. Click **Save** , and click **Exit** .

If you wish to run a Win16 ODBC application against a remote Scalable SQL 4 server engine (Btrieve access disabled), perform the following steps:

Note: Scalable SQL 4 and Btrieve 7 server engines are both available by default. You should only use this procedure if you wish to disable Btrieve access and use only the SQL server engine.

1. Using Setup (Win32), choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.
2. Choose **Local Usage** under Settings. Set the value to “Off.”
3. Choose **Remote Usage** under Settings. Set the value to “On.”
4. Under Component, choose **MicroKernel Router (Win32)** , and under Categories, choose **Access Control** .
5. Under Settings, choose **Local** . Set the value to “Off.”
6. Under Settings, choose **Requester** . Set the value to “Off.”
7. Click **Save** , and click **Exit** .
8. Using Setup (Win16), choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.
9. Choose **Local Usage** under Settings. Set the value to “Off.”
10. Choose **Remote Usage** under Settings. Set the value to “Off.”
11. Choose **Use Thunk** under Settings. Set the value to “On.”
12. Under Component, choose **MicroKernel Router (Win16)** , and under Categories, choose **Access Control** .
13. Under Settings, choose **Local** . Set the value to “Off.”
14. Under Settings, choose **Requester** . Set the value to “Off.”
15. Choose **Use Thunk** under Settings. Set the value to “Off.”
16. Click **Save** , and click **Exit** .

If you wish to run a Win16 ODBC application with access to both Btrieve and Scalable SQL enabled, perform the following steps:

Note: The SQL and Btrieve server engines are both available by default. You should only use this procedure if you wish to restore or verify the default configuration.

1. Using Setup (Win32), choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.
2. Choose **Local Usage** under Settings. Set the value to “Off.”

3. Choose **Remote Usage** under Settings. Set the value to “On.”
4. Under Component, choose **MicroKernel Router (Win32)** and under Categories, choose **Access Control** .
5. Under Settings, choose **Local** . Set the value to “Off.”
6. Under Settings, choose **Requester** . Set the value to “On.”
7. Click **Save** , and click **Exit** .
8. Using Setup (Win16), choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.
9. Choose **Local Usage** under Settings. Set the value to “Off.”
10. Choose **Remote Usage** under Settings. Set the value to “Off.”
11. Choose **Use Thunk** under Settings. Set the value to “On.”
12. Under Component, choose **MicroKernel Router (Win16)** and under Categories, choose **Access Control** .
13. Under Settings, choose **Local** . Set the value to “Off.”
14. Under Settings, choose **Requester** . Set the value to “Off.”
15. Choose **Use Thunk** under Settings. Set the value to “On.”
16. Click **Save** , and click **Exit** .

Running Applications on the Server (Windows NT only)

In some cases you may wish to run an application on the computer where the server engine is located. This configuration is only permitted on Windows NT. This section explains how to specify that an application running on Windows NT should access the local server engine.

Note: This section only applies to Pervasive.SQL Server.

If you wish to enable access to the local Scalable SQL server engine only:

1. Using Setup (Win32), choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.
2. Choose **Local Usage** under Settings. Set the value to “On.”
3. Choose **Target Engine** under Settings. Set the value to “Server Only.”
4. Click **Save** , and click **Exit** .

If you wish to enable access to the local Btrieve server engine only:

1. Using Setup (Win32), choose **MicroKernel Router** under Component, and choose **Access Control** under Categories.
2. Choose **Local** under Settings. Set the value to “On.”
3. Choose **Target Engine** under Settings. Set the value to “Server Only.”
4. Choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.
5. Choose **Local Usage** under Settings. Set the value to “On.”
6. Choose **Target Engine** under Settings. Set the value to “Workstation Only.”
7. Click **Save** , and click **Exit** .

If you wish to enable access to both the local Btrieve server engine and the local Scalable SQL server engine:

1. Using Setup (Win32), choose **MicroKernel Router** under Component, and choose **Access Control** under Categories.
2. Choose **Local** under Settings. Set the value to “On.”
3. Choose **Target Engine** under Settings. Set the value to “Server Only.”
4. Choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.
5. Choose **Local Usage** under Settings. Set the value to “On.”
6. Choose **Target Engine** under Settings. Set the value to “Server Only.”
7. Click **Save** , and click **Exit** .

If you wish to enable access to the local Btrieve Win32 workstation engine only:

1. Using Setup (Win32), choose **MicroKernel Router** under Component, and choose **Access Control** under Categories.
2. Choose **Local** under Settings. Set the value to “On.”
3. Choose **Target Engine** under Settings. Set the value to “Workstation Only.”
4. Choose **Scalable SQL Requester** under Component, and choose **Access Control** under Categories.
5. Choose **Local Usage** under Settings. Set the value to “On.”
6. Choose **Target Engine** under Settings. Set the value to “Workstation Only.”
7. Click **Save** , and click **Exit** .

Supplying Btrieve Owner Names

The ODBC Interface supports supplying a Btrieve owner name for a data file. This is useful if you are maintaining a database that has Btrieve owner names set on individual files. Owner names are a Btrieve security feature, and you must supply the owner name in order to access or update the file or retrieve file statistics.

To supply the Btrieve owner name, you can call SQLSetConnectOption, as shown in the following example.

- SQLSetConnectOption:
    - In:

hdbc=0x00821AE8,fOption=1000, vParam=ownerNames, Parameter Type=SQL_C_CHAR

    - Return:

SQL_SUCCESS=0

Set the vParam parameter to the address of a null-terminated string that contains the owner names separated by commas. You must specify each unique owner name required by the database. (For example, if all the files use the same owner name, you specify that owner name just once.) If an owner name contains spaces, enclose it in single quotes.

If you make this call more than once for the same connection, each call supersedes the previous call, rather than adding to it.

Supported Versions

Two different versions of Pervasive ODBC Interface are currently available. The versions are ODBC Interface 2.5 for use with Pervasive.SQL 7, and ODBC Interface 2.04 for use with Scalable SQL v4 (or earlier) or Btrieve 6.15 (or earlier).

The APIs and functionality offered by these versions are identical. The major difference is that version 2.5 is designed to work with the new Smart Components architecture in Pervasive.SQL, while 2.04 is not.

This manual is accurate for both versions of ODBC Interface.

Variations From the ODBC SQL Grammar

This section describes areas in which the SQL grammar of the interface varies from the ODBC SQL grammar.

Variable Length Fields as the Last Field in Views and Tables

The standard approach to variable-length fields in Pervasive.SQL data is to require that they appear as the last field in a table or view. When constructing joins using the ODBC Interface, you must structure the SQL statement so the variable-length field is the last field in the field selection list.

For example, suppose two tables exist in the following formats:

```
CREATE TABLE T1 (Key INT(4), Memorandum NOTE(1000))
CREATE TABLE T2 (Key INT(4),Earned^value INT(4))
```

A SELECT statement such as the following does not work:

```
SELECT * FROM T1, T2 WHERE T1.Key = T2.Key
```

This is because the NOTE field in T1 would appear in the middle of the view. The statement must be restructured so that the NOTE field is at the end of the view, for example:

```
SELECT * FROM T2, T1 WHERE T1.Key = T2.Key
```

You can achieve the same effect by explicitly naming the fields instead of using the asterisk to indicate all fields:

```
SELECT T1.Key, Earned^value, Memorandum FROM T1, T2 WHERE T1.Key = T2.Key
```

Qualify Index Names with Table Names

The ODBC SQL grammar supports using table names to qualify index names. With Pervasive.SQL, index names must be unique within the DDF.

Note: The index name is not necessarily related to the name of the column or columns used to create the index. Rather, the index name is assigned by the user as part of the CREATE INDEX statement.

For example, the ODBC Interface does not recognize the following SQL statement:

```
DROP INDEX Patients.FirstLast
```

Instead, use the following statement:

```
DROP INDEX FirstLast
```

Note: Indices created using Btrieve v6.15 do not have names and cannot be dropped using the DROP INDEX statement.

Scalar Functions

The following scalar functions are supported:

Table 4-1

Supported Scalar Functions

String		Numeric	Time and Date
	RIGHTRTRIM	ROUNDTRUNCATE	
CONCATLC	SUBSTRING		
ASELEFTLE	UCASE		
NGTHLTRIM			

CURDATECURTIMEDAYOFMONTHDAYOFWEEKHOUR MINUTEMONTHSECONDYEARNOW (Scalable SQL v4.0 and Pervasive.SQL only)

Although LENGTH and SUBSTRING are listed as applying to strings, they also work for NUMERIC and DECIMAL fields.

Scalar functions can be entered as-is into SQL statements, but the resulting SQL statement is not ODBC-compliant and therefore not portable. To create portable SQL, use an ODBC “vendor string” escape sequence, as shown in the *ODBC 2.0 Programmer’s Reference* . For example:

```
SELECT {fn LCASE(keycol1)} FROM table1
```

If a field name in a vendor string contains a space, the default blank replacement character (caret) does not work. In this case, set blank to underscore and use underscore in place of the space in the vendor string. After executing the scalar function, you can reset blanks to carets. See the following example:

```
SET BLANK = ‘_’{fn LCASE(first_name)} FROM PATIENTSBLANK = ‘^’
```

What is ODBC?

Open Database Connectivity is a standard introduced by Microsoft Corporation, based on the SQL Access Group's Call Level Interface (SAG CLI) specification. The ODBC standard is a superset of the SAG CLI and the X/Open and ANSI SQL-92 grammar.

The specification of the ODBC standard is set forth in the *ODBC 2.0 Programmer's Reference and SDK Guide* , published by Microsoft Press. Developers should refer to this specification to familiarize themselves with the ODBC call-level interface and the ODBC SQL grammar.

Who Should Read This Manual

This manual provides information for programmers and users of Pervasive's ODBC Interface to Btrieve and Scalable SQL.

Pervasive Software would appreciate your comments and suggestions about this manual. As a user of our documentation, you are in a unique position to provide ideas that can have a direct impact on future releases of this and other manuals. Please complete the User Comments form on the Pervasive web site or send email to docs@pervasive.com.
