

ProvideX

ODBC

Local and Client/Server

Introduction	1
Installation Procedures	4
Local & Client Configuration	8
Server Configuration	16
Table Definitions	23
Using the ODBC Driver	37



ProvideX is a trademark of Best Software Canada Ltd.

All other products referred to in this document are trademarks or registered trademarks of their respective trademark holders.

©2004 Best Software Canada Ltd. — Printed in Canada

8920 Woodbine Ave. Suite 400, Markham, Ontario, Canada L3R 9W9

All rights reserved. Reproduction in whole or in part without permission is prohibited.

The capabilities, system requirements and/or compatibility with third-party products described herein are subject to change without notice. Refer to the Best Software Canada Ltd. website www.pvx.com for current information.



ProvideX ODBC

The ProvideX ODBC driver delivers third party access to ProvideX data. It enables any ODBC-compliant application on any Windows platform to communicate with your ProvideX database from any location on the network. Currently, two ProvideX ODBC products are available for download:

- **Local** driver version (read only and read/write capabilities).
- **Client/Server** version (TCP-based, part of the Professional or eCommerce bundle).

These products are available separately from the base ProvideX installation and require separate licenses, installation files, and activation procedures. Contact your dealer/distributor or visit the ProvideX website at www.pvx.com for the latest information on ODBC product options and licensing.

This document discusses the basic concepts and features of ProvideX ODBC. It covers installation and configuration procedures for both local driver and client-server versions, defining/accessing data files, and use of the ODBC driver to access ProvideX data from other languages and applications.

What is ODBC?

ODBC is the acronym for *Open DataBase Connectivity*, an interface standard that maintains a common access method for DBMS (*DataBase Management Systems*). The ODBC interface provides a standard set of functions or APIs (*Application Program Interfaces*) that allow applications to access a variety of ODBC-compliant databases. It also administers the database names and drivers associated with the data files.

ODBC access is based on SQL (*Structured Query Language*) which is an English-like database access language designed to enable end-users to view and manipulate data files. Over the years, the SQL language has been standardized by ANSI and adopted by a large number of database manufacturers. SQL's original intent was to provide ad-hoc access to data — but not as a development language or as a database interface tool. With the advent of ODBC and other generic interfaces, SQL became the de-facto standard used to manipulate databases.



Because the SQL language is English-like in its structure, it is easy to learn and understand. The basic SQL directives are:

- SELECT** to read and return data
- UPDATE** to alter existing data records
- INSERT** to add records
- DELETE** to remove data records

Example:

```
SELECT cst_id, cst_name FROM Customer
```

This retrieves customer numbers and names from the Customer file. For more information on the use of SQL with ProvideX ODBC, see [Using the ODBC Driver, p.37](#).

ODBC Architecture

Typically, the standard ODBC architecture consists of four major components:

- Application** Responsible for interacting with the user and for calling ODBC functions to submit SQL statements to and retrieve results from one or more data sources.
- Driver** Processes the ODBC function calls, submits SQL requests to a specific data source, and returns results to applications. Also, the driver is responsible for interacting with the software needed to access a specific data source.
- Driver Manager** Loads/calls drivers on behalf of an application. The driver manager processes ODBC function calls or passes them to the driver.
- Data Source** Represents the data to be accessed. It can be a flat-file, or a particular database in a DBMS. It also refers to the actual location of the data as well as any technical information needed to access the data (driver name, network address, user ID, password, etc.)

This architecture enables an application to access different ODBC data sources, in different locations, using the same function calls available in the ODBC API. Components interact in the following chain of events:

- ODBC-compliant application uses an API to submit SQL directives to the data source.
- Communication between the application and ODBC driver is handled by the driver manager, which loads the driver and passes along the API requests.
- The ODBC driver implements the functions of the ODBC API for the selected DBMS data source.
- Requests are processed by the data source, and the results are sent back up the chain to be retrieved by the application.



Why Use ODBC/SQL?

ODBC allows your ProvideX data to be opened up to the most popular database managers, query applications, and report writers: MS SQL Server 7, Excel or Word with MSQUERY, Informix, and Crystal Reports, just to name a few. Most programming languages have an ODBC access facility to allow files to be read or updated as well.

ODBC/SQL allows standardized access to ProvideX data via:

- Standardized Data Formats: Text strings, numerics, dates.
- Logical Relationships: Relates files with common data elements.
- Data Sorting, Grouping and Filtering.
- Simple Data Computations: Sum, Max, Min, Count, Avg.

The ProvideX ODBC driver supports three basic types of data: strings, numerics, and dates.

The `SELECT` statement is used to establish logical relationships between data files (usually referred to as joining files). A typical `JOIN` would be:

```
SELECT cst_id, cst_name, smn_name FROM Customer, Salesman  
WHERE smn_id = cst_smn
```

The statement reads the entire `Customer` file and for each customer, reads the `Salesman` file for any records whose `smn_id` matches `cst_smn`. If the field `smn_id` is a Key field for the file, then the ProvideX ODBC driver reads the file directly by key, otherwise the file is read in its entirety. The `WHERE` clause can be used to selectively filter out any unwanted data.

The ODBC driver can sort the data on any field using the `ORDER BY` clause of the `SELECT` statement. If the `ORDER BY` fields match any of the key fields of the primary file, then the primary file is accessed by this key. In addition, you can `GROUP` data `BY` common fields.

`SUM`, `COUNT`, `AVG`, `MAX`, `MIN` functions can be used to provide statistical information on the data fields.

You can find a list of SQL keywords supported by the driver in *Appendix A*.

Installation Procedures

Installation files for the different versions of ProvideX ODBC Local and Client/Server can be obtained from your dealer/distributor or downloaded from the ProvideX website, www.pvx.com.

In order to set up and run a *permanent* version of ProvideX ODBC, you must obtain a license and be issued a unique serial number, user count, and activation key for the product you purchased. No activation information is required to record a demo mode activation. The various activation categories and procedures are further described in the *ProvideX Installation and Configuration Guide*.

ProvideX Client/Server ODBC (available as part of the Professional or eCommerce bundle) requires components to be installed on both sides of the client/server connection. The server component is available for installation on both Windows and UNIX/Linux platforms. The client component is available for Windows platforms only. To ensure compatibility, the client and server-side components must have the same version number.



Note: Downloads for the Local driver and the Client component of the Client/Server driver are available with or without MDAC (Microsoft Data Access Components). If you choose not to install MDAC, the installation program automatically verifies if your current version of MDAC (if any) is compatible with ProvideX ODBC.

The following sections describe procedures for the installation of ProvideX ODBC components on different platforms: [ODBC Installation and Activation \(Windows\)](#) and [ODBC Server Installation \(UNIX/Linux\)](#). Information on configuring data sources via the Windows ODBC Data Source Administrator can be found under the heading [Local & Client Configuration, p.8](#). ProvideX ODBC Server settings for Windows and UNIX/Linux are explained under the heading [Server Configuration, p.16](#).

ODBC Installation and Activation (Windows)

Installation programs for the ODBC Local driver, Client driver, or Server can be obtained from your dealer/distributor or from the ProvideX website. The installation process is virtually identical for all ProvideX ODBC Windows driver/server components.

After you download the appropriate installation program (e.g., `podb1330.exe`), follow these steps to install the Local driver, Client driver, or Server on your computer:

1. If possible, remain connected to the Internet. The installation process may include some options to download additional components.
2. Double-click on the installation program that was downloaded to your computer to begin the installation process. The installation program launches an *InstallShield Wizard* similar to the ProvideX installation and immediately checks for existing ProvideX ODBC components (driver or server, depending on the installation):

- If identical ODBC components already exist on your machine, you will be given the option to modify, repair, or remove the existing driver/server components.
- If a different version (older or newer) exists on your machine, you will be warned that the existing driver/server components are to be overwritten.
- If you are installing the Server and a ProvideX ODBC service is already running on your computer, you will be warned that the existing service must be closed to resume the installation.

When the installation wizard has checked the above criteria and is cleared to proceed, it takes you through a series of dialogue boxes:



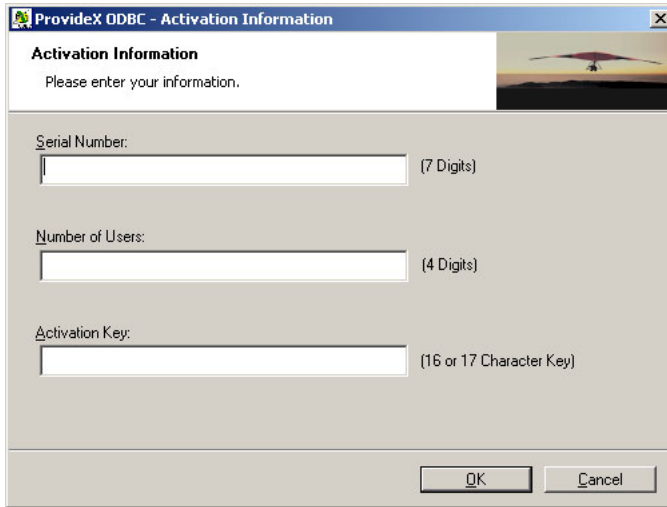
3. Follow the wizard instructions and click **Next >** to complete each step. The final step installs driver/server components onto your hard disk and displays a progress bar to indicate the current installation status. *This process may take several minutes.*
4. When all components are copied to disk, the ODBC *activation dialogue* is invoked for a new installation, otherwise the wizard will indicate that the driver or server has been updated successfully.

A valid serial number, number of users, and activation key are required in order to set up and run permanent versions of the ODBC. The necessary activation information will be issued to you once you purchase a product package from Best Software Canada Ltd or your authorized dealer/distributor.



Note: When installing the ODBC as part of an eCommerce or Professional licence, use your *temporary key* for permanent ODBC activation. Permanent keys that are generated for bundled activations do not apply to ODBC components.

The ProvideX ODBC activation dialogue appears as follows:



If you press OK and the activation is invalid, you will be given the option to enter your information again. If you press Cancel, the activation utility automatically records a demo mode activation for the ODBC; in this case, the activation dialogue pops up for every ODBC connection and a "nag" message appears on every execute.

Refer to [Local & Client Configuration](#) for configuration details.

ODBC Server Installation (UNIX/Linux)

Obtain the ODBC Server distribution file from your dealer/distributor or via the ProvideX website. Ensure that you download the correct version for your specific UNIX/Linux operating system. The ODBC distribution file is named with a .taz extension, which is short for .tar.Z, a compressed version of a UNIX .tar file:

poxxxvvv.taz

Where:

- xxx identifies a specific operating system; e.g., RH6 for Redhat versions 6 to 7.1.
- vvv identifies the version of the ODBC server; e.g., 312 for version 3.12.

The poxxxvvv.taz distribution file contains the following installation components:

- pvxodbs ODBC Server executable.
- pvxodbs.conf.sample ODBC Server configuration file (sample).
- install.txt Installation readme file.
- license.txt License agreement.
- podbcsvv.txt ODBC version readme file containing current change information.

After you download the `poxxxvvv.tar` file to a `/tmp` directory, follow these steps to expand, install, and activate the ProvideX ODBC Server program on your computer:

1. Change directories to the `/tmp` directory and rename the `poxxxvvv.tar` with a `.tar.Z` extension so that it can be uncompressed:

```
umask 0
cd /tmp
mv poxxxvvv.tar poxxxvvv.tar.Z
uncompress poxxxvvv.tar.Z
```

2. Create the new directory to receive the ProvideX software and move into it. Use `/usr/pvxodbc` for the directory name:

```
mkdir /usr/pvxodbc
cd /usr/pvxodbc
```

3. Use the `tar` command to copy the software into the `/usr/pvxodbc` directory:

```
tar xvf /tmp/poxxxvvv.tar
```

4. Set the file permissions on the `pvxodbs` executable and configuration files to whatever is necessary depending on the `username` who will be running the server daemon (typically root):

```
chmod 500 pvxodbs
chmod 600 pvxodbs.conf.sample
chown root pvx*
chgrp root pvx*
```

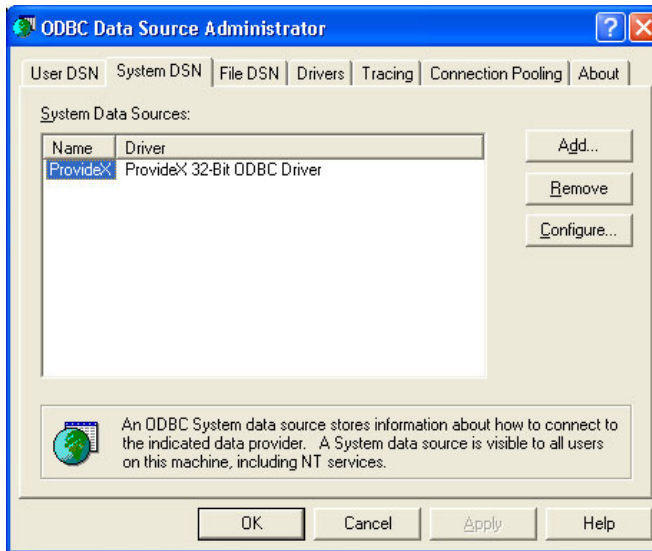
5. If this is the first time the ODBC Server has been installed on this system, then rename the `pvxodbs.conf.sample` file to `pvxodbs.conf`.

```
mv pvxodbs.conf.sample pvxodbs.conf
```

At this point, the installation of the ODBC Server is complete; however, the `pvxodbs.conf` file may require updated settings. For configuration/activation details and the list of command line arguments, refer to the [ProvideX ODBC Server Settings for UNIX/Linux, p.19](#).

Local & Client Configuration

The ProvideX ODBC local driver and the Client component of the Client/Server driver are configured using the *ODBC Data Source Administrator*, which can be accessed via the Windows *Control Panel* (in the *Administrative Tools* subfolder on Windows 2000/XP):



This is where you define each database and set up associated configuration details; i.e.,

- Data Source.
- Directory containing a ProvideX Data Dictionary file `providex.ddf`.
- INI file used when manually defined.
- Company and User codes.
- Options.

Either the data file directory or the INI file, (or both), must be defined. There must be at least one source for a Data Dictionary. If both have been specified, then the contents of both will be merged. Additional ProvideX ODBC Server settings are required for the client-server version of the driver:

- Server Name or IP (e.g., `localhost` or `127.0.0.1`)
- TCP/IP Port (default: `20222`).

Data Source Names (DSN)

A data source defines the location of data, and the connection information needed to access that data. In effect, it defines the path to the data, which may include a network, library, server, database, and other attributes.

In order to establish a connection to a data source, you must do the following:

1. Ensure that the appropriate ODBC client driver is installed on the computer that contains the data source. This is described under **Installation Procedures, p.4**.
2. Use the *ODBC Data Source Administrator* to set up a *data source name* (DSN) to store the necessary connection information in the Windows registry or in a DSN file.

If the ODBC connection information is stored in the *Windows registry*, it is called a *machine* data source. A machine data source can be either a *user* data source (one user has access) or a *system* data source (visible to all users on, or connected to, the same computer). The main advantage to having a machine data source is that it provides security within the system to limit who is logged on to view the data source and prevent the data source from being copied to other computers. Machine data sources can only be used on the computer where they were defined.

If the ODBC connection information is stored in a *DSN file*, it is called a *file* data source. A file data source is defined in a flat text file and, unlike machine data sources, they can be ported to any system. The main advantage to having a file data source is that it can be placed in common directories and shared between users; e.g., a file DSN can be distributed among clients as a part of an installation package.

The ODBC Data Source Administrator interface allows you to choose between different DSN tabs, depending on the type of data source to be modified:

User DSN Defines machine data sources for the user currently signed on.

System DSN Defines machine data sources for a particular workstation.

File DSN Places and maintains data source definitions in a portable text file.

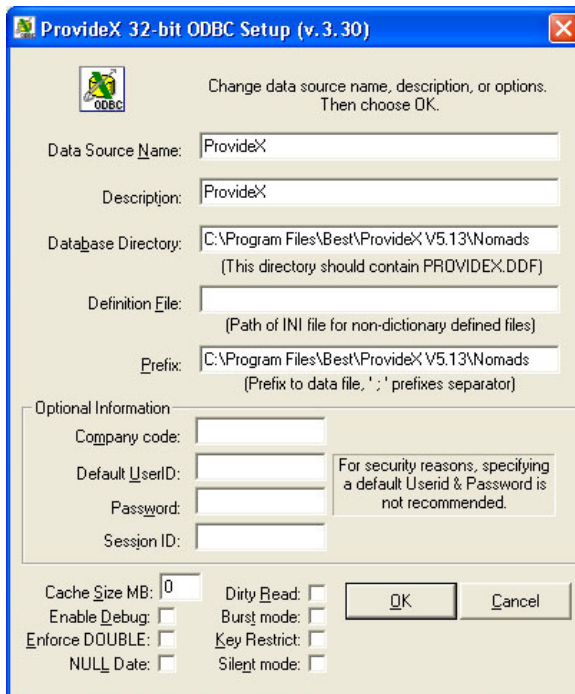
Click on one of the tabs to list the current connections for that DSN type. From here you can change/remove an existing DSN or add/configure a new DSN.

Creating a New DSN

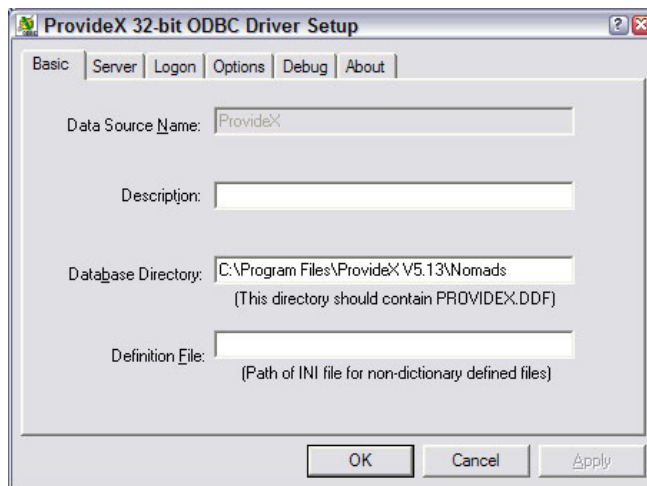
To create a new DSN for ProvideX ODBC, click the **Add** button. The next dialogue displays a list of the ODBC drivers that are installed on your system. Select the appropriate ProvideX ODBC Driver from the list and click **Finish**. This invokes the *ProvideX 32-bit ODBC Setup* dialogue, which allows you to create and configure access to a ProvideX database.

Dialogues for the Local driver and the Client component of the Client/Server driver are identical, except for the Server Name and TCP/IP fields. While the dialogues (shown below) are different in appearance between versions of the ProvideX ODBC, they cover most of the same information. The most notable difference is that the configuration fields are grouped under *tabbed headings* in newer versions of the dialogue. The new dialogue also includes an **About** folder that displays the ProvideX ODBC driver version, and the location of the DLL being used.

For *ProvideX ODBC Version 3.30*, the setup dialogue appears as follows:



For *ProvideX ODBC Version 3.31* and later, the setup dialogue appears as follows:



Basic Configuration Entries

The following fields appear in beginning of the driver setup dialogue regardless of the ProvideX ODBC version:

Data Source Name: Name (DSN) that other applications will use to access the database. Case-insensitive, maximum length is 32 characters.

With regards to the ProvideX ODBC driver, the DSN can be considered the logical name of the database. The following characters are not permitted in a DSN:

[] { } () , ; ? * = ! @ \

Description: Optional freeform remark describing the Data Source Name. Maximum length is 127 characters.

Database Directory: Location of the ProvideX Data Dictionary file (`providex.ddf`) which is the relative starting point for all embedded file references. Maximum length is 127 characters. If used with MAS90, then the directory must contain the `DDICT` directory.

If `providex.ddf` is found in this directory, then all file/table definitions contained in it are made available to the ODBC driver. Using the embedded data dictionary simplifies the installation and maintenance issues regarding the ODBC.

The `providex.ddf` file located in the database directory can be set up to contain only a subset of the files used by an application. This can be used to control which files/tables are presented to the end-user. In order to provide different "views" of the database, create separate directories, each containing a different `providex.ddf` file.

Note that the `providex.dde` file is not required by the ODBC driver.

For more information, see [ProvideX Data Dictionary, p.23](#).

Definition File: Path and name of the INI file used to define the data dictionary manually for files that cannot be handled by the ProvideX embedded data dictionary. Maximum length is 127 characters. This INI file-based data dictionary follows the ODBC driver to access these files.

For more information, see [INI Definition, p.26](#).

Server Entries

The following entries set up the client component of the Client/Server version of the ProvideX ODBC driver:

Server Name or IP: Server network name or IP address required for connecting to the ProvideX ODBC Server. Maximum length is 100 characters. For example,

ProvideXFileServer

or

127.34.28.15

TCP/IP Port: TCP/IP Port required for connecting to the ProvideX ODBC Server. Default is 20222. Maximum length is 15 characters.

You can change the TCP/IP port that the server is listening on via the Control Panel Configuration program, in which case the DSN TCP/IP Port setting on the client side must be changed as well.

Logon or Optional Information

Default values can be set in the **Company code**, **User ID** and **Session ID** fields for use in the definition of data file pathnames. Whenever a data file pathname starts with an equal sign =, the pathname will be scanned. All occurrences of %C\$ will be replaced with the value set in the default company code, %U\$ will be replaced with the default user ID and %S\$ will be replaced with the default Session ID. The search for occurrences is case-insensitive, thus %c\$ and %C\$ will both be found and replaced with the value of the company code field.

When using MAS90 data files the ODBC driver will prompt the user to enter a valid company and user ID every time a connection to the database. For other databases, enter a question mark ? in any of the optional fields during the DSN setup and the driver will prompt for the values during a database connection. There is no validation of the values entered.

The following fields are found in the **Optional Information** section (Version 3.30) or under the **Logon** tab (Version 3.31 or later) of the setup dialogue.

Company code: Optional value to replace occurrences of %C\$ in a definition pathname. Maximum length is 127 characters.

Default User ID: Optional value to replace occurrences of %U\$ in a definition pathname. Maximum length is 64 characters.

Password: Optional password value — used in conjunction with a MAS90 system only. Maximum length is 63 characters.

Session ID: Optional value to replace occurrences of %S\$ in a definition pathname. Maximum length is 15 characters.

Options

The setup dialogue provides for other optional entries. Some of these fields appear together under the Options folder in ODBC Version 3.31 or later:

- P**refix: Search paths to be inserted in front of all relative file references used in Data Dictionary or INI definitions. Use a ';' separator between multiple prefixes. The maximum length is 1023 characters.
- V**iews DLL: *Version 3.31 & later.* Path to `pvxwin32.dll`. This is required by the ODBC in order to use the Views system (ProvideX Version 5.10 or later, or the ProvideX ODBC Server for Windows).
- C**ache **S**ize MB: *Version 3.30 & earlier.* Size of ODBC cache buffer. The execution of some SQL join commands may require multiple passes through the same data file. The ODBC cache buffer option eliminates the need to go to disk for each pass of the data file. If defined, the cache is used whenever a secondary disk file is required whose size is equal to or less than the cache size.
- E**nforce **D**ouble: Checkbox to set default format of "double" for numeric data. This helps avoid conflicts with MS Office 2000 and other applications that do not support the decimal data type for numeric values.
- N**ULL **D**ate: Checkbox to suppress invalid date error. The driver validates the contents of date columns at run time. If a value is invalid, the driver generates an error message and ceases processing of the table. This replaces an invalid entry with a null value and allows the driver to continue processing.
- K**ey **R**estrict: Checkbox to restrict keys. This option allows the driver to be used with applications which do not support keys or support them with limitations on length, field segments, use of sub-strings, etc.

For example, Lotus Approach 97 does not manage a stack memory properly if you are using files with a key consisting of more than one data field.
- S**ilent **M**ode: Checkbox to suppress most prompts or message boxes that the ODBC driver generates during processing.

Performance Tuning

Normally, when the ODBC driver accesses data files, it must place a temporary lock on the file. This temporary lock guarantees that the driver reads key tables and structures that are in a consistent state and not in the process of being altered by other applications.

Once the temporary lock is established, the driver checks the file header to see if it has been changed since the last time the file was accessed. If the file has not been altered, then the ODBC driver can use any of the data still maintained in its buffers. If the file has been altered, then all data in the buffers is discarded. When the driver has completed its access to the data file, the temporary lock is released.

The process is repeated for each file accessed by the driver, for each operation on the file. The following fields provide methods to reduce the overhead when processing a file:

- Dirty Read:** Checkbox for Dirty Read mode of operation to skip the normal file consistency checks. Dirty read never locks the file header or checks for file changes. This speeds file processing but can result in inconsistent data if the file is being updated while being read by the ODBC driver.
- Burst mode:** Checkbox to enable Burst mode. Placing the temporary lock and reading the file header places significant overhead on the ODBC driver. With Burst mode set, the ODBC driver locks the file header for either 20 file operations or three-tenths of a second, whichever occurs first. This reduces the number of times the file must be locked, and the number of times that internal buffers may need to be purged

Debug

The ProvideX ODBC debug option traces active sessions within the ProvideX ODBC driver and generates a log file. This reports internal debug information that is different from the SQL tracing provided by the Microsoft ODBC Driver Manager. The following fields set the debug option and log file:

- Enable Debug:** Checkbox to enable the ProvideX ODBC debug option. Prior to Version 3.31, the driver automatically generates output to C:\pvxodb32.log (regular driver) or C:\pvxodc32.log (client-server version). For Version 3.31 and later, see below.
- Log File:** *Version 3.31 & later.* Path and name of debug log file.
If this field is left blank then the driver automatically generates output to C:\pvxodb32.log (regular driver) or C:\pvxodc32.log (client-server version).

Connection String	<i>Version 3.31 & later.</i> Button to invoke a display of the connection string returned by the driver. If using SQLDriverConnect then the information displayed in the area above the button is the connection string representing the currently saved DSN attributes. See <i>Connection String Keywords</i> below.
Test Connection	<i>Version 3.31 & later.</i> Button to test the connection to the configured database. If successful then the area above the button will display the text: Connection succeeded. Datasource includes <i>x</i> tables. Where <i>x</i> is the number of tables reported for the database.

Connection String Keywords

The ODBC driver recognizes keywords as part of a connection string. Format is *keyword=value* (case-insensitive) with multiple entries separated by semi-colons; e.g., DSN=MyDSN;UID=John;PWD=foo;Company=ABC. The keywords are listed below:

DSN	Name of the DSN to use for default values.
FILEDSN	Name of the file DSN to use for default values.
DRIVER	Name of the driver to use (DSN-less connection).
Description	Description of the DSN (optional).
Directory	Directory containing the <code>provider.ddf</code> file.
IniFile	Directory and file name of the INI file to be used.
RemoteHost	Server name or IP address of the ODBC server.
RemotePort	Port the ODBC server is monitoring.
Company	Company code.
UID	User ID.
PWD	Password.
SID	Session ID.
Prefix	Data search prefix.
ViewDLL	Location of the Views DLL.
BurstMode	0 burst mode off, 1 burst mode on.
DirtyReads	0 dirty read off, 1 dirty read on.
Debug	0 debug output off, 1 debug output on.
Silent	0 silent off, 1 silent on.
KeyRestrict	0 report key columns, 1 disable reporting of key columns.
EnforceDouble	0 do not force numerics to double, 1 report all numerics as double.
EnforceNullDate	0 null date off, 1 null date on.
LogFile	Path and name of the file to write debug output to.

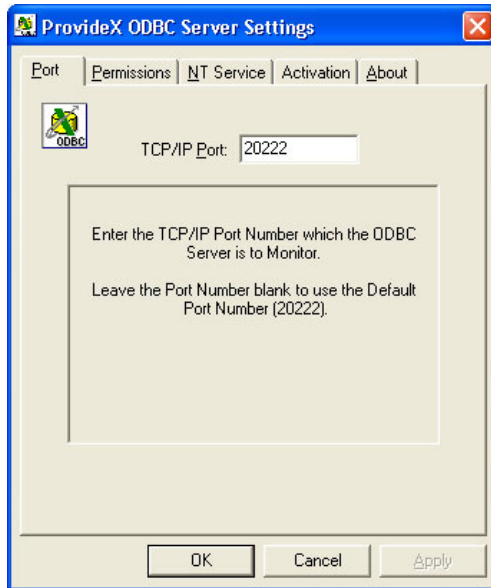
Server Configuration

The configuration settings for the server-side of the ProvideX Client/Server ODBC allow you to specify a TCP/IP port number, set up and manage the data files access permission, and establish the server activation.

When the ProvideX ODBC Client/Server is installed on a Windows system, the server component is configured using the *ProvideX ODBC Server Settings* interface. Under UNIX/Linux, the server is configured using command line arguments and a configuration text file. The following sections describe the ProvideX ODBC **Server Settings for Windows** and **Server Settings for UNIX/Linux**.

Server Settings for Windows

The server is configured in Windows via the *ProvideX ODBC Server Settings* interface, which can be accessed directly from the *Control Panel* at any time.



The default TCP/IP port number is 20222.

Activation

As with most ProvideX products, valid activation information (serial number, number of users, and activation key) must be recorded in order to activate the ODBC server. If the activation is accepted, the configuration program returns a confirmation message when you press **Apply**.

It is possible to change the activation at any time. For example, an increase in the number of users on the system could require a new license and new activation values. The ProvideX ODBC Server controls the number of concurrent client connections and denies access if the number of users is exceeded.

Because activations are only verified during the initialization process, the server must be restarted when a new activation is recorded.

NT Service

The ProvideX ODBC Server installation program checks the operating system and, if it detects a NT/2000/XP system, it automatically sets up the ProvideX ODBC as a service. On installation, the server's *Startup Type* defaults to *Automatic*, which means that the ProvideX ODBC service will start automatically every time the system reboots and will run independently of any logged-on user. This setting is evident by the message displayed in the NT Service folder:

ProvideX ODBC Service is running.

As with other services, the ODBC service can be controlled (stopped, paused, etc.) using the Windows *Services* interface, which is accessed via the *Control Panel* (in the *Administrative Tools* subfolder on Windows 2000/XP). The ProvideX ODBC service can also be uninstalled (and reinstalled) via the NT Service panel.



Note: The NT Service tab will be hidden if the server is installed on Windows 9x systems — they do not support applications running as services. While it is still possible to run the ProvideX ODBC server as an application on a Windows 9x desktop, it is not recommended. Windows 9x is designed to be a single user system only.

Permissions

On installation, the server is set to default access permissions. These permissions can be viewed/changed in the Permissions folder of the ProvideX ODBC Server Settings interface:

Setting	Default	Definition
Access	A	Access allowed
R/W	R	Read only
User ID	*	Any users
Company Code	*	Any company
Data Dictionary Path	*	Any data dictionaries
INI Path	*	Any INI files

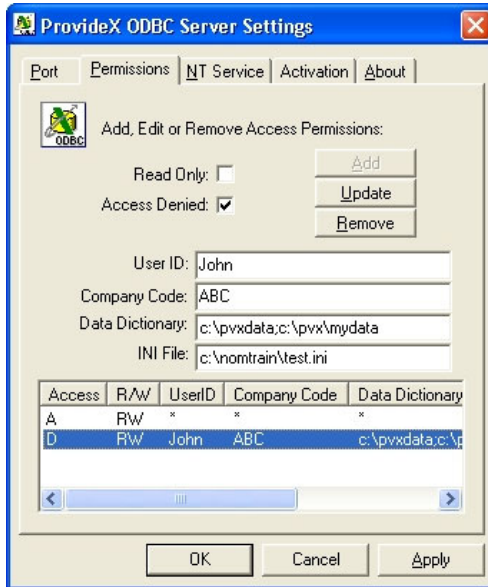
The above default settings grant users almost unrestricted read access to the server's data sources. (An asterisk * indicates *any*.) Therefore, for security reasons, you should reset the parameters based on your own business rules *immediately following the installation*.

The ProvideX ODBC Server checks access permissions by searching the permission rules from the maximum restriction to the lowest one. It is a method to grant access to specific directories on the server based on a client's User ID and Company Code.

If the check for a specific User ID and Company Code fails, then the User ID is substituted with * (*any*) and the combination for User ID = *any* with Company Code = *specific* is checked against the corresponding rule if it is present on the system. The next check is performed for User ID = *specific*, Company Code = *any*, and the last check is for User ID = *any*, Company Code = *any*. Refer to the following table:

Sequence	UserID	Company Code	
1.	Specific	Specific	Highest Restriction
2.	Any	Specific	
3.	Specific	Any	
4.	Any	Any	Lowest Restriction

In the following example, user *John* from *ABC* company is granted access to the data files defined in the *providex.ddf* files located in *c:\pvxdata* and/or *c:\pvx\mydata* directories and in the *test.ini* file located in *c:\nomtrain*:



According to this example, the server administrator has temporarily denied John access; however, John would still be able to access directories not in the list as *all* users of *all* companies may access any directory.

Server Settings for UNIX/Linux

There is no configuration interface for the ProvideX ODBC Server installed on a UNIX/Linux system. Instead, the server behaviour is controlled via command line arguments and a plain text configuration file (`pvxodbs.conf`). For a description of ProvideX UNIX/Linux ODBC Server components and file locations, see [ODBC Server Installation \(UNIX/Linux\), p.6](#).

Running the Server

To run the server, enter the following at the command line:

```
/usr/pvxodbc/pvxodbs [-f ConfigFile] [-p TCPPort]
```

Where:

- f *ConfigFile* Path and file name of the ODBC Server configuration file. If this value is not given the server automatically looks for `/usr/pvxodbc/pvxodbs.conf`. The configuration file is described later in this section.

If no configuration file is located, the server will not start and an error message will be displayed or printed to the log file (if debug is enabled).
- p *TCPPort* TCP/IP port number the server is to listen on. This overrides the port number specified in the `pvxodbs.conf` file.

The following arguments can also be used with the ProvideX ODBC Server executable (`pvxodbs`) at the command line:

- h *or* --help Displays a message listing the available command line arguments with brief descriptions.
- v *or* --version Displays the server version information; e.g.,
ProvideX ODBC Server Ver: 3.30.3000 For: AIX 4.2
Copyright (c) 2003 Best Software Canada Ltd.
- d Enables output to the debug log file `pvxodbs.log`

Shutting Down the Server

The server can be shutdown via SIGHUP signal; i.e.,

```
kill [-pid]
```

Where *pid* is the process ID of the ProvideX ODBC Server, `pvxodbs`.

Permissions

The server configuration file allows for customizable security for users and files. Security entries are case-insensitive except where noted. All the non-alpha characters, “/[]”, are part of the security syntax. The format of a security policy appears as follows:

user id/company code=[mode] [type] [data dictionary] [ini file]

Where:

- user id* Specific user ID supplied by the client driver. An asterisk * signifies *all* user IDs. Spaces are significant. “John /” and “John/” are considered two different entries.
- company code* Specific company code supplied by the client driver. An asterisk * signifies *all* company codes. Spaces are significant. “ / ABC” and “/ABC” are considered two different entries.
- mode* Either A for access or D for denied. If in denied mode, the administrator can temporarily deny access without removing the policy entry.
- type* Either R for read-only or RW for read-write
- data dictionary* A comma or semi-colon separated list of paths to the `provider.ddf` files that the client’s DSN will have access to. An asterisk * signifies that *any* Data Dictionary path is valid. This entry is case-sensitive.
- ini file* Comma or semi-colon separated list of paths and file names of INI files that the clients DSN will have access to. An asterisk * signifies *any* INI file path and file name. This entry is case-sensitive.

The ProvideX ODBC Server checks access permissions by searching the permission rules from the maximum restriction to the lowest one. It is a method to grant access to specific directories on the server based on a client’s User ID and Company Code.

If the check for a specific User ID and Company Code fails, then the User ID is substituted with * (*any*) and the combination for User ID = *any* with Company Code = *specific* is checked against the corresponding rule if it is present on the system. The next check is performed for User ID = *specific*, Company Code = *any*, and the last check is for User ID = *any*, Company Code = *any*. Refer to the following table:

Sequence	UserID	Company Code	
1.	<i>specific</i>	<i>Specific</i>	Highest Restriction
2.	<i>Any</i>	<i>Specific</i>	
3.	<i>Specific</i>	<i>Any</i>	
4.	<i>Any</i>	<i>Any</i>	Lowest Restriction

By default, access to all ODBC resources is denied, unless access is granted via a security policy configuration line.



Note: Access policies are currently kept only by User ID/Company Code, which means that each User ID/Company Code may only have one policy entry. It is not currently possible to specify that a specific User ID/Company Code has read access to one set of entries, and read-write access to a different set of entries.

Sample Configuration Entries

```
serial=12345-6-123456789ABCDEF0
port=20000
security:
*/*=[a][r][*][*]
John/ABC=[D][RW][ /pvxdata;/pvx/mydata][ /nomtrain/test.ini ]
```

Automatic Start

In order to have the ODBC UNIX /Linux server start automatically it must be set up in the `inittab` file. Each `inittab` entry is position dependent and has the following format:

id: *rstate*: *action*: *process*

Where:

- id* Unique identifier for the entry.
- rstate* Run-level for which this entry is to be processed. More than one run-level can be specified
- action* Actions to affect the process specified.
- process* Command to be executed by the system.

The following is an example of an `inittab` entry for the ProvideX ODBC server:

```
podb:2:once:/usr/pvxodbc/pvxods -f /usr/pvxodbc/myOdbc.conf </dev/null >/dev/null 2>&1
```

The above example would start the ProvideX ODBC server the first time the server booted to run level 2. The configuration file named `myOdbc.conf` located in `/usr/pvxodbc/` would be used to configure the server. Any messages sent to standard out or standard error by the server would be suppressed. If the server stopped for any reason the system will not restart it.

Table Definitions

In order to access ProvideX data files in ODBC, the contents of the files (or *tables*) must be described for use with the ProvideX ODBC Driver. Tables can be defined in two ways:

- ProvideX Data Dictionary. This is the preferred method, as files are easily defined using NOMADS Data Dictionary Maintenance and are fully compatible with the ProvideX ODBC driver.
- Formatted text file (INI file). Files are created manually. This method is required when files contains multiple record types.



Note: As of Version 5, the ProvideX Data Dictionary Maintenance allows the entry of multiple record definitions. However, the ODBC driver does not use this information. The ODBC driver will read only the first record format.

Data Dictionary and INI file locations are defined for the ProvideX ODBC driver using the *ODBC Data Source Administrator*. For more information, see [Local & Client Configuration, p.8](#).

This section describes both methods for defining data sources for use with the ProvideX ODBC driver: [ProvideX Data Dictionary](#) and [INI Definition](#).

Terminology

When working with ODBC, the standard term for a set of data is a *table*. In ProvideX a table is often a *physical file*. In this manual the terms table and file are both used. When the manual refers to the data as stored on disk the term physical file is used. Also note that due to the ability to store multiple record formats in a single physical file there may be multiple logical tables for a single physical file.

ProvideX Data Dictionary

ProvideX Data Dictionary file definitions created in NOMADS are compatible with the ProvideX ODBC driver.

Data Dictionary Maintenance is a menu-driven utility in NOMADS that allows you define files by entering pertinent information for each of element (variable name, type, length, delimiter, etc...). From this information, ProvideX builds and maintains the Data Dictionary for your given database and creates corresponding entries in:

- `providex.ddf` file (listing your defined tables by physical file's name)
- `providex.dde` file (listing the data sources' records/fields and descriptors).

As well, the data dictionary for each of your data source definitions is embedded in the corresponding physical database file. For more information on the ProvideX Data Dictionary, refer to the *NOMADS Reference Manual*.

The ProvideX ODBC driver reads the `providex.ddf` file to obtain a listing of tables/files from your Data Dictionary. As each data source defined in the `providex.ddf` file is accessed, the ProvideX ODBC driver reads its embedded data dictionary to determine the fields and the format of the data.

Any table defined in the `providex.ddf` file whose logical name begins with an asterisk `*` will not be made available to the user by the ODBC driver.

The ability to define non-normalized data files (i.e., files with multi-format records) is allowed in the Data Dictionary Maintenance; however, the ODBC driver will only recognize and use the first record format. To define non-normalized files, use an [INI Definition](#).

Fields Used by the ODBC Driver

The ProvideX ODBC driver uses the following fields from the Data Dictionary Maintenance > Element Description screen in NOMADS:

Name	Column name as displayed. Must be unique within the table. Maximum length is 30 characters.
Class	Optional field used to control the output type of the data. Maximum length is 30 characters. See Classes, p.29 .
External Only	Enable this flag to identify that the field exists as part of external key only.
Type and Format Mask	Type and format mask elements combined. This describes both the output type (String or Number) and how the data will be formatted in the record. See Type-Format Mask Combinations below for the complete list of element combinations.
Length	Precision, or maximum length, of the data field. The scale, or number of digits to the right of the decimal place, is optional; e.g., 6 Describes integer of 6 digits total. 6.2 Describes numeric of 6 digits total, 2 are right of decimal point. Maximum length is 6 digits total. Maximum precision is 999999. Maximum scale is 99.
Occurs	Dimensions of an array. If the value is a single number, such as 3, then it is considered to be a single element of an array rather than an entire array. If this field contains two values, colon separated, then the ODBC driver will generate multiple column names (<i>column name + underscore + numeric index</i>) for the elements in the array. For example, MyArray occurs 3 times (has 3 dimensions). The ODBC driver will generate column names: MyArray_1, MyArray_2, MyArray_3 to represent the elements in the array.

Type-Format Mask Combinations

The table below lists all of the Type and Format Mask element combinations used by the ODBC driver. The equivalent of the element combination for the INI definition appears in the column on the right.

Type - Format Mask	Description	INI Equivalent
String - Delimited	<i>Default.</i> String of variable length up to the size defined by Length. Field delimiter terminates field.	string, variable
String - Fixed	Trailing spaces are stripped during read.If the field is the last segment of an external key then it will <i>not</i> be padded with spaces during insert/update. Non-external key fields are padded with spaces during insert/update. Field has no field delimiter.	string, fixed
String - Padded	Always padded with spaces during insert/update. Fields are not stripped of trailing spaces during read. Field has no field delimiter.	string, nostrip
String - Substring	Always padded with spaces during insert/update. Fields are not stripped of trailing spaces during read. Field has no field delimiter.	string, substring
String - Last Substring	Always padded with spaces during insert/update. Fields are not stripped of trailing spaces during read. Field delimiter terminates field.	string, padded
Number - Delimited	Number of variable length up to the size defined by Length. Field delimiter terminates field.	numeric, variable
Number - Fixed	Sub-stringed field. Field has an implied decimal point if scale is provided. Field has no field delimiter.	numeric, fixed
Number - Padded	Sub-stringed field. Field has an implied decimal point if scale is provided. Field has no field delimiter.	numeric, nostrip
Number - Substring	Sub-stringed field. Field has no field delimiter.	numeric, substring
Number - Last Substring	Sub-stringed field. Field delimiter terminates field.	numeric, padded
Number - Binary Numeric	Sub-stringed field. Field has no field delimiter.	numeric, binary
Number - Decimal	Sub-stringed field which is number with an embedded decimal. Field has no field delimiter.	numeric, decimal
Number - Decimal Delimited	Sub-stringed field which is number with an embedded decimal. Field delimiter terminates field.	numeric, delimited
Number - Sign Fixed Numeric	Sub-stringed field. Field has no field delimiter.	numeric, signed
Number - Unsigned Integer	Sub-stringed field. Field has no field delimiter.	uni

INI Definition

Structured text files (INI files) may be used to manually define data that is not normalized (i.e., data sources with more than one record type), or cannot be handled by the ProvideX Data Dictionary. INIs are typically used to define files from legacy systems that were not created using the NOMADS Data Dictionary facilities.

These definition files consist of a table declaration section that assigns a logical table name to the physical path of each file. The logical names become section headings for column definitions. The maximum line length in an INI definition is 255 characters. The INI contents are described in the following sections.



Note: The square brackets enclosing section headings are part of the INI syntax. Other square brackets in the format examples below indicate optional elements.

Table Declaration

The [*tables*] declaration section is used to assign a logical name to a database's physical filename. For example:

```
[ *tables* ]
INVOICELINE=\INVOICE\INVLINE
Client= %c$+"cstfile"
```

The [*tables*] section heading is not case sensitive; however, square brackets, asterisks, and the word tables are all part of the required syntax. The syntax for assigning a logical table appears as follows:

```
table_name=path_filename[ ,alternate.INI][,SORTTABLE]
```

Where:

table_name Logical name assigned to the physical file. For example, `invoiceline` is the logical name for the `invline` file in the `invoice` directory:

```
[ *tables* ]
invoiceline=\invoice\invline
```

path_filename Physical location and file name of database in the system. Either absolute or relative path names can be specified. Relative path names are resolved based on the database directory setting in the ODBC driver configuration.

If the first character of the path is an equals sign =, the ProvideX ODBC driver treats the path as an expression and replaces all instances of %C\$ with *company*, %U\$ with *user ID*, and %S with *session ID* that are supplied during the connection; e.g.,

```
Client= %c$+"cstfile"
```

In this example, if ABC is entered in the company field of the ODBC driver, then `Client` would be evaluated to `ABCcstfile`.

- alternate*.INI Optional alternate INI definition file. Early Windows systems had a limit on the amount of information that could be stored in a single INI file. This option allows the definition to be spread over multiple INI files to keep the size of any one file below 64K.
- SORTTABLE Optional entry informing the ODBC driver that the column definitions are not defined in the file in physical order. The physical order is controlled through the use of the FIELD= keyword. The default is that all fields are defined in the physical order that they exist in the file.

Column Declaration

The record descriptors define logical columns extracted from the ProvideX data file with each entry consisting of:

- The column name as it appears to the user.
- Additional parameters separated by commas.

The minimum information required is a column name and its length. All columns default to *string, delimited*. The column descriptors can be in any order and are comma delimited. Only the first 3 characters of the keywords are required. Invalid keywords are ignored.

Column descriptors have the following format:

```
[table_name]
column_name = LENGTH=n,[type, formatting, attributes]
```

Where:

[table_name] Section heading for the column definition. This is the logical table name assigned to the file. Square brackets are part of the syntax.

column_name Logical name of the column; e.g.,

```
[Client]
CustomerID=STRING,LENGTH=6,FIELD=1,OFFSET=0
Name=LEN=20
```

In this example, *CustomerID* is the first column in the logical table [Client] and *Name* is the second.

LENGTH=n Mandatory value. Use a numeric expression or integer for *n*; e.g., LEN=30. If desired, you can set the number of digits to the right of the decimal; e.g., LEN=5 . 2.

<i>type</i>	Optional type. The following keywords set the type of the data:
	BNR Numeric values stored as a signed binary.
	LOGICAL Logical field - resulting output type is SQL_BIT.
	MAS90*YEAR Special MAS90 Year only format.
	NUMERIC Numeric value - in a ProvideX file, this is an an ASCII representation of the number.
	STRING ASCII string, <i>default</i> .
	UNI Data is an unsigned integer stored as a binary.
	UNSIGNEDBINARY Numeric value stored as unsigned binary.
<i>formatting</i>	Optional format mask. The following keywords describe the layout of data in the file:
	BINARY Numeric value stored as a signed binary as a sub-string of a longer field.
	DECIMAL Sub-stringed numeric with an embedded decimal. Numerics are right justified.
	DELIMITED Alternate description for PADDED with the exception of how numerics are handled. If the field is a numeric then it will be space-padded, right justified.
	FIXED Fixed length with no separator, trailing spaces stripped on read. Numerics are right justified.
	I86 Swapped. On Intel machines numbers are natively stored as swapped; e.g., 0001 is stored as 0100.
	NOSTRIP Sub-string - trailing spaces are never stripped. Same as the Data Dictionary formats Padded and Substring.
	PADDED Fixed length, padded to length and with a field delimiter. Same as the Data Dictionary format: Last Substring
	SIGNED Same as NUMERIC , FIXED except the 1st character of the field will have a negative sign (-).
	SUBSTRING Same as NOSTRIP. Added for consistency with Data Dictionary.
	VARIABLE Variable length delimited by \$8A\$, <i>default</i> .

<i>attributes</i>	Optional attributes that are not handled by the Data Dictionary:
CLASS= <i>str</i>	Class declaration. See Classes below.
FIELD= <i>n</i>	Logical column number in the record. Zero indicates "from start of record". The INI default is "in sequential order by position in the list".
FORMAT= <i>value</i>	Mask to be applied to the data when returned to the calling application. Maximum is 39 characters.
HIDE	Field is not in Data Dictionary (use for fields duplicated in key) and not available to user (use for filler values).
KEY	Defines external key fields.
MUSTBE="str"	String comparison for filtering data. If the condition is not met, the record is skipped. Maximum is 80
MUSTBE<"str"	characters. See Record Selection, p.31 .
MUSTBE>"str"	
NOSHOW	Field in Data Dictionary, data never returned.
OFFSET= <i>n</i>	Defines the offset (zero based) in the field.
RECTYPE= <i>value</i>	
or *RECTYPE	Flattens data. See Record Selection, p.31 .
SAMEAS	Used to link duplicate columns. This attribute is designed for columns which comprise an external key, and the data is duplicated in the record; e.g., CustId=len=6 CustId_dup=len=6, sameas=CustId,hide During insert/update operations, the data is copied from the column referenced to the target column.
SEPARATOR=	Delimiter for variable length field. Use the decimal value of your delimiter character; e.g., for the LF character (\$0A\$) you would use either SEPARATOR=10 or SEP=10.

Classes

Classes are used to define the format of special string or numeric data types; e.g., *date* values require special formatting. The class option can be used to convert to and from the SQL date format (YYYY-MM-DD) to the format of the date field stored in ProvideX files. The maximum length for a date field is 30 characters.

Since there are no rules on date formatting, separate keywords are available to assist the driver converting data to and from the SQL date format. Use keywords in the CLASS field to define a date in a Data Dictionary definition. Use CLASS= in an INI definition.

Date Formats

The syntax for a date definition appears as follows:

DATE[*keywords*]

DATE with no optional keywords defaults to YYYYMMDD.

The following secondary keywords can be included to further define the format:

- BIN Binary value; e.g., DATE-BIN-YYMMDD = BIN(990101, 4)
- PACK Packed numeric; e.g., DATE-PACK-YYMMDD = PCK(990101)
- BCD Binary packed decimal; e.g.,
DATE-BCD-YYMMDD = ATH(STR(990101))
- JUL Julian date. The default base year is 1970. The default year can be overridden by adding -YYYY, where YYYY represents the base year. For example, a base year of zero would be represented as DATE-JUL-0000.
- UNKNOWN Date value is processed as a string, without formatting and validation. This is provided for debugging purposes as the ODBC driver will report an error if a date string fails to convert to an SQL date.
- *MAS90 Best's MAS90 packed date.
- *SSI Aperum's FACTS packed date.
- AAMMDD AA or KK are special cases of YY. The first time a K or A is encountered and there have been no Y's then:
If the first character is greater than or equal to A, the year is $200 + \text{ASC}(\text{data}\$) - \text{ASC}('A')$; otherwise, the year is $190 + \text{ASC}(\text{data}\$) - \text{ASC}('0')$ or zero. All subsequent occurrences of A are treated as Y.
If the first character is K, the year is $190 + \text{ASC}(\text{data}\$) - \text{ASC}('0')$. All subsequent occurrences of K are treated as Y.
- KKMMDD

Example:

The INI field definitions for dates in a DATE_data record appear as follows:

```
[DATE_data]
Date_1=String, len=8, class=DATE-YYYYMMDD
Date_2=String, len=8, class=DATE-YY-MM-DD
Date_3=String, len=4, class=DATE-BCD-JUL
Date_4=String, len=4, class=DATE-PACK-YYYYMMDD
```

Right-Justified Data

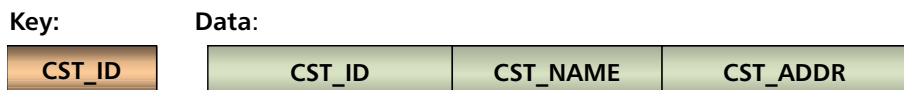
The format for right-justified data appears as follows:

```
RIGHT[nnn]
```

where *nnn* is the decimal value of the fill character. If a fill value is not supplied, then the fill defaults to a space (decimal 32).

External Keys

External keys are an issue when working with ODBC because the data may be duplicated on the file. This layout can be illustrated as follows:



When an external key is used in a ProvideX file, the key data can be stored as part of the record data as well. The ODBC driver prefixes the data record with the external key, which can result in key data being duplicated on the record:

CST_ID	CST_ID	CST_NAME	CST_ADDR
--------	--------	----------	----------

The solution is to *hide* the data from the ODBC end-user; e.g.,

```
[*Tables*]
Customer=\pvx\nomads\cstfile
[Customer]
CST_ID=STRING,LEN=6,FIXED
CST_ID_DUP=STRING,LEN=6,HIDE,SAMEAS=CST_ID
CST_NAME=STRING,LEN=30
CST_ADDR=STRING,LEN=30
```

However, the keyword HIDE is only available when using an INI file to define the data. HIDE is not supported when using the ProvideX Data Dictionary to define file layouts.

Keyed Files with External Keys - Direct Files

For the purposes of defining fields for a *Direct file*, the external key is inserted in front of the record as it is read from the file and passed to the ODBC system. For example, if the ProvideX file is created with ORD_NUMBER as the 6-byte external key, and ORD_CUSTOMER and ORD_AMOUNT as data, then the INI record descriptor would be:

```
[Order]
ORD_NUMBER=STRING,LEN=6,FIXED
ORD_CUSTOMER=STRING,LEN=10
ORD_AMOUNT=NUMERIC,LEN=10.2
```

If the key is duplicated in the data, you should expose the field that is the key and hide the duplicate that is within the data portion. The SQL optimizer will recognize the key field and be able sort the file much faster by using the key chain.

Record Selection

Because ProvideX has allowed users to evolve their applications, some developers have files that are not normalized. The following techniques are available for use in an INI file definition to convert a non-normalized data file logically into a normalized one.

Filtering the File Contents

This creates one logical table per record. The MUSTBE clause allows you to access specific record formats only. Any records found in the ProvideX data file that do not satisfy the MUSTBE condition are skipped. Filtering the file usually results in less rows in the logical tables than records in the physical data file.

Flattening the Data File

The `RECTYPE=` and `*RECTYPE` options allow you to create a logical table that contains all elements from all possible record formats. This preserves a one-to-one relationship between the rows in the logical table and the records in the physical file as all records can be represented as a row. This technique is compatible with migration to SQL.

Examples of Filtering and Flattening

This section describes how to represent non-normalized data file using either of the filtering or flattening techniques available in the ProvideX ODBC driver. In this example, the non-normalized data file `INVDTA` has two record types:

Record Type 1

Invoice_no	Line_no	Line_count	Customer_id	Order_dt
------------	---------	------------	-------------	----------

This is an invoice *header* record with a key of `Invoice_no` and `Line_no` (000 pseudo line number) with data fields of `Line_count`, `Customer_id`, and `Order_dt`.

Record Type 2

Invoice_no	Line_no	Product_no	Ord_qty	Sale_price
------------	---------	------------	---------	------------

This is an invoice *detail* record with a key of `Invoice_no` and `Line_no` with data fields of `Product_no`, `Ord_qty`, and `Sale_price`.

Filtering the Data. The example below filters the data in the `INVDTA` database by converting it into two data sources, `[InvoiceHeader]` and `[InvoiceDetail]`, both logical tables based on the value in `Line_no`:

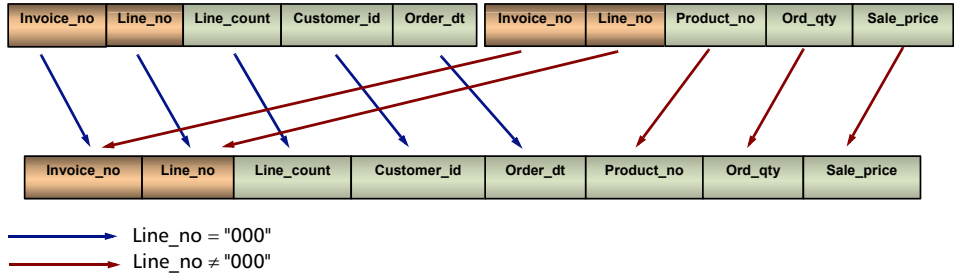
```
[ *Tables* ]
InvoiceHeader=invdta
InvoiceDetailLines=invdta

[ InvoiceHeader ]
Invoice_no=STRING,LEN=6
Line_no=STRING,LEN=3,MUSTBE="000",HIDE
Line_count=NUMERIC,LEN=4.0
Customer_id=STRING,LEN=6
Order_dt=STRING,LEN=8

[ InvoiceDetailLines ]
Invoice_no=STRING,LEN=6
Line_no=STRING,LEN=3,MUSTBE>"000"
Product_no=STRING,LEN=8
Ord_qty=NUMERIC,LEN=5.0
Sale_price=NUMERIC,LEN=8.2
```

If more than one field defines the record type, then the data must be filtered using the `MUSTBE` keyword. The maximum length of a `MUSTBE` value is 80 characters.

Flattening the Data File. A data file is *flattened* using the keywords *RECTYPE and RECTYPE=. When flattened, the fields for each record format exist in each row of the logical table.



For example, the data field `Line_no` would be declared the record type identifier (*RECTYPE clause). In the example below, the header records are identified by the `RECTYPE="000"` and the detail records by the `RECTYPE="~000"`. Note `FIELD=1` on the `Product_no` entry. The driver reads through the fields and if the `FIELD=1` is not there, the driver assumes that `Product_no` is the fourth field.

The value on the right of `RECTYPE=` can be multiple values; e.g., `Line_count` is part of 3 different record formats the `RECTYPE` value would appear as follows:

```
RECTYPE = "000001002"
```

Thus, `Line_count` would appear in record formats, "000", "001", and "002".

Example:

```
[*Tables*]
InvoiceData=invdta
[InvoiceData]
Invoice_no=STRING,LEN=6,FIXED
Invoice_line=STRING,LEN=3,FIXED,*RECTYPE
Line_count=NUMERIC,LEN=4.0,RECTYPE="000"
Customer_id=STRING,LEN=6,RECTYPE="000"
Order_dt=STRING,LEN=8,RECTYPE="000"
Product_no=STRING,LEN=8,RECTYPE="~000",FIELD=1
Ord_qty=NUMERIC,LEN=5.0,RECTYPE="~000"
Sale_price=NUMERIC,LEN=8.2,RECTYPE="~000"
```

The leading tilde ~ in the `RECTYPE="value"` clause indicates that the record data must not match the value given. The *RECTYPE keyword only allows for a single field per table to be defined. If multiple fields define the record type, then use the MUSTBE keyword.

Example Data and Definitions

The following example consists of all the possible field types:

```
STRINGDLM$="ABCD"
STRFIX$="EFGH"
STRPAD$="IJKL"
STRSUB$="MNOP"
STRLAST$="QRST"
NUMDLM=1.2
NUMFIX=3.4
NUMPAD=5.6
NUMSUB=7.8
NUMLAS=9.1
NUMBIN=2.3
NUMDEC=4.5
NUMDECDLM=6.7
NUMSGN=8.9
NUMUNS=12
LASSTR$="UVWX"
```

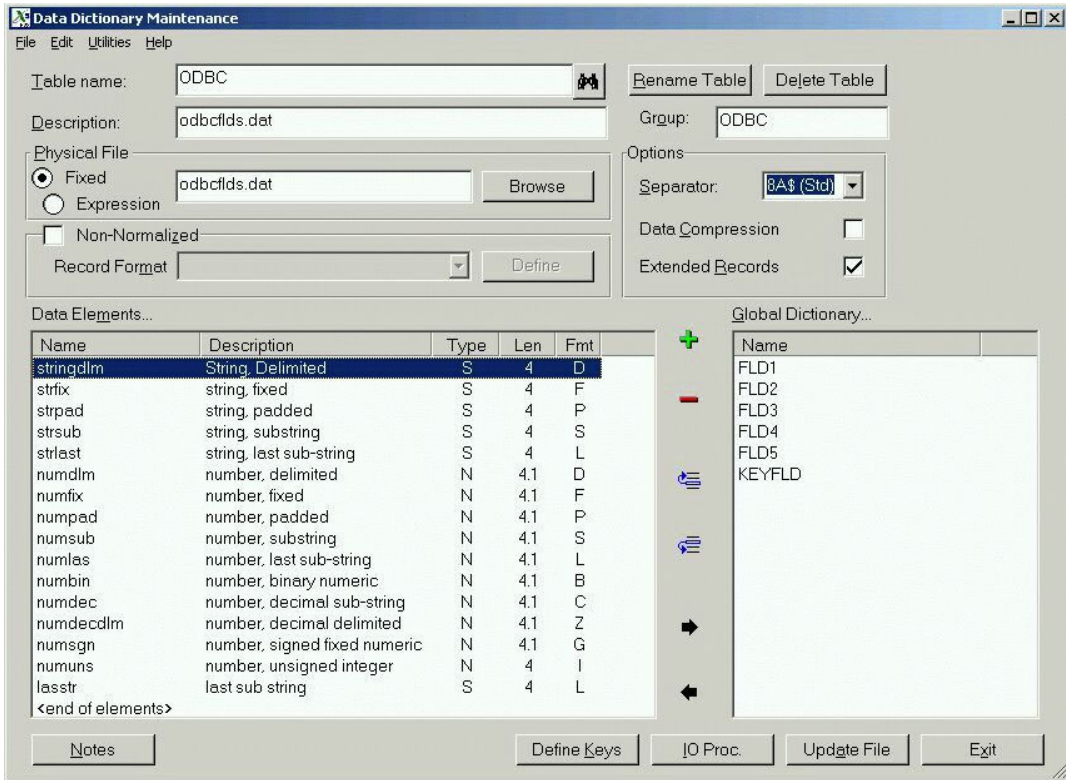
The following example shows the record as it would appear in the physical file. Values delimited by curly braces, "{ }", are hexadecimal values. Line breaks are for readability only:

```
ABCD{8a}EFGHIJKLMNOPQRST{8a}
1.2{8a}00340005600780091
{8a00000017} 4.5 6.7{8a}089+{0000000C}UVWX{8a}
```

The INI file definition appears as follows:

```
[*Tables*]
ODBC = odbcflds.dat
[ODBC]
stringdlm = STRING, FIELD=1, OFFSET=0, LEN=4, VARIABLE
strfix = STRING, FIELD=2, OFFSET=0, LEN=4, FIXED
strpad = STRING, FIELD=2, OFFSET=4, LEN=4, NOSTRIP
strsub = STRING, FIELD=2, OFFSET=8, LEN=4, SUBSTRING
strlast = STRING, FIELD=2, OFFSET=12, LEN=4, PADDED
numdlm = NUMERIC, FIELD=3, OFFSET=0, LEN=4.1, VARIABLE
numfix = NUMERIC, FIELD=4, OFFSET=0, LEN=4.1, FIXED
numpad = NUMERIC, FIELD=4, OFFSET=4, LEN=4.1, FIXED
numsub = NUMERIC, FIELD=4, OFFSET=8, LEN=4.1, FIXED
numlas = NUMERIC, FIELD=4, OFFSET=12, LEN=4.1, PADDED
numbin = NUMERIC, FIELD=5, OFFSET=0, LEN=4.1, BINARY
numdec = NUMERIC, FIELD=5, OFFSET=4, LEN=4.1, DECIMAL
numdecdlm = NUMERIC, FIELD=5, OFFSET=8, LEN=4.1, DELIMITED
numsgn = NUMERIC, FIELD=6, OFFSET=0, LEN=4.1, SIGNED
numuns = NUMERIC, FIELD=6, OFFSET=4, LEN=4, UNSIGNEDBINARY
lasstr = STRING, FIELD=6, OFFSET=8, LEN=4, PADDED
```

The Data Dictionary definition appears as follows:



Generating INI Table Definitions

INI table definitions may be generated in NOMADS Data Dictionary Maintenance by selecting Generate external/INI file contents from the Utilities menu. Select a table name and press Generate. The table entry is generated and displayed and the contents may be cut and pasted from the display, or exported to a text file.

INI table definitions may also be generated using the following program call:

```
CALL "**Dict/Defini",Contents$,ErrMsg$,TableName$,DDFpath$
```

Where:

Contents\$ Returns a string containing the table definition.

ErrMsg\$ Returns an error message if problems are encountered during generation, or null if successful.

TableName\$ Logical name of the table for which the definition is generated.

DDFpath\$ Path of the providex.ddf file containing the table definition.

In the case of tables with multiple record formats, a table entry is created for each format, using the `MUSTBE` clause to identify the field used as the record type indicator.

Using the ODBC Driver

Primarily, ODBC is used to provide *read access* to data files from other products such as Crystal reports, Excel, or Word. Most programming languages have an ODBC access facility to allow files to be read or updated as well. The process of bringing ProvideX data to your application begins with making a data connection — the steps involved will vary according to your application and your server technology. For specifics on ProvideX ODBC DSNs and connection information, refer to the [Local & Client Configuration, p.8](#).

Once a data connection is established, the ProvideX data itself can be accessed from the database using SQL commands. As mentioned earlier, SQL (Structured Query Language) is the standard interactive and programming language for accessing and manipulating databases. This sections describes the specific elements of SQL that can be used with ProvideX ODBC.

Statements

An SQL *statement* is used to perform various operations on the database. The ProvideX ODBC driver supports four types of SQL statements: `SELECT`, which retrieves data from the database; `INSERT`, which adds new data to the database; `DELETE`, which removes data from the database; and `UPDATE`, which changes data in the database.

Joining

SQL statements operate with logical sets of data — they declare what data is required, not how the data is to be retrieved. When data is required from two tables, the statement must establish a relationship between Table 1 and Table 2. In SQL, this concept is called *joining*. The join operation selects rows from two different tables such that the value in one column of Table 1 also appears in a column of Table 2.

For example, a *customer* table includes a code for sales representatives called `SALESREP` and the *sales representative* table includes a `SALESREP` code among other information about sales representatives (names, addresses etc). A join relationship between the *customer* table and *sales representative* table can be established because they each have a `SALESREP` column.

The ProvideX ODBC driver supports three types of joins:

<code>CROSS JOIN</code>	Returns all the records in Table 2 for each common record in Table 1 . To create a cross join, a comma is used to separate the declared tables. As of Version 3.32, the keywords <code>CROSS JOIN</code> can be used in place of the comma; e.g., <code>SELECT * FROM Customer, SalesReps</code>
-------------------------	---

[INNER] JOIN	Discards unmatched rows in either table. As of Version 3.32, the keyword INNER is optional; e.g., SELECT * FROM { oj Customer INNER JOIN SalesReps ON Customer.SALESREP = SalesReps.SALESREP }
LEFT [OUTER] JOIN	Will, for each record in Table 1, join the matching record in Table 2, if any. As of Version 3.32, the keyword OUTER is optional; e.g., SELECT * FROM { oj Customer LEFT OUTER JOIN SalesReps ON Customer.SALESREP = SalesReps.SALESREP }

Providex Views

Please note that ProvideX Views requires the ProvideX COM interface and is therefore only available with the ProvideX Local ODBC driver or the ProvideX ODBC Server for Windows. Views also requires a copy of ProvideX Version 5.1 or later be installed.

SQL Syntax Table

The ProvideX ODBC driver supports the SQL syntax described in the table below. For an illustration of this syntax, see [Constructing a Left Outer Join Using the Syntax Table, p.47](#).

In the following table, SQL keywords are shown in uppercase, vertical bars (pipes) "|" separate choices where more than one command is represented, and a *blank* indicates that no qualifier is required:

Syntax	Description
<i>statement</i>	SELECT <i>select</i> <i>orderby</i> SELECT <i>insert</i> SELECT <i>delete</i> SELECT <i>update</i>
<i>select</i>	<i>selectcols</i> SELECT <i>tablelist</i> <i>where</i> <i>groupby</i> <i>having</i>
<i>delete</i>	SELECT <i>tablename</i> <i>where</i>
<i>insert</i>	SELECT <i>tablename</i> <i>insertvals</i>
<i>update</i>	<i>tablename</i> SELECT <i>setlist</i> <i>where</i>
<i>setlist</i>	<i>set</i> <i>setlist</i> , <i>set</i>
<i>set</i>	<i>columnname</i> SELECT <i>columnname</i> = <i>expression</i>
<i>insertvals</i>	(<i>columnlist</i>) VALUES (<i>valuelist</i>) VALUES (<i>valuelist</i>) (<i>columnlist</i>) VALUES (SELECT <i>select</i>) VALUES (SELECT <i>select</i>)
<i>columnlist</i>	<i>columnname</i> , <i>columnlist</i> <i>columnname</i>
<i>valuelist</i>	NULL , <i>valuelist</i> <i>expression</i> , <i>valuelist</i> <i>expression</i> NULL
<i>selectcols</i>	<i>selectallcols</i> * <i>selectallcols</i> <i>selectlist</i>
<i>selectallcols</i>	<i>blank</i> ALL DISTINCT

Syntax	Description
<i>selectlist</i>	<i>selectlistitem</i> , <i>selectlist</i> <i>selectlistitem</i>
<i>selectlistitem</i>	<i>expression</i> <i>expression aliasname</i> <i>expression AS aliasname</i> <i>aliasname.*</i> <i>colref</i>
<i>where</i>	<i>blank</i> WHERE <i>boolean</i>
<i>having</i>	<i>blank</i> HAVING <i>boolean</i>
<i>boolean</i>	<i>and</i> <i>and</i> OR <i>boolean</i>
<i>and</i>	<i>not</i> <i>not</i> AND <i>and</i>
<i>not</i>	<i>comparison</i> NOT <i>comparison</i>
<i>comparison</i>	(<i>boolean</i>) <i>colref</i> IS NULL <i>colref</i> IS NOT NULL <i>expression</i> LIKE <i>pattern</i> <i>expression</i> NOT LIKE <i>pattern</i> <i>expression</i> IN (<i>valuelist</i>) <i>expression</i> NOT IN (<i>valuelist</i>) <i>expression op expression</i> EXISTS (SELECT <i>select</i>) <i>expression op selectop</i> (SELECT <i>select</i>) <i>expression</i> IN (SELECT <i>select</i>) <i>expression</i> NOT IN (SELECT <i>select</i>) <i>expression</i> BETWEEN <i>expression</i> AND <i>expression</i>) <i>expression</i> NOT BETWEEN <i>expression</i> AND <i>expression</i>)
<i>selectop</i>	<i>blank</i> ALL ANY
<i>op</i>	> >= < <= = <>
<i>pattern</i>	<i>string</i> ? USER
<i>expression</i>	<i>expression</i> + <i>times</i> <i>expression</i> - <i>times</i> <i>times</i>
<i>times</i>	<i>times</i> * <i>neg</i> <i>times</i> / <i>neg</i> <i>neg</i>
<i>neg</i>	<i>term</i> + <i>term</i> - <i>term</i>
<i>term</i>	(<i>expression</i>) <i>colref</i> <i>simpleterm</i> <i>aggterm scalar</i>
<i>scalar</i>	<i>scalarescape</i> <i>scalarshorthand</i>
<i>scalarescape</i>	--*(VENDOR (MICROSOFT), PRODUCT (ODBC) FN <i>fn</i>)*--
<i>scalarshorthand</i>	{ FN <i>fn</i> }
<i>fn</i>	<i>functionname</i> (<i>valuelist</i>) <i>functionname</i> ()
<i>aggterm</i>	COUNT (*) AVG (<i>expression</i>) MAX (<i>expression</i>) MIN (<i>expression</i>) SUM (<i>expression</i>) COUNT (<i>expression</i>)
<i>simpleterm</i>	<i>string</i> <i>realnumber</i> ? USER <i>date</i> <i>time</i> <i>timestamp</i>
<i>groupby</i>	<i>blank</i> GROUP BY <i>groupbyterms</i>
<i>groupbyterms</i>	<i>colref</i> <i>colref</i> , <i>groupbyterms</i>
<i>orderby</i>	<i>blank</i> ORDER BY <i>orderbyterms</i>
<i>orderbyterms</i>	<i>orderbyterm</i> <i>orderbyterm</i> , <i>orderbyterms</i>
<i>orderbyterm</i>	<i>colref asc</i> <i>integer asc</i>
<i>asc</i>	<i>blank</i> ASC DESC

Syntax	Description
<i>colref</i>	<i>aliasname . columnname</i> <i>columnname</i>
<i>tablelist</i>	<i>tablelistitem , tablelist</i> <i>tablelistitem</i>
<i>tablelistitem</i>	<i>tableref</i> <i>outerjoin</i>
<i>outerjoin</i>	<i>ojescape</i> <i>ojshorthand</i>
<i>ojescape</i>	--*(VENDOR(MICROSOFT),PRODUCT(ODBC) OJ <i>oj</i>)*--
<i>ojshorthand</i>	{ OJ <i>oj</i> }
<i>inneroj</i>	<i>tableref</i> INNER JOIN <i>tableref</i> ON <i>boolean</i> <i>tableref</i> INNER JOIN <i>inneroj</i> ON <i>boolean</i>
<i>oj</i>	<i>tableref</i> LEFT OUTER JOIN <i>tableref</i> ON <i>boolean</i> <i>tableref</i> LEFT OUTER JOIN <i>oj</i> ON <i>boolean</i> <i>inneroj</i>
<i>tableref</i>	<i>tablename</i> <i>tablename aliasname</i>
<i>indexname</i>	<i>identifier</i>
<i>functionname</i>	<i>identifier</i> (see Scalar Functions , below).
<i>tablename</i>	<i>identifier</i>
<i>datatype</i>	<i>identifier</i>
<i>columnname</i>	<i>identifier</i>
<i>aliasname</i>	<i>identifier</i>
<i>identifier</i>	Identifier (must be enclosed in double quotes if it contains spaces).
<i>string</i>	String (enclosed in single quotes).
<i>realnumber</i>	Non-negative real number (including E notation).
<i>integer</i>	Non-negative integer.
<i>date</i>	<i>dateescape</i> <i>dateshorthand</i>
<i>dateescape</i>	--*(VENDOR(MICROSOFT),PRODUCT(ODBC) d <i>dateval</i>)*--
<i>dateshorthand</i>	{ d <i>dateval</i> }
<i>dateval</i>	Date in <i>yyyy-mm-dd</i> format in single quotes; e.g., '1996-02-05'.
<i>time</i>	<i>timeescape</i> <i>timeshorthand</i>
<i>timeescape</i>	--*(VENDOR(MICROSOFT),PRODUCT(ODBC) t <i>timeval</i>)*--
<i>timeshorthand</i>	{ t <i>timeval</i> }
<i>timeval</i>	Time in <i>hh:mm:ss</i> format in single quotes; e.g., '10:19:48')
<i>timestamp</i>	<i>timestampescape</i> <i>timestampshorthand</i>
<i>timestampescape</i>	--*(VENDOR(MICROSOFT),PRODUCT(ODBC) ts <i>timestampval</i>)*--
<i>timestampshorthand</i>	{ ts <i>timestampval</i> }
<i>timestampval</i>	Timestamp in <i>yyyy-mm-dd hh:mm:ss[.ffffff]</i> format in single quotes; e.g., '1996-02-05 10:19:48.529'.

Scalar Functions

Scalar functions are supported through the use of the escape sequence:

```
{ fn scalar function }
```

The argument *scalar function* can be any of the string, numeric, or time and date functions listed in this section. The supported scalar functions are listed with descriptions of their results in the sections that follow:

String Functions

ASCII (<i>string</i>)	Integer representing the ASCII code value of the leftmost character of <i>string</i> .
BIT_LENGTH (<i>string</i>)	Length in bits of <i>string</i> .
CHAR (<i>num</i>)	Character that has the ASCII code value specified by <i>num</i> . The value of <i>num</i> should be between 0 and 255.
CHAR_LENGTH (<i>string</i>) or CHARACTER_LENGTH (<i>string</i>)	Length in characters of the <i>string</i> , if <i>string</i> is of a character data type.
CONCAT (<i>string1</i> , <i>string2</i>)	Character string that is the result of concatenating <i>string2</i> to <i>string1</i> .
DIFFERENCE (<i>string1</i> , <i>string2</i>)	Integer value that indicates the difference between the values returned by the SOUNDEX function for <i>string1</i> and <i>string2</i> .
INSERT (<i>string1</i> , <i>start</i> , <i>length</i> , <i>string2</i>)	Character string where length characters have been deleted from <i>string1</i> , beginning at <i>start</i> , and where <i>string2</i> has been inserted into <i>string1</i> , beginning at <i>start</i> .
LCASE (<i>string</i>)	String equal to that in <i>string</i> , with all uppercase characters converted to lowercase.
LEFT (<i>string</i> , <i>count</i>)	Leftmost count characters of <i>string</i> .
LENGTH or LEN (<i>string</i>)	Number of characters in <i>string</i> , excluding trailing blanks.
LOCATE (<i>string1</i> , <i>string2</i> [, <i>start</i>])	Starting position of the first occurrence of <i>string1</i> within <i>string2</i> . The search for the first occurrence of <i>string1</i> begins with the first character position in <i>string2</i> unless the optional argument, <i>start</i> , is specified. If <i>start</i> is specified, the search begins with the character position indicated by the value of <i>start</i> . The first character position in <i>string2</i> is indicated by the value 1. If <i>string1</i> is not found within <i>string2</i> , the value 0 is returned.
LTRIM (<i>string</i>)	Characters of <i>string</i> , with leading blanks removed.

OCTET_LENGTH (<i>string</i>)	Returns the length in bytes of <i>string</i> .
POSITION (<i>string1</i> IN <i>string2</i>)	Position of <i>string1</i> in <i>string2</i> . The result is an exact numeric with precision of double and a scale of 0.
REPEAT (<i>string</i> , <i>count</i>)	Character string composed of <i>string</i> repeated <i>count</i> times.
REPLACE (<i>string1</i> , <i>string2</i> , <i>string3</i>)	Search <i>string1</i> for occurrences of <i>string2</i> , and replace with <i>string3</i> .
RIGHT (<i>string</i> , <i>count</i>)	Rightmost <i>count</i> characters of <i>string</i> .
RTRIM (<i>string</i>)	Characters of <i>string</i> with trailing blanks removed.
SOUNDEX (<i>string</i>)	4-digit SOUNDEX code.
SPACE (<i>count</i>)	Character string consisting of <i>count</i> spaces.
SUBSTRING (<i>string</i> , <i>start</i> , <i>length</i>) or SUBSTR (<i>string</i> , <i>start</i> , <i>length</i>)	Character string that is derived from <i>string</i> , beginning at the character position specified by <i>start</i> for <i>length</i> characters.
UCASE (<i>string</i>)	String equal to that in <i>string</i> , but with all lowercase characters converted to uppercase.

Numeric Functions

ABS (<i>num</i>)	Absolute value of <i>num</i> .
ACOS (<i>float</i>)	Arccosine of <i>float</i> as an angle, expressed in radians.
ASIN (<i>float</i>)	Arcsine of <i>float</i> as an angle, expressed in radians.
ATAN (<i>float</i>)	Arctangent of <i>float</i> as an angle, expressed in radians.
ATAN2 (<i>float1</i> , <i>float2</i>)	Arctangent of the x and y coordinates, specified by <i>float1</i> and <i>float2</i> , respectively, as an angle, expressed in radians.
CEILING (<i>num</i>)	Smallest integer greater than or equal to <i>num</i> . The return value is of the same data type as the input parameter.
COS (<i>float</i>)	Cosine of <i>float</i> , where <i>float</i> is an angle expressed in radians.
COT (<i>float</i>)	Cotangent of <i>float</i> , where <i>float</i> is an angle expressed in radians.
DEGREES (<i>num</i>)	Number of degrees converted from <i>num</i> radians.
EXP (<i>float</i>)	Exponential value of <i>float</i> .
FLOOR (<i>num</i>)	Largest integer less than or equal to <i>num</i> . The return value is of the same data type as the input parameter.
LOG (<i>float</i>)	Natural logarithm of <i>float</i> .
LOG10 (<i>float</i>)	Base 10 logarithm of <i>float</i> .
MOD (<i>int1</i> , <i>int2</i>)	Remainder (modulus) of <i>int1</i> divided by <i>int2</i> .

PI ()	Constant value of pi as a floating-point value. Pi is defined internally as : 3.14159265358979323846264338327950288419716939937510
POWER(num, int)	Value of <i>num</i> to the power of <i>int</i> .
RADIANS(num)	Number of radians converted from <i>num</i> degrees.
RAND([int])	Random floating-point value using <i>int</i> as the optional seed value.
ROUND(num, int)	Returns <i>num</i> rounded to <i>int</i> places right of the decimal point. If <i>int</i> is negative, <i>num</i> is rounded <i>int</i> places to the left of the decimal point.
SIGN(num)	Returns an indicator of the sign of <i>num</i> . If <i>num</i> is less than zero, -1 is returned. If <i>num</i> equals zero, 0 is returned. If <i>num</i> is greater than zero, 1 is returned.
SIN(float)	Sine of <i>float</i> , where <i>float</i> is an angle expressed in radians.
SQRT(float)	Square root of <i>float</i> .
TAN(float)	Tangent of <i>float</i> , where <i>float</i> is an angle expressed in radians.
TRUNCATE(num, int)	Returns <i>num</i> truncated to <i>int</i> places right of the decimal point. If <i>int</i> is negative, <i>num</i> is truncated <i>int</i> places to the left of the decimal point.

Time and Date Functions

CURRENT_DATE ()	Current date.
CURRENT_TIME[(time-precision)]	Current local time. The <i>time-precision</i> argument determines the seconds precision of the returned value.
CURRENT_TIMESTAMP[(timestamp-precision)]	Current local date and local time as a timestamp value. The <i>timestamp-precision</i> argument determines the seconds precision of the returned timestamp.
CURDATE ()	Current date.
CURTIME ()	Current local time.
DAYNAME(date_exp)	Character string containing the name of the day for the day portion of <i>date_exp</i> . Only long English names are returned; e.g., Monday through Sunday.
DAYOFMONTH(date_exp)	Day of the month based on the month field in <i>date_exp</i> as an integer value in the range of 1-31.

DAYOFWEEK(*date_exp*) Day of the week based on the week field in *date_exp* as an integer value in the range of 1-7, where 1 represents Sunday.

DAYOFYEAR(*date_exp*) Day of the year based on the year field in *date_exp* as an integer value in the range of 1-366.

EXTRACT(*extract-field* FROM *extract-source*)

Returns the *extract-field* portion of the *extract-source*. The *extract-source* argument is a date time or interval expression. The *extract-field* argument can be one of the following keywords:

YEAR	MONTH	DAY	HOURL
MINUTE	SECOND		

The precision of the returned value is implementation-defined. The scale is 0 unless **SECOND** is specified, in which case, the scale is not less than the fractional seconds precision of the *extract-source* field.

HOUR(*time_exp*) Hour based on the hour field in *time_exp* as an integer value in the range of 0-23.

MINUTE(*time_exp*) Minute based on the minute field in *time_exp* as an integer value in the range of 0-59.

MONTH(*date_exp*) Month based on the month field in *date_exp* as an integer value in the range of 1-12.

MONTHNAME(*date_exp*)

Character string containing the name of the month for the month portion of *date_exp*. Only long English names are returned; e.g., January through December.

NOW () Current date and time as a timestamp value.

QUARTER(*date_exp*) Quarter in *date_exp* as an integer value in the range of 1-4, where 1 represents January 1 through March 31.

SECOND(*time_exp*) Second based on the second field in *time_exp* as an integer value in the range of 0-59.

TIMESTAMPADD(*interval, integer_exp, timestamp_exp*)

Returns the timestamp calculated by adding *integer_exp* intervals of type *interval* to *timestamp_exp*. Valid values of *interval* include the following keywords:

SQL_TSI_SECOND	SQL_TSI_MINUTE	SQL_TSI_HOUR
SQL_TSI_DAY	SQL_TSI_WEEK	SQL_TSI_MONTH
SQL_TSI_QUARTER	SQL_TSI_YEAR	

For example, the following SQL statement returns the name of each employee and his or her one-year anniversary date:

```
SELECT NAME, {fn TIMESTAMPADD(SQL_TSI_YEAR, 1, HIRE_DATE)}
FROM EMPLOYEES
```

If *timestamp_exp* is a time value and *interval* specifies days, weeks, months, quarters, or years, the date portion of *timestamp_exp* is set to the current date before calculating the resulting timestamp. If *timestamp_exp* is a date value and *interval* specifies seconds, minutes, or hours, the time portion of *timestamp_exp* is set to 0 before calculating the resulting timestamp.

TIMESTAMPDIFF(interval, timestamp_exp1, timestamp_exp2)

Returns the integer number of intervals of type *interval* by which *timestamp_exp2* is greater than *timestamp_exp1*. Valid values of *interval* include the following keywords:

```
SQL_TSI_SECOND SQL_TSI_MINUTE SQL_TSI_HOUR
SQL_TSI_DAY SQL_TSI_WEEK SQL_TSI_MONTH
SQL_TSI_QUARTER SQL_TSI_YEAR
```

For example, the following SQL statement returns the name of each employee and the number of years he or she has been employed:

```
SELECT NAME, {fn TIMESTAMPDIFF(SQL_TSI_YEAR, {fn
CURDATE()}, HIRE_DATE)} FROM EMPLOYEES
```

If either *timestamp* expression is a time value and *interval* specifies days, weeks, months, quarters, or years, the date portion of that timestamp is set to the current date before calculating the difference between the timestamps. If either *timestamp* expression is a date value and *interval* specifies seconds, minutes, or hours, the time portion of that timestamp is set to 0 before calculating the difference between the timestamps.

WEEK(date_exp)

Week of the year based on the week field in *date_exp* as an integer value in the range of 1-53.

YEAR(date_exp)

Year based on the year field in *date_exp* as an integer value.

Example SQL

The examples below use two tables, Customer and SalesRep. The Customer table contains four fields CustomerId, Name, SalesRepId and ARBalance. The SalesRep table contains two fields SalesRepId and Name.

The Customer table contains two rows

CustomerId	Name	SalesRepId	ARBalance
0001	ABC Corp	01	1234.99
0002	Acme Inc		1.23

The SalesRep table contains three rows

SalesRepId	Name
01	John Doe
02	Jane Smith
03	House Account

The following tables illustrate the results of the three different joins.

Cross Join

```
SELECT Customer.Name, SalesRep.Name
FROM Customer Customer, SalesRep SalesRep
```

Result set ...

ABC Corp	John Doe
ABC Corp	Jane Smith
ABC Corp	House Account
Acme Inc	John Doe
Acme Inc	Jane Smith
Acme Inc	House Account

Inner Join

```
SELECT Customer.Name, SalesRep.Name FROM { OJ Customer Customer
INNER JOIN SalesRep SalesRep ON Customer.SalesRepId =
SalesRep.SalesRepId }
```

Result set ...

ABC Corp	John Doe
----------	----------

Left Outer Join

```
SELECT Customer.Name, SalesRep.Name FROM { OJ Customer Customer
      LEFT OUTER JOIN SalesRep SalesRep ON Customer.SalesRepId =
      SalesRep.SalesRepId }
```

Result set ...

ABC Corp	John Doe
Acme Inc	

Constructing a Left Outer Join Using the Syntax Table

The following example applies the syntax descriptors explained in the [SQL Syntax Table](#). In the descriptions below, a "::=" symbol represents the phrase "consists of" and a "|" symbol represents an exclusive OR.

Suppose we want a statement that will retrieve all rows from the Customer table, we want to display the customer's name, and we also want to display the name of the sales representative when the customer has a sales representative assigned.

Because the Customer table and the SalesRep table both contain a column called Name we must use an *alias* so that the ODBC driver can determine which column we are referring to.

We begin with a *statement* ::= SELECT *select orderby* which consists of a *select* ::= *selectcols* FROM *tablelist where groupby having*, which has *selectcols* ::= *selectallcols* * | *selectallcols selectlist*. We are want to display a limited number of columns so we want a *selectlist* ::= *selectlistitem* , *selectlist* | *selectlistitem*, which consists of two *selectlistitem* ::= *expression* | *expression aliasname* | *expression AS aliasname* | *aliasname.** | *colref*. Our two *selectlist* items are *colref* ::= *aliasname . columnname* | *columnname* which are composed of an *aliasname* ::= *identifier* and a *columnname* ::= *identifier*. An *identifier* consists of an identifier (identifiers containing spaces must be enclosed in double quotes).

As the alias we have chosen to use the name of the table; however, an alias is not limited to the table name. Thus far we have:

```
SELECT Customer.Name, SalesRep.Name FROM
```

Now we parse the *tablelist*. A *tablelist* ::= *tablelistitem* , *tablelist* | *tablelistitem* where a *tablelistitem* ::= *tableref* | *outerjoin* and we want all rows from the first table and matching rows from the second table, if any. Therefore we need an *outerjoin* ::= *ojescape* | *ojshorthand*. Since we don't like to type we use *ojshorthand* ::= { OJ *oj* } where *oj* := *tableref* LEFT OUTER JOIN *tableref* ON *boolean*. We have two *tableref* ::= *tablename* | *tablename aliasname* where *tablename* ::= *identifier*.

Thus far we have:

```
SELECT Customer.Name, SalesRep.Name
       FROM { OJ Customer Customer
             LEFT OUTER JOIN SalesRep SalesRep ON }
```

The final piece is the relationship between the `Customer` table and the `SalesRep` table which is specified with *boolean ::= and | and OR boolean*. Ultimately we want a *comparison* which is part of *not ::= comparison | NOT comparison* which is part of *and ::= not | not AND and*. The *comparison* is *expression op expression* where the expressions are *colref* and the *op* is an `=`.

The result is:

```
SELECT Customer.Name, SalesRep.Name
       FROM { OJ Customer Customer
             LEFT OUTER JOIN SalesRep SalesRep
             ON Customer.SalesRepId = SalesRep.SalesRepId }
```