*xf*ODBC User’s Guide

Version 10.1

Printed: June 2013

DCN OD-01-10.1_02

Synergex
2330 Gold Meadow Way
Gold River, CA 95670 USA

http://www.synergex.com
phone 916.635.7300
fax 916.635.6549

Contents

Preface

About this manual

This guide serves two groups of ODBC users: developers who create Synergy™ databases and are responsible for making them ODBC-enabled, and end-users who use *xf*ODBC as a tool for accessing data. With these different audiences in mind, we divided the *xfODBC User's Guide* into three parts: Part 1 provides an overview of *xf*ODBC and ODBC technology and should be read by both developers and end-users. Part 2 is written for developers and explains how to generate system catalogs from Synergy repository definitions. Part 3 is intended for developers end-users and explains the steps required to access data.

The *xfODBC User's Guide* is written for users who are already familiar with database concepts and terminology and with the platforms discussed in this guide.

Manual conventions

Throughout this manual, we use the following conventions:

- In code syntax, text that you type is in `Courier` typeface. Variables that either represent or should be replaced with specific data are in *italic* type.
- Optional arguments are enclosed in *[*italic square brackets*]*. If an argument is omitted and the comma is outside the brackets, a comma must be used as a placeholder, unless the omitted argument is the last argument in a subroutine. If an argument is omitted and the comma is inside the brackets, the comma may also be omitted.
- Arguments that can be repeated one or more times are followed by an ellipsis...
- A vertical bar (|) in syntax means to choose between the arguments on each side of the bar.
- Data types are **boldface**. The data type in parentheses at the end of an argument description (for example, (**n**)) documents how the argument will be treated within the routine. An **a** represents alpha, a **d** represents decimal or implied-decimal, an **i** represents integer, and an **n** represents numeric (which means the type can be **d** or **i**).
- The term "environment variable" refers to logicals on OpenVMS, as well as environment variables on Windows and UNIX platforms.
- To "enter" data means to type it and then press ENTER. ("ENTER" refers to either the ENTER key or the RETURN key, depending on your keyboard.)

WIN

Items or discussions that pertain only to a specific operating system or environment are called out with the name of the operating system.

Other resources

- Synergy/DE™ Connectivity Series release notes (**REL_CONN.TXT**) and the release notes for the Synergy/DE Data Provider for .NET (**REL_SDP.TXT**)
- *Installation Configuration Guide*
- *Repository User's Guide*
- *Environment Variables & System Options*
- *Getting Started with Synergy/DE*

Product support information

If you cannot find the information you need in this manual or in the resources listed above, you can reach the Synergy/DE Developer Support department at the following numbers:

800.366.3472 (in the U.S. and Canada)
916.635.7300 (in all other locations)

To learn about your Developer Support options, contact your Synergy/DE account manager at one of the above numbers.

Before you contact us, be sure you have the following information:

- The version of the Synergy/DE product(s) you are running (both client and server)
- The name and version of the operating system you are running
- The name and version of the ODBC-enabled application you are using
- The hardware platform you are using
- The error mnemonic and any associated error text (for ODBC-reported errors)
- The exact steps that preceded the problem
- What changed (for example, code, data, or hardware) before the problem occurred
- Whether the problem happens every time and whether it is reproducible in a small test program
- The following *xf*ODBC logs: Synergy DBMS logging (on Windows—use SDMS_AUDIT_SRV), Vortex API logging (use VORTEX_API_LOGFILE and VORTEX_API_LOGOPTS=FULL), ODBC trace logging. See "Error Logging" on page 11-2 for information on *xf*ODBC logging.

Note that Synergex® provides support for ODBC access to Synergy data. Synergex cannot support your ODBC application and may refer you to your application provider.

Synergex Professional Services Group

If you would like assistance implementing new technology or would like to bring in additional experienced resources to complete a project or customize a solution, Synergex Professional Services Group (PSG) can help. PSG provides comprehensive technical training and consulting services to help you take advantage of Synergex's current and emerging technologies. For information and pricing, contact your Synergy/DE account manager at 800.366.3472 (in the U.S. and Canada) or 916.635.7300.

Comments and suggestions

We welcome your comments and suggestions for improving this manual. Send your comments, suggestions, and queries, as well as any errors or omissions you've discovered, to **doc@synergex.com**.

Part 1: Introduction to *xf*ODBC

This section contains information for end-users and developers. It contains an overview of *xf*ODBC and describes the steps to making a Synergy database accessible to ODBC-enabled applications.

1

Welcome to *xf*ODBC

What Is *xf*ODBC?

*xf*ODBC is a package of components that enables you to make your Synergy data accessible to third-party applications that can use ODBC (applications such as Crystal Reports, Microsoft Access, Microsoft Query, and Visual Basic). For example, *xf*ODBC includes the *xf*ODBC Database Administrator (DBA) and **dbcreate** programs, which enable you to create system catalogs from your Synergy repository definitions, and it includes the *xf*ODBC driver, which uses system catalogs to provide ODBC access to your Synergy database. See "xfODBC components" on page 1-3 .

At the heart of *xf*ODBC is a technological standard called Open Database Connectivity (ODBC). Through ODBC drivers, ODBC enables a wide variety of applications to access various databases by ensuring that both the databases and the third-party applications conform to a standard set of rules for data access. When you instruct an ODBC-enabled application to communicate with a database, the application first calls the Microsoft-supplied ODBC Driver Manager. The ODBC Driver Manager then calls the ODBC driver that communicates with that particular database. (For *xf*ODBC, this is the *xf*ODBC driver.) The ODBC driver translates messages and data between the application and the database, using an ODBC version of SQL to communicate with the application, and using the database's version of SQL to communicate with the database.

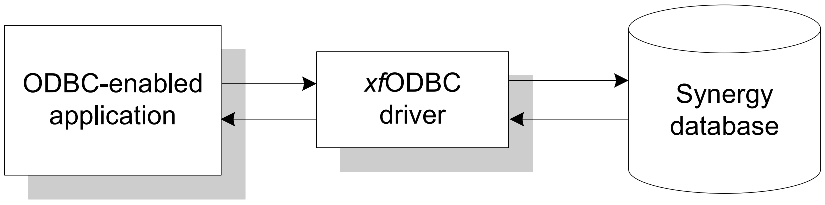


Figure 1-1. An ODBC-enabled application accessing a Synergy database.

*xf*ODBC supports up to 1,024 concurrent ODBC handles (which generally have a one-to-one correspondence with connections) and can access Synergy Database files, relative files, tagged files, and ASCII text files. (You can read data from ASCII text files, but you can't update them.)

The *xf*ODBC driver supports the ODBC version 2.5 API (level 1) and SQL92 entry level syntax plus extensions. It supports up to 64 table references (including inline views and table joins).

- See **RESTRICT.TXT**, a text file distributed with Synergy/DE, for information on restrictions to ODBC version 2.5 API support.
- See "Appendix B: SQL Support" for information on SQL statements, commands, and functions supported by *xf*ODBC.

For updating Synergy databases, we strongly recommend using a Synergy application that's designed to efficiently maintain database integrity. If you use an ODBC-enabled application to update a Synergy database, you may run into record-locking issues.

*xf*ODBC components

*xf*ODBC consists of several components. The main component, the *xf*ODBC driver, enables you to access your Synergy data from third-party applications. Before the driver can do this, however, your Synergy database must be prepared for ODBC access. To prepare a Synergy database for ODBC access, a system catalog must be generated from the database's repository files. System catalogs describe Synergy databases in a way that the *xf*ODBC driver can understand (see "System catalog" on page 1-5).

The following sections describe all of the files and components used to

- prepare a Synergy database for ODBC access (administrative components).
- access a Synergy database with the *xf*ODBC driver (runtime components).

Administrative components

Figure 1-2 illustrates how the administrative components work together to generate and verify a system catalog. These components are described in the sections that follow. (The generation process itself is described in detail in chapter 4, "Creating a System Catalog.")

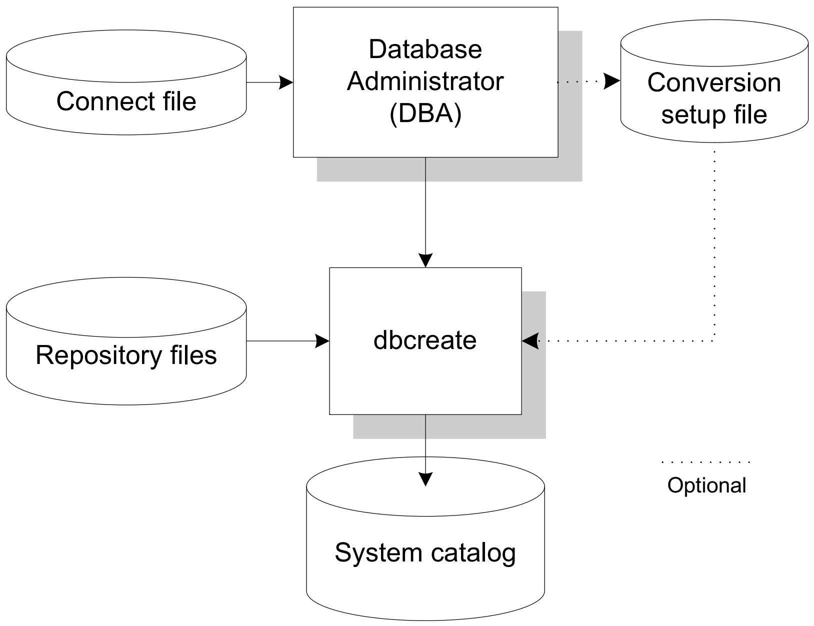


Figure 1-2. Administrative components generating a system catalog.

Synergy database

To use *xf*ODBC, you'll need a Synergy database. A Synergy database consists of files of one of the following types:

- Synergy ISAM files. We strongly recommend using ISAM files with *xf*ODBC because they enable the *xf*ODBC driver to create additional keys for optimization. A Synergy database can contain tagged files, which are ISAM files that contain multiple types of records.
- Relative files, which contain a single type of data record with a single record number key.
- ASCII text files. You can read data from ASCII text files sequentially, but you can't update them. There is no key.

The Synergy database is a runtime component. It is not directly involved in the creation of a system catalog. Repository files are used to create the system catalog.

Repository files

Repository files refer to the ISAM files generated by Synergy/DE Repository; these files describe and define the actual data files, providing index, tag, field, structure, and key information, along with other definitions. They often have the filenames **rpsmain** and **rpstext** (along with the **.ism** filename extension and, for Windows and UNIX, the **.is1** filename extensions). *xf*ODBC uses repository data definitions to create data definitions in system catalogs.

When the term *repository* (all lowercase) is used, it refers to the repository files (**rpsmain** and **rpstext**, or their equivalents in your database); the term *Repository* (capital "R") refers to the Synergy application you use to define your repository files.

For more information on using Repository, see the *Repository User's Guide*.

Connect file

The connect file is a text file you create to tell *xf*ODBC where to find your Synergy data files and the system catalog that describes those data files. The connect file can also be used to define environment variables used by the *xf*ODBC driver, set the convert_error option (which instructs the *xf*ODBC driver to treat invalid dates as null data), and set Synergy driver logging (which enables you to determine if a system catalog is cached). You must have a connect file to open the system catalog in DBA.

For more information, see chapter 5, "Setting Up a Connect File."

Conversion setup file

The conversion setup file is a text file that stores information about table locations and access levels. You will probably use DBA to create and modify this file automatically, but you can also perform these steps manually with a text editor. DBA and **dbcreate** can use the conversion setup file when regenerating a system catalog. For information, see "Generating and Editing a Conversion Setup File" on page 6-27.

Database Administrator program (DBA)

With its graphical user interface, the *xf*ODBC Database Administrator (DBA) program enables you to manage system catalogs. Use DBA to initialize user and group access to your database; create a conversion setup file; generate, maintain, customize, and verify your system catalogs; and run **dbcreate**. You can start DBA from the command line or from the Synergy Control Panel (in Windows Control Panel).

dbcreate utility

The **dbcreate** utility generates system catalogs using repository definitions in repository files (**rpsmain** and **rpstext** files, as they're commonly named). The repository definitions must contain all the structure, tag, field, and key information you need in the system catalog.

You can run this utility from the command line or from within DBA. For more information on generating a system catalog, see chapter 4, "Creating a System Catalog."

Runtime components

The runtime components enable you to access Synergy data from ODBC-enabled applications. Some of these components are distributed with *xf*ODBC. Others must be created for your Synergy data—namely the system catalog, the connect file, a DSN, and possibly an environment setup file.

Figure 1-3 on page 1-10 illustrates how *xf*ODBC runtime components work together to access Synergy data.

Synergy database

A Synergy database can consist of Synergy ISAM files, relative files, tagged files, or ASCII text files. We use the term *data files* to refer to the files that make up a Synergy database. See "Synergy database" on page 1-4.

System catalog

For ODBC-enabled applications to access a Synergy database, a *system catalog* is required. The system catalog is generated by **dbcreate** or DBA and provides a "translation" of the Synergy database in a form that the *xf*ODBC driver can understand. You might think of the system catalog as a kind of road map that contains "directions" for *xf*ODBC, providing table location, column, key, access, and other necessary information about the Synergy database.

The **dbcreate** utility and the DBA program enable you to generate a system catalog from repository files. The DBA program also enables you to view and modify your system catalog files.

A system catalog is composed of tables for database files and ten system tables (see table below). In Windows and UNIX environments, these tables are composed of two ISAM files with the **.ism** and **.is1** filename extensions. In OpenVMS, these are composed of one ISAM file with the **.ism** extension. Each system table contains specific information about the Synergy database, except the GENESIS_DUAL table, which is a read-only table with a synonym (dual) and with one row and one column that's used for statements that require a single row—for example, "SELECT curdate() FROM dual".

Without the user and group files (**sodbc_users.***, **sodbc_groups.***), there is no user or password validation when connecting to the database, and all connected users have read-only access to all tables in the database. Be sure to generate these files (see "Generating the System Catalog" on page 4-2) and keep them with the other system catalog files.

System Catalog Files

Table name	Filenames	Contents
GENESIS_COLUMNS	**GENESIS_COLUMNS.ISM** **GENESIS_COLUMNS.IS1**	Field size, type, and position information
GENESIS_DEPENDS	**GENESIS_DEPENDS.ISM** **GENESIS_DEPENDS.IS1**	SQL view dependency information and information about the names, owners, and database names for the views
GENESIS_DUAL	**GENESIS_DUAL.ISM** **GENESIS_DUAL.IS1**	A read-only table with one row and one column for operations such as "SELECT curdate() FROM dual"
GENESIS_FORKEYS	**GENESIS_FORKEYS.ISM** **GENESIS_FORKEYS.IS1**	Foreign key information
GENESIS_INDEXES	**GENESIS_INDEXES.ISM** **GENESIS_INDEXES.IS1**	Access keys
GENESIS_TABLES	**GENESIS_TABLES.ISM** **GENESIS_TABLES.IS1**	File, structure, access level, and tag information
GENESIS_VIEWS	**GENESIS_VIEWS.ISM** **GENESIS_VIEWS.IS1**	SQL view definitions, which include information such as view name and the query used to create the view
GENESIS_XCOLUMNS	**GENESIS_XCOLUMNS.ISM** **GENESIS_XCOLUMNS.IS1**	Column references for access key segments
Does not appear as a table in DBA	**SODBC_GROUPS.ISM** **SODBC_GROUPS.IS1**	Group ID, group name, number of users assigned to each group, group access level, and group description
Does not appear as a table in DBA	**SODBC_USERS.ISM** **SODBC_USERS.IS1**	User name, password, user's full name, and group ID

DSN

A data source name (DSN) is a text file that contains the information needed to access a database (the name of the connect file, user and password information, and so forth). Once you've created a DSN for a database, users can access the database from an ODBC-enabled application by selecting the DSN. DSNs make connection details invisible to end-users and free end-users from having to remember the location of the data files and other connection information. See "Setting Up Access with DSNs" on page 8-4 for information.

Environment setup file

The optional environment setup file is a text file you write to define the data environment variables that are used by *xf*ODBC when locating Synergy data files. The environment setup file is typically used to set environment variables that are used in the Open filename field of a repository file definition.

For more information on using an environment setup file, see "Setting environment variables in an environment setup file" on page 3-34.

*xf*ODBC driver

The *xf*ODBC driver is a DLL (**tod32.dll** or **tod64.dll**) called by the ODBC Driver Manager whenever a third-party, ODBC-enabled application accesses a Synergy database. The *xf*ODBC driver uses a connect file to locate Synergy data files and the system catalog. Using the system catalog as a road map, the driver then reads the data files and transfers data between the database and the third-party application.

Synergy database driver

They Synergy database driver (**vtx4**) enables the *xf*ODBC driver to access Synergy databases. The Synergy database driver directly processes SQL commands.

External components

The following are not part of *xf*ODBC, but they work with *xf*ODBC.

ODBC Driver Manager

A DLL provided by Microsoft that opens and closes ODBC drivers as directed by an ODBC-enabled application.

ODBC-enabled application

A 32-bit application running on Windows that uses the ODBC API to access databases. Crystal Reports, Microsoft Access, and Microsoft Query are examples. Synergy applications that use SQL Connection are also "ODBC-enabled."

SQL OpenNet

A Synergy product that enables *xf*ODBC to work in a client/server configuration. Figure 1-3 on page 1-10 illustrates how SQL OpenNet works with *xf*ODBC runtime components to access Synergy data.

For more information on SQL OpenNet, see the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.

The Synergy/DE Data Provider for .NET

An ADO.NET data provider that enables access from .NET applications and Visual Studio to Synergy databases. The Synergy/DE Data Provider for .NET includes all of the functionality of the .NET Framework data provider for ODBC (which it wraps), along with support for the Entity Framework. It enables you to create an entity data model (EDM) and then modify and query the EDM (and, by extension, the Synergy database) using LINQ to Entities and Entity SQL. The Synergy/DE Data Provider for .NET also includes a Visual Studio plug-in (a DDEX provider) that enables you to create Visual Studio data connections for Synergy databases.

For information on the Synergy/DE Data Provider for .NET, see "Accessing Synergy Data in a .NET Environment" on page 9-10.

*xf*ODBC requirements and installation

You'll need to install *xf*ODBC for both development and deployment of an ODBC-accessible Synergy database. General requirements are listed below. See the installation instructions and the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide* for more information.

For development, you'll need Connectivity Series on the development machine. You'll also need Synergy data files and a Synergy repository that defines them. See chapter 3, "Preliminary Steps," for more information.

For deployment of an ODBC-accessible Synergy database, you'll need the following, and you may need to set data access options. (See chapter 8, "Configuring Data Access.")

Component	Location in client/server configuration
Connect file	Server
System catalog	Server
DSN	Client
Synergy data files	Server
32-bit or 64-bit ODBC-enabled Windows-based application	Client

For a stand-alone deployment, you'll need to install Connectivity Series.

For a client/server deployment, you'll need the following (see figure 1-3 on page 1-10):

- Connectivity Series on the server
- Either Connectivity Series or the *xf*ODBC Client on the client
- A TCP/IP network set up between the client systems and the server system

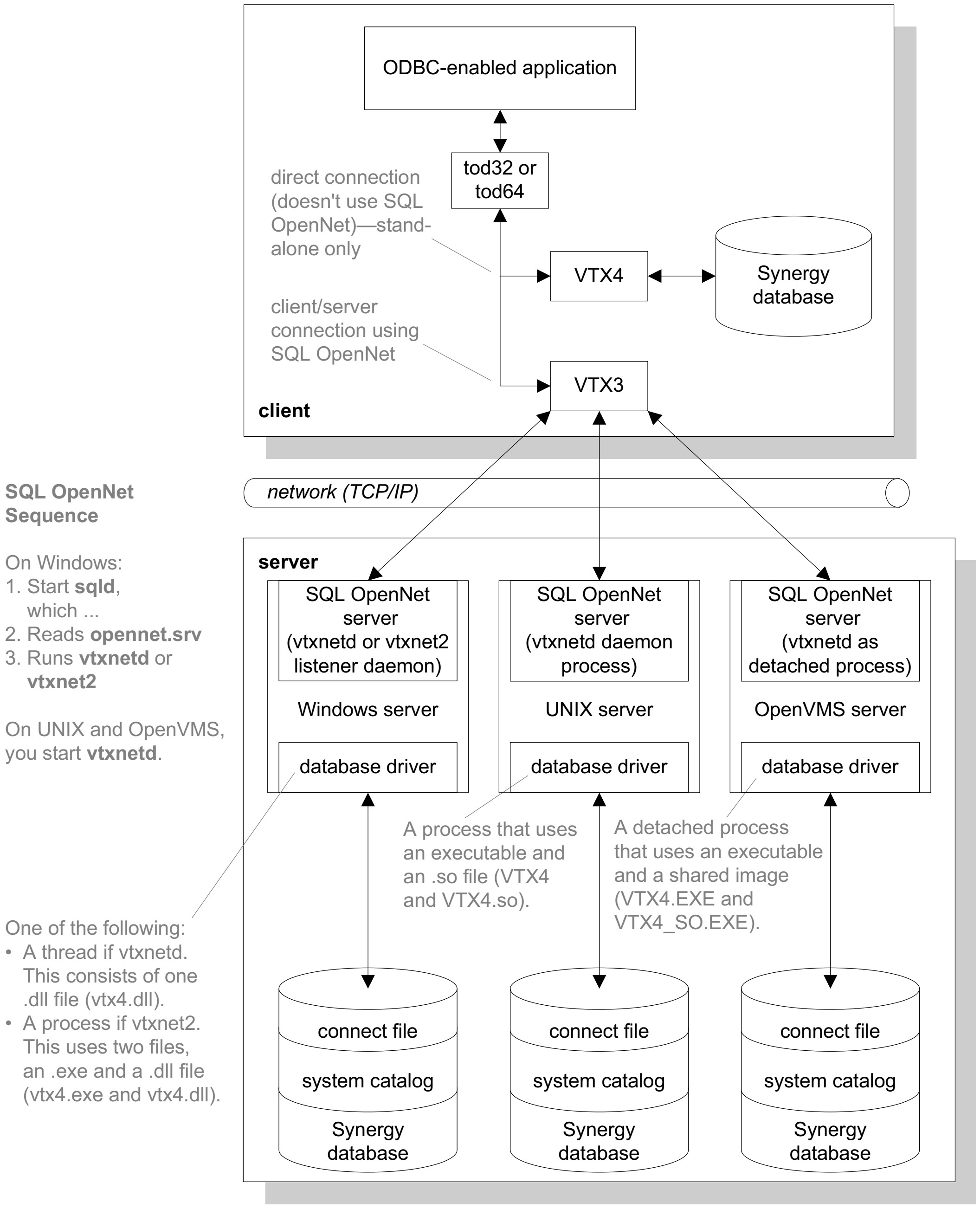
In a client/server configuration, *xf*ODBC uses SQL OpenNet for the network layer. The SQL OpenNet client must be installed on each client machine (it's installed with either Connectivity Series or the *xf*ODBC Client), and the SQL OpenNet server must be installed on the server (it's installed with Connectivity Series).

The Connectivity Series installation includes all *xf*ODBC components (development and deployment, client and server). The *xf*ODBC Client installation includes only those components that are needed for an *xf*ODBC client:

- Licensing information
- The *xf*ODBC driver, (**tod32.dll** or **tod64.dll**)
- The SQL OpenNet client, **vtx3.dll**
- **Dltest.exe**, which enables you to get a list of needed Connectivity Series DLLs (see "Error Logging" on page 11-2 for more information)
- The error message file, **sql.msg**, which contains the error messages *xf*ODBC and SQL OpenNet display when they encounter errors (see "Editing the SQL Message File" on page 11-10)
- The **net.ini** file, which enables you to set an encryption key, time-outs, and other network settings for SQL OpenNet (see "SQL OpenNet Client Options in net.ini" on page 8-26)
- The **vtxping** utility, which enables you to test SQL OpenNet connections (see "The vtxping Utility" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*)

For information on requirements for third-party software, see "Third-Party Software Requirements" on page 9-3.

For information on requirements for the Synergy/DE Data Provider for .NET, see "System requirements for the Synergy/DE Data Provider for .NET" on page 9-11.



SQL OpenNet Sequence

On Windows:
1. Start **sqld**, which ...
2. Reads **opennet.srv**
3. Runs **vtxnetd** or **vtxnet2**

On UNIX and OpenVMS, you start **vtxnetd**.

One of the following:
- A thread if vtxnetd. This consists of one .dll file (vtx4.dll).
- A process if vtxnet2. This uses two files, an .exe and a .dll file (vtx4.exe and vtx4.dll).

A process that uses an executable and an .so file (VTX4 and VTX4.so).

A detached process that uses an executable and a shared image (VTX4.EXE and VTX4_SO.EXE).

Figure 1-3. xfODBC and SQL OpenNet components accessing Synergy data.

How Third-Party Applications Use *xf*ODBC

When a user opens a 32-bit ODBC-enabled application, such as Crystal Reports, and accesses a Synergy database, the following components are called or read: ODBC Driver Manager, the *xf*ODBC driver, your connect file, the system catalog, and the Synergy database.

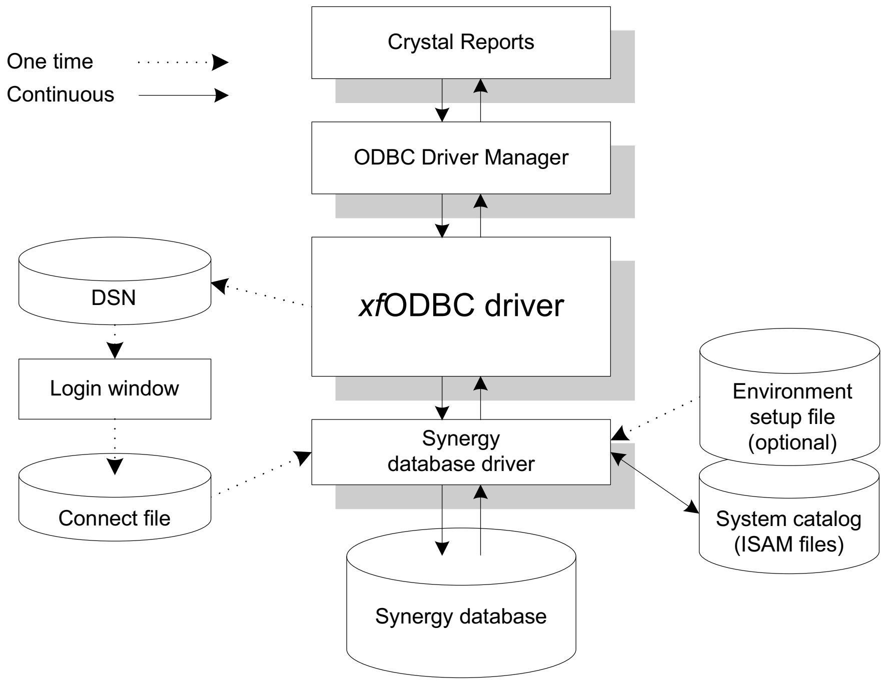


Figure 1-4. Crystal Reports accessing a Synergy database in a stand-alone configuration.

The *xf*ODBC process can be summarized by the following steps:

1. The ODBC-enabled application (Crystal Reports in figure 1-4) makes a request to the ODBC Driver Manager, which loads the *xf*ODBC driver and establishes an interface between the application and *xf*ODBC.
2. *xf*ODBC reads the data environment variables set in the environment setup file (optional).
3. *xf*ODBC reads the information in the DSN.
4. *xf*ODBC prompts the user for any information that's missing from the DSN.
5. *xf*ODBC reads the connect file and then locates the system catalog and data files.
6. *xf*ODBC reads the system catalog for a road map of the Synergy database and then verifies the user name and password against the registered users in the system catalog.

7. The ODBC-enabled application passes the SQL-based request to *xf*ODBC, which then passes it to the Synergy database driver.
8. The Synergy database driver retrieves the requested data and passes it on to *xf*ODBC, which “translates” it into a form recognized by the ODBC-enabled application.

The Steps to ODBC Access

To prepare Synergy data for ODBC access, you'll need to create a system catalog, a connect file, and a DSN. In addition, you'll need a Synergy repository that defines the data in the Synergy database, and you'll need to install *xf*ODBC.

A summary of the steps

The following is a summary of the steps you'll follow as you use *xf*ODBC to generate a system catalog, modify the system catalog, and then access your Synergy database. To use these steps with the sample database, see chapter 2, "Using the Sample Database As a Tutorial."

Installing and configuring *xf*ODBC

1. Install Synergy/DE Professional Series, including the Connectivity Series component, on your system.

Setting catalog generation options

2. Prepare your environment and set options for system catalog generation. See "Setting Options and File Locations" on page 3-18.

Generating a system catalog

3. Generate a system catalog from your repository definitions. Initialize users and groups. See "Generating the System Catalog" on page 4-2.

Creating a connect file

4. Create a connect file that specifies the current location of the Synergy database. See "Creating the Connect File" on page 5-2.

Customizing the system catalog

5. Use DBA to open your system catalog. See "Opening the System Catalog in DBA" on page 6-10.
6. Modify the users and groups you initialized earlier, and assign group access levels. See "Customizing Users and Groups" on page 6-13 and "Setting Security Levels" on page 8-2.
7. Make any necessary changes to the system catalog tables or columns and their attributes. See "Customizing Tables and Table Elements" on page 6-19.
8. If necessary, set table access levels in your conversion setup file and regenerate the system catalog, specifying the conversion setup file as input. See "Generating and Editing a Conversion Setup File" on page 6-27 and "Regenerating the System Catalog" on page 4-9.

Creating a DSN

9. Create a DSN to access your data. See "Setting Up Access with DSNs" on page 8-4.

Setting data-access options

10. Set options that affect the way *xf*ODBC accesses data. See "Setting Runtime Data Access Options" on page 8-13.

Accessing your Synergy database

For more information on these steps, see chapter 9, "Accessing a Synergy Database."

11. Open a third-party, ODBC-enabled application.
12. In the third-party application, choose the DSN for the Synergy database. (The third-party application calls the ODBC Driver Manager, which in turn calls *xf*ODBC.)
13. If necessary, at the log-in window enter your user name and password. You should now be able to access the Synergy database from the third-party application.

2

Using the Sample Database As a Tutorial

This tutorial guides you through the steps needed to prepare a Synergy database for access by an ODBC-enabled application. As you follow these steps, you will do the following:

- Set catalog generation options
- Generate a system catalog
- Create a connect file
- Customize the system catalog by adding users and groups, deleting a table, and changing a table's access level
- Regenerate the system catalog

You will do all of this with the sample database, so you don't have to practice on your own data. The sample database is included in the Connectivity Series distribution. (It is not included in the *xf*ODBC Client installation.) This database includes a repository and is stored in the connect\synodbc\dat and connect\synodbc\dict subdirectories of the main Synergy/DE installation directory.

Use this tutorial with a stand-alone *xf*ODBC configuration or on the server of a client/server configuration.

For Windows Vista and higher, you may need to change the GENESIS_HOME environment variable setting and move the sample database and repository to a writable location outside of Program Files so that files can be created and updated. If you do this, adjust the procedures in this tutorial to use the new location.

1. Install *xf*ODBC

The first step is to install *xf*ODBC. Follow the installation instructions, and refer to the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.

The remainder of this tutorial assumes that during installation you set the Synergy directory as /usr/synergyde in a UNIX environment or DKA600:[SYNERGYDE] on OpenVMS. If your Synergy directory has another name or location, you must substitute that name or location in the examples.

2. Set file locations and options

Once you've installed *xf*ODBC, you need to set some options and file locations. These settings prepare your environment for the next step in the process, generating a system catalog. To prepare for system catalog generation, set *xf*ODBC environment variables as follows. (For information on *how* to set environment variables, see "Setting environment variables" on page 3-30.)

- RPSMFIL — Set to the path and filename of the sample repository main file. For example:

  ```
  RPSMFIL=%CONNECTDIR%synodbc\dict\rpsmain.ism
  ```

  For more information, see "Specifying repository file locations" on page 3-22.

- RPSTFIL — Set to the path and filename of the sample repository text file. For example:

  ```
  RPSTFIL=%CONNECTDIR%synodbc\dict\rpstext.ism
  ```

  For more information, see "Specifying repository file locations" on page 3-22.

- SODBC_CNVFIL — Make sure this is *not* set. This should not be set until the conversion setup file has been created. For more information on this variable, see "Specifying a conversion setup file" on page 3-26.

You may want to set other environment variables that set system catalog generation options. For information, see "Setting Options and File Locations" on page 3-18.

3. Generate the system catalog from DBA

To make Synergy data accessible to ODBC-enabled applications, you must create a system catalog for the database. The system catalog is generated from repository definitions and provides the information the *xf*ODBC driver needs to access the data files.

You can generate the system catalog from the command line using the **dbcreate** utility, or you can generate it from the *xf*ODBC Database Administrator (DBA), a program that you can also use to modify the system catalog. For this tutorial, we'll generate the system catalog both ways. In this step, we'll use DBA to generate it; in the next step ("Generate the system catalog from the command line" on page 2-5), we'll use the **dbcreate** utility to generate it from the command line.

1. Open DBA by doing one of the following:

- In Windows Control Panel, select Synergy Control Panel, and then click "xfODBC DBA."
- Type the following at a Windows or UNIX prompt:

  ```
  dbr SODBC_DBA:xfdba.dbr
  ```

- Type the following at an OpenVMS prompt:

  ```
  $ RUN SODBC_DBA:XFDBA.EXE
  ```

2. From the Catalog menu in DBA, select Generate. (In UNIX and OpenVMS environments, press CTRL+P to activate the DBA menu. For more information, see "Understanding DBA, the Customization Program" on page 6-2.)

The following message is displayed:

No system catalog connected.

3. Click OK or press ENTER.

4. In the Generate System Catalog window, fill and set the following fields and options:

Main repository. This field sets the path and filename of the repository main file. (By default, this field contains the value of the RPSMFIL environment variable.) Make sure this field contains one of the following:

- On Windows:

  ```
  CONNECTDIR:synodbc\dict\rpsmain.ism
  ```

- On UNIX:

  ```
  CONNECTDIR:synodbc/dict/rpsmain.ism
  ```

- On OpenVMS:

  ```
  DKA600:[CONNECT.SYNODBC.DICT]RPSMAIN.ISM
  ```

Text repository. This field sets the path and filename of the repository text file. (By default, this field contains the value of the RPSTFIL environment variable.) Make sure this field contains one of the following:

- On Windows:

  `CONNECTDIR:synodbc\dict\rpstext.ism`

- On UNIX:

  `CONNECTDIR:synodbc/dict/rpstext.ism`

- On OpenVMS:

  `DKA600:[CONNECT.SYNODBC.DICT]RPSTEXT.ISM`

Dictsource path. This field specifies where the system catalog will be saved. Type one of the following paths:

- On Windows,

  `CONNECTDIR:synodbc\dict`

- On UNIX,

  `CONNECTDIR:synodbc/dict`

- On OpenVMS,

  `DKA600:[CONNECT.SYNODBC.DICT]`

Conversion setup. Clear this field.

Field report view. Clear this option to generate the system catalog from all repository fields. If this option is selected, the system catalog won't include fields for which the "No report view" option is set in the repository.

Update option. Leave the default option (Clear and re-create catalog) selected. This option ensures that the system catalog is generated from scratch. Before generating, DBA clears existing system catalog files from the directory you specify in the Dictsource path field.

Initialize users and groups. Select this option to ensure that the initial groups (SYSTEM and USER) and the initial users (DBADMIN, DBA, and PUBLIC) are created. These initial users and groups enable you to open the system catalog in DBA and customize the system catalog.

Overwrite existing. This option instructs DBA to overwrite existing users and groups. It is available only when the "Initialize users and groups" option is selected. It has no effect if you're generating a system catalog for the first time, but if you've generated a system catalog and made changes to the initial set of user and groups, these changes will be lost if you select this option.

5. Click OK or press ENTER.
6. When DBA is finished generating the system catalog, a message ("System catalog generated") is displayed. Click OK or press ENTER.

   For more information, see "Generating the System Catalog" on page 4-2.

4. Generate the system catalog from the command line

There are two ways to generate a system catalog: from DBA and from the command line. "Generate the system catalog from DBA" on page 2-3 uses the first method. In this section, you'll use the second method; you'll use the **dbcreate** utility to generate a system catalog from the command line.

Once you have completed the previous step ("Generate the system catalog from DBA"), you've generated the system catalog, so you can skip this step. If you want to try generating the system catalog from the command line, however, follow the procedure in this section.

To generate the system catalog from the command line, enter the following command:

```
dbcreate -c -p -r rpsmain rpstext
```

The **-c** option clears and regenerates the system catalog, the **-p** option creates a default set of users and groups, and the **-r** option specifies the location and name of the repository main file and repository text file. (If you don't include the **-r** option, **dbcreate** uses the RPSMFIL and RPSTFIL environment variable settings.)

You should now have several ISAM files that begin with **GENESIS_** or **SODBC_** in the current working directory. To generate the system catalog in any other directory, enter a command with the following syntax:

```
dbcreate -c -p -d target_directory -r rpsmain rpstext
```

where *target_directory* is the location the system catalog will be saved to. For more information, see "Generating the System Catalog" on page 4-2.

5. Create a connect file

The next step is to create a connect file. You'll need a connect file to customize the system catalog and to create a data source name (DSN), which is required for ODBC access. Connect files contain information on where the Synergy data files and the system catalog are located. Connect files can also contain environment variable settings and data access settings.

The *xf*ODBC distribution includes a sample connect file, **sodbc_sa**, located in the directory that the GENESIS_HOME environment variable is automatically set to (connect\synodbc). You can use this connect file to complete the tutorial, or you can create your own as an exercise.

Every connect file must have a dictsource line and a datasource line. The *dictsource* line specifies the directory where your system catalog will be located, and the *datasource* line specifies the directory where your Synergy data files reside. You can also set some environment variables in the connect file. If you use any environment variables in the Open filename field of Repository file definitions, this file is the best place to define those variables. It's better than defining them system wide (where they're not needed), and it keeps them all in one location.

For the sample database, the connect file must contain three lines: the dictsource line, the datasource line, and a line that sets the XFDBTUT environment variable.

- For Windows:

```
dictsource "C:\Program Files\Synergex\SynergyDE\connect\synodbc\dict\"
datasource ";C:\\Program Files\\Synergex\\SynergyDE\\connect\\syn-
odbc\\dat;"
XFDBTUT=C:\Program Files\Synergex\SynergyDE\connect\synodbc\dat
```

- For UNIX:

```
dictsource  /usr/synergyde/connect/synodbc/dict
datasource ;/usr/synergyde/connect/synodbc/dat;
XFDBTUT=/usr/synergyde/connect/synodbc/dat
```

- For OpenVMS:

```
dictsource  DKA600:[SYNERGYDE.CONNECT.SYNODBC.DICT]
datasource ;DKA600:[SYNERGYDE.CONNECT.SYNODBC.DAT];
XFDBTUT=DKA600:[SYNERGYDE.CONNECT.SYNODBC.DAT]
```

If you make your own connect file, create a text file with the above lines. Then, to make sure you created the connect file correctly, compare the file you create to the sample connect file, **sodbc_sa**.

For more information, see chapter 5, "Setting Up a Connect File."

6. Open the system catalog

Once you've generated a system catalog and created a connect file, you can view and customize the system catalog.

1. Open the DBA program if it isn't already open. (See step 1 on page 2-3 for instructions.)
2. From the Catalog menu, select Open.

   The Open System Catalog window opens.
3. Enter the connect file name, user name, and password name in the Open System Catalog window. Note that the user name and password are case sensitive.

   In the Connect file field, enter the name of the connect file. If you created your own connect file, enter its filename here. Otherwise, type

   ```
   sodbc_sa
   ```

   In the User name field, type

   ```
   DBADMIN
   ```

   In the Password field, type

   ```
   MANAGER
   ```

Alternatively, you can type an entire connect string in the Connect file field—for example, DBADMIN/MANAGER/sodbc_sa. (The syntax for a connect string is *username/password/connect_filename*.) If you do this, leave the User name and Password fields blank.

Note that you can also open the system catalog from the command line. To do this, close DBA; then do one of the following:

- Type the following at a Windows or UNIX prompt:

```
dbr SODBC_DBA:xfdba.dbr -c DBADMIN/MANAGER/sodbc_sa
```

- Type the following at an OpenVMS prompt:

```
$ XFDBA -C DBADMIN/MANAGER/SODBC_SA
```

For more information, see "Opening the System Catalog in DBA" on page 6-10.

If you get an error message that says "Login failed: unable to open user file", it may be that

- the users and groups were not initialized when you generated the system catalog.
- one of the entries may have been spelled incorrectly.
- the case of the user name or password was incorrect.

Try opening the system catalog again, double-checking the case and the spelling. If you still aren't able to open it, regenerate the system catalog by following the instructions in "Generate the system catalog from DBA" on page 2-3 or "Generate the system catalog from the command line" on page 2-5. Make sure you either select the Initialize users and groups option (in DBA) or set the **-p** option (for the command line). Once you've regenerated the system catalog, open the system catalog in DBA.

7. Modify the users and groups

Once you've opened the system catalog in DBA, you can view and customize users and groups; you can view tables, columns, indexes, and segments; and you can delete tables and columns.

1. From the Maintenance menu, select Groups. Then, from the Group Maintenance menu, select New Group.

   The Group window is displayed.

2. In the Group window, fill in the following fields:

   Group name. Enter a temporary group name, such as TEMPGRP.

   Access level. Enter an access level of 102. (You may want to test different access levels to see how they affect read/write access. For more information on access levels, see "Setting Security Levels" on page 8-2.)

   Description. Enter a description, such as "Temporary group", and then click OK or press ENTER. Notice the Group ID in the Group List window; it should be 3.

3. Close the Group List window.
4. From the Maintenance menu, select Users. Then, from the User Maintenance menu, select New User. The User window is displayed.
5. In the User window, fill in the following fields:

   User name. Enter a temporary user name, such as TempUser.

   Password. Enter a password for this user, such as DBAPSWD. Remember that the password and user name are case sensitive.

   Group ID. Assign the user to the new temporary group by entering 3 in this field.

   Full name. Enter the user's full name, or enter a description of the user, such as "Temporary user".
6. Click OK or press ENTER, and then close the User List window.
7. From the Maintenance menu, select Groups. Notice that the TEMPGRP now has one user assigned to it.
8. Close the Group List window.

8. Generate a conversion setup file

Conversion setup files are text files that contain information on tables in the data files. This information includes table names, table access levels, and data file locations, among other things. Using a conversion setup file, you can change the access level of a table or add a table back into the system catalog. Additionally, if you use DBA to delete a table from the system catalog, you can use a conversion setup file to preserve that change when you regenerate. See "Generating and Editing a Conversion Setup File" on page 6-27.

1. From the Catalog menu, select Generate Conversion Setup File.
2. Click OK or press ENTER to generate the conversion setup file to the displayed path and name.
3. If the file already exists, another window is displayed. Click OK or press ENTER to overwrite the existing file.
4. Set the SODBC_CNVFIL environment variable to the path and filename of the conversion setup file you just generated. (For more information, see "Specifying a conversion setup file" on page 3-26.) For example:

   ```
   SODBC_CNVFIL=GENESIS_HOME:sodbccnv.ini
   ```

   For more information on generating a conversion setup file, see "Generating and Editing a Conversion Setup File" on page 6-27.

9. Edit the conversion setup file

Once you've generated the conversion setup file, you can use a text editor to do the following:

- Mark a table as IN or OUT. This determines whether the table will be considered when the system catalog is regenerated. For information, see "IN | OUT" on page 6-29.
- Change a table's access level.
- Change a table's data file location.

1. Open the conversion setup file **sodbccnv.ini** with a text editor. Notice the following:
   - Each of the four data tables is set to IN.
   - Each data table has an access level of 100.
   - Each data table specifies the XFDBTUT environment variable for opening the data file.
2. Change the access level of the ORDERS table to 101 and the PLANTS table to 200. An access level of 101 allows read/write access for users with an access level of 101 or greater; an access level of 200 allows read-only access for users with an access level of 200 or greater. See "Setting Security Levels" on page 8-2.
3. Save your changes and close the file.

   For more information, see "Editing the conversion setup file" on page 6-29.

10. Remove a table from the system catalog

You can use DBA to delete a table from the system catalog, but the table will be added back if you regenerate the system catalog. To remove a table and *keep* it out—even if you regenerate—use a conversion setup file.

1. Make sure the SODBC_CNVFIL environment variable is set to the location of your conversion setup file.
2. In DBA, open the system catalog for the sample database.
3. From the Maintenance menu, select Tables.
4. Highlight the VENDORS table and select Delete Table from the Table Maintenance menu.
   A prompt will be displayed.
5. Click OK or press ENTER to delete the table.
6. Open the conversion setup file in a text editor. Note that the VENDOR table is set to OUT, so it is *not* available to an ODBC-enabled application. Before you deleted the VENDORS table, this was set to IN, but because the SODBC_CNVFIL environment variable is set, tables deleted from the DBA are automatically set to OUT in the conversion setup file. (If SODBC_CNVFIL is *not* set when you delete a table in DBA, the conversion setup file will *not* be automatically updated.) Note that you can also change a table's IN|OUT setting by editing the conversion setup file.

   For more information, see "Deleting a table" on page 6-21.

11. Change a table's access level

To change a table's access level, you must edit the conversion setup file and then regenerate the system catalog, using the conversion setup file as input.

1. Using a text editor open the conversion setup file.
2. On the CUSTOMERS line, change ACC=100 to ACC=101. This changes the CUSTOMERS table from a read-only table to a table that can be viewed and changed by users that belong to groups with access levels of 101 or greater. For more information, see "Setting Security Levels" on page 8-2.
3. Save the conversion setup file.

   To complete this change, you must regenerate the system catalog, using the conversion setup file as input. As the system catalog is regenerated, DBA reads the conversion setup file and makes any changes specified in this file to the system catalog.

12. Regenerate the system catalog

After making changes to the conversion setup file, regenerate the system catalog. You can regenerate the system catalog from the command line or from DBA. If you generate from the command line, use the **-i** option and specify the conversion setup file's path and filename after the option. (See "Using a conversion setup file" on page 4-10.) If you use DBA to regenerate, make sure the Conversion setup field of the Generate System Catalog window contains the conversion setup file name. For information, see "Regenerating the System Catalog" on page 4-9.

13. Access your data with an ODBC-enabled application

Now that you've set file locations and options, generated a system catalog, and created a connect file, you're almost ready to access the sample database with odbc-enabled applications. There is, however, one more requirement: in most cases, you must have a DSN (data source name). DSNs contain the information needed to access a database. For example, a DSN may contain the name of the connect file as well as user and password information. You can create a DSN for the sample database or you can use the system DSN (named xfODBC) included with the Connectivity Series installation. For information on DSNs, see "Setting Up Access with DSNs" on page 8-4. Note the following:

- It is possible to connect via ADO or ADO.NET without a DSN. See "DSN-less connections" on page 9-9 and "Connection strings" on page 9-18.
- See "Examples" on page 9-33 for examples that take you through the steps of accessing data.

Part 2: Preparing for ODBC Access

This section is for developers. It explains how to use *xf*ODBC and its utilities to generate, modify, and verify system catalogs as well as how to create connect files and DSNs.

3

Preliminary Steps

Before you generate a system catalog, you must define the database schema in a repository and set any needed environment variables for file locations, data conversion options, and other system catalog generation options.

Setting Up a Repository

To generate a system catalog for a Synergy database, *xf*ODBC must have access to a repository, which is a set of files that define the schema of the database. Repositories are created with Synergy/DE Repository and contain definitions for structures, fields, keys, relations, files, and tags.

You can use the following procedure to set up your repository, or you can enlist the help of Synergex Professional Services to optimize an existing repository for ODBC access or to create a new repository from your data files. For information, contact your Synergy/DE account manager.

The following steps outline the process for creating a repository for *xf*ODBC. (For information on setting up a repository to use multiple databases, see "Handling a repository shared by multiple databases" on page 3-7.) When you're finished, your repository should contain the following:

- A complete set of structures, tags, fields, and file information
- Templates for similar fields
- Well-chosen keys and relations

Before you define your repository

Plan and define your keys when you create your data files. Then, when you create the repository, create an access key for each of the data file keys. For more information on keys, see "Optimizing with Keys" on page 10-2.

1. Gather record layout information

Start by gathering information about the database. You'll need all the record layout, key, and tag information for the data files. Sources of information include

- *definition files* (also called include files). These files contain data definitions and can be added to your program with .INCLUDE statements. For information on definition files, see .INCLUDE in the "Preprocessor and Compiler Directives" chapter of the *Synergy DBL Language Reference Manual*.
- *FDL or XDL files* used to create the data files. If your data files were created with FDL or XDL files, you can print these files and use the information in them to manually define structures, fields, tags, and keys in your repository. For information on XDL and FDL, see "ISAM Definition Language" in the "Synergy DBMS" chapter of *Synergy Tools*. On OpenVMS, you can use the following to generate an FDL file:

  ```
  ANALYZE/RMS/FDL filename
  ```

- *parameter files*. You can use the **ipar** utility to generate parameter files, which are files that list record layout and key information for ISAM files. Although these files can be used as input for the **bldism** utility, they're also useful when creating a repository; they list data file information in a readable format. You can use this information to manually define structures in Repository. For information, see "ipar" in the "Synergy DBMS" chapter of *Synergy Tools*.

As you gather record layout information,

- list the primary key and all secondary keys for each record layout. Keep track of the key order. See step 8 for information on keys.
- list all tags for the records. See step 7 for information on tags.
- identify template candidates. See step 4 for information on templates.

2. Create a new repository

To create a new repository, open Synergy/DE Repository and select Create New Repository from the Utilities menu. For more information, see "Creating a New Repository" in the "Utility Functions" chapter of the *Repository User's Guide*.

TIP A single repository can be used to define multiple databases if the files, structures, and fields in the databases are identical. For information, see "Handling a repository shared by multiple databases" on page 3-7.

3. Create structures

Structures are record definitions or compilations of field and key characteristics for a particular file or files. Create a structure for each record layout in your data files. If you have a file with multiple record layouts, you must create a structure for each. See the "Working with Structures" chapter of the *Repository User's Guide*.

4. Create templates

A template is a set of characteristics that can be applied to multiple fields. Templates simplify maintenance and enable you to maintain consistency. For example, if several fields share some characteristics including Alternate name, you can create a template for the fields. Then, when you change the Alternate name for the template, the Alternate name is automatically updated for all the template's fields. For more information, see "Defining Field Templates" in the "Working with Fields" chapter of the *Repository User's Guide*.

If a template specifies an Alternate name, that template can be applied to only one field per structure.

5. Define fields and import field definitions

Use record layout information from step 1 to define fields for each structure. Note the following:

- If you have include files, you can use these to load field definitions directly into your repository. See "Loading Fields from a Definition File" in the "Working with Fields" chapter of the *Repository User's Guide*.
- Assign the templates you created in step 4 to the fields they were created for.

- If you want a field to have a different name in the system catalog then it has in the repository, use the Alternate name option. You can specify an Alternate name in a template or in the field itself. This option enables you to normalize names and to replace cryptic names with names your customers can understand (which we strongly recommend). For example, if a field that contains a customer number has the name "cf_1" in the repository, you can use Alternate name to give it the name "Customer_number" in the system catalog. When users access the database with an ODBC-enabled application, they will see "Customer_number" as the column name.

  If you specify an Alternate name, it will be used only if SODBC_ODBCNAME is set when the system catalog is generated. For information on the Alternate name field, see "Display information" in the "Working with Fields" chapter of the *Repository User's Guide*. For information on SODBC_ODBCNAME, see "Renaming columns for clarity" on page 3-24.
- To prevent *xf*ODBC from accessing a field, set the Excluded by ReportWriter option for that field. This causes the field to be omitted from the system catalog. (For example, you can use this option to omit overlay fields, which present alternative views of a database.) For information on setting the Excluded by ReportWriter option, see "Basic field information" in the "Working with Fields" chapter of the *Repository User's Guide*. (For information on an option that causes *xf*ODBC to ignore Excluded by ReportWriter settings, see "Including and omitting fields" on page 3-23.)
- If you create an overlay field, set it to read-only. (Set the Read-only option in Repository or use the READONLY repository schema keyword.) If a field is *not* in an INSERT statement's column list, the INSERT statement sets the field to null unless it is read-only. If the field is an overlay field, the overlay *and* the fields that make up the overlay are set to null (even if the overlay is not in the statement's column list). To prevent this, set overlay fields to read-only in the repository (and then regenerate the system catalog).
- If you use a format string to define a decimal point for a decimal field in Repository, you can either set SODBC_USEFORMAT or do the following. (See "Using decimal information in the repository format string" on page 3-27 for information on SODBC_USEFORMAT.)

  1. Create an implied decimal overlay field that matches the precision of the format string. For example, if you have a **d6** field with a format string that specifies the XXXX.XX format, create an overlay field with a **d6.2** data type.
  2. Use the Repository Alternate name option to specify an Alternate name that's identical to the original field. For example, if the field is named in_price, set the Alternate name for the overlay field to in_price.
  3. Set the Repository Excluded by ReportWriter option for the original field (not the overlay field). This will prevent the original field from becoming part of the system catalog, unless the SODBC_CNVOPT environment variable is set. For information on this variable, see "Including and omitting fields" on page 3-23.

If the field is a key segment, overlaying it with multiple overlay fields will prevent **dbcreate** from creating an index for the overlay fields. See "Create keys" on page 3-5 for information.

6. Change .INCLUDE statements in code

If you loaded data definitions from an include file into your repository, you will want to change the .INCLUDE statements in your code to include the repository definitions rather than the files. For information, see .INCLUDE in the "Preprocessor and Compiler Directives" chapter of the *Synergy DBL Language Reference Manual*.

7. Create tags to describe multiple structures in a file

If some of your structures are related, you may have them grouped in one file. (You'll assign structures to files in step 10 on page 3-6.) If you have more than one structure in a file, create a tag for each structure, a tag that uniquely identifies the structure in the file, and make sure you construct the tags so they can be optimized. See "Tags and optimization" on page 10-8 for information.

For information on how to define a tag, see "Defining Tags" in the "Working with Structures" chapter of the *Repository User's Guide*.

8. Create keys

Keys are portions of a record structure that individually identify records and enable records to be quickly accessed and sorted. Your data files may already have keys. However, for *xf*ODBC to use keys, they must be defined in the repository before the system catalog is generated. You can define two types of keys in a repository: *access keys*, which mirror keys defined in the database, and *foreign keys*, which are keys defined only in the repository. Foreign keys enable you to define additional relations that can't be defined with access keys.

Keep in mind that the keys you choose will greatly affect the performance of SQL queries, so choose them carefully.

Note the following:

- Foreign keys are used only if an ODBC-enabled application supports the ODBC API function SQLForeignKeys.
- The *xf*ODBC driver considers the first unique access key to be the primary key. Note that the ADO.NET Entity Framework must have a primary key, so if you plan to use the Entity Framework, make sure each file has a unique key.

See "Optimizing with Keys" on page 10-2 for more information on what keys are, how *xf*ODBC uses keys, and how to define keys for optimal performance.

For information on preventing *xf*ODBC from including keys, see "Omitting keys" on page 3-24.

9. Define files in Repository

File definitions are the Repository mechanism for storing information (name, file type, etc.) for data files associated with a repository. Create file definitions for all data files that are described by a repository's structures. See the "Working with Files" chapter of the *Repository User's Guide*.

Note that for file definitions, you must specify the filename in the Open filename field (which is in the Repository's File Definition window). However, you can specify the location of the data file in either the Open filename field or the datasource line of the connect file. (You cannot specify a filename in the datasource line.) The Open filename field has precedence; *xf*ODBC uses the datasource line only if the Open filename field does not specify a path or environment variable. For example, if a data file for the repository is **c:\datafiles\mydata.ism**, and you enter only **mydata.ism** in the Open filename field, you must have the following datasource line in the connect file:

```
datasource ;c:\\datafiles;
```

Note the following:

- For client/server configurations, the path must be local to the *server*.
- If you use an environment variable for the path or the path and filename, be sure to set the environment variable in the connect file, in an environment setup file, or in the environment (not in **synergy.ini**). In a client/server configuration, set it on the server.

For information on the datasource line, see "Creating the Connect File" on page 5-2.

10. Assign structures to files

Once you've created file definitions for the data files described by your repository, assign structures to those files. Every structure that you want included as a table in the system catalog must be assigned to a file definition. If a file contains multiple record layouts, assign all corresponding structures to that file.

For more information, see "Assigning Structures to Files" in the "Working with Files" chapter of the *Repository User's Guide*.

11. Define relations between structures

Relations enable you to link the keys for one structure to the keys for other structures. For example, if you create a relation between a customer ID key for a transaction structure with a customer ID key for a customer structure, you can create SQL statements that retrieve transaction information and associated customer information. (For an example of a relation to a table with a literal tag, see "Keys with literals" on page 10-6.)

When the system catalog is generated, structure relations are imported as table relations. Note that most ODBC-enabled applications aren't able to interpret table relations.

12. Validate, verify, and compare

When you have finished defining your repository, do the following in order:

- Validate your repository with the Validate Repository utility. This utility validates all values specified for repository definitions (field options, structure options, template options, and so forth). See "Validating Your Repository" in the "Utility Functions" chapter of the *Repository User's Guide* for more information.
- Verify your repository with the Verify Repository utility. This utility verifies the integrity (internal consistency) of your repository. The Verify Repository utility attempts to repair any problem it discovers. See "Verifying Your Repository" in the "Utility Functions" chapter of the *Repository User's Guide* for more information.
- Use the Compare Repository to Files (**fcompare**) utility to compare the repository definitions to your Synergy database files. This is available from the Utilities menu in S/DE Repository and from the command line. See "Comparing a Repository to ISAM Files" in the "Utility Functions" chapter of the *Repository User's Guide* and fcompare in the "Synergy DBMS" chapter of *Synergy Tools*.

Be sure to run these utilities and fix all errors and warnings *before* you generate the system catalog. Once the system catalog is generated, there is no connection between the system catalog and the repository. If you make changes to the repository after you generate the system catalog, the changes will *not* be reflected in the system catalog unless you regenerate it.

Handling a repository shared by multiple databases

A single repository can be used to define multiple databases if the file, structure, and field definitions for the databases are identical. The names of data files, however, do not need to be the same for each database. You could, for example, create data files named companyA for one database and companyB for another database. However, if the databases have the same names for the data files, each database's files must be in a separate location.

There are four common ways to handle a repository shared by multiple databases:

- *Specifying a filename (but no path) in the Open filename field* (see page 3-8). If generating a single system catalog suits your purpose, this method is generally the most convenient. You generate a single system catalog for all the databases, use the datasource line in each connect file to set the path for the data files, and use a separate connect file for each database.
- *Using an environment variable in the Open filename field* (see page 3-9). For this method, you generate a single system catalog for all the databases, but instead of using the datasource line in the connect files, you set an environment variable to determine which database is accessed.

- *Using a conversion setup file to change the Open filename field* (see page 3-9). If you want to create a separate system catalog for each database, this is generally the most convenient method. You enter whatever you want in the Open filename file and then use a conversion setup file to edit the data file locations before generating a system catalog. With this method, you also use a separate connect file for each database.
- *Using USR_DD_FILNAM to change replaceable characters* (see page 3-10). If you use the RPS_FILNAM_METHOD in Synergy/DE ReportWriter to interpret the Open filename field in your repository, this method is probably the best. You create a routine that interprets replaceable characters in the Open filename field. You rebuild the DBA program to include the new routine, and then use a conversion setup file to invoke the routine as you generate a system catalog for each database.

For information on the Open filename field, see "Defining a New File" in the "Working with Files" chapter of the *Repository User's Guide*.

The procedures in the following sections are overviews. For more information on individual steps, see the corresponding documentation elsewhere in this manual.

Specifying a filename (but no path) in the Open filename field

With this method, you'll generate a single system catalog that will be used for all the databases defined by the repository. Then, when you use an ODBC-enabled application to access one of the databases, the datasource line in the database's connect file will determine which database to access.

1. For each file in the repository, enter only a filename in the Open filename field. Do *not* include a path. For example:

```
customer.ism
```

2. Generate the system catalog. The system catalog will include data file specifications, but these will include only the filenames, no paths.
3. Create a connect file and DSN for each set of data files. In each connect file, set the datasource line to the directory that contains the data files for the database. For example:

```
datasource   ;c:\\databases\\company4\\dat;
```

4. Test by accessing the databases. Make sure each DSN uses the correct connect file and accesses the correct database.

Using an environment variable in the Open filename field

With this method, you'll generate a single system catalog that will be used for all the databases defined by the repository. Then, when you select a DSN in an ODBC-enabled application, an environment variable setting will determine which database is accessed.

1. Enter an environment variable in the Open filename field for each file in the repository. The environment variable can take the place of the path, the filename, or both. For example:

   ```
   COMPANY:plants.ism
   ```

   Make sure the environment variable is followed by a colon—even if the environment variable is for both the path and filename (which is a good way to handle data files whose names *and* locations are different for each database). For example:

   ```
   ACME_PLANTS:
   ```

2. Generate the system catalog. The environment variable will take the place of hard-coded paths in the system catalog.
3. Create a connect file and DSN for each database that the repository describes.
4. Before accessing the database, set the environment variables you entered in the Open filename field.

   Note that a good place to set this environment variable is in the connect files. This way the variable is redefined each time you use one of the connect files to access a database. For example, one of the connect files might have the following:

   ```
   COMPANY=c:\databases\companyabc\dat
   ```

   The other connect files would have different settings. For example:

   ```
   COMPANY=c:\databases\companyxyz\dat
   ```

5. Test by accessing the databases. Make sure each DSN uses the correct connect file and accesses the correct database.

Using a conversion setup file to change the Open filename field

With this method, you'll generate a conversion setup file and then use this file to generate a different system catalog for each database.

1. In the Open filename field for each file in the repository, enter a value. Later in this procedure (step 5), you'll replace all or part of this value.

   For example:

   ```
   &&&&&&customer.ism
   ```

2. Generate a system catalog for a database. Be sure to initialize users and groups.

3. Create a connect file for each database that the repository describes. Set the dictsource line in each connect file to the directory that will contain the system catalog files.
4. Create a conversion setup file for the system catalog.
5. Open the conversion setup file in a text editor and edit the data file settings. For example, if a data file setting is

   ```
   &&&&&&customer.ism
   ```

   you could change the setting to something like

   ```
   c:\companyabc\customer.ism
   ```

6. Regenerate the system catalog. If you use DBA to regenerate, be sure to use the Clear and re-create catalog, Initialize users and groups, and Conversion setup options. If you use **dbcreate**, be sure to use the **-c**, **-p**, and **-i** options. Use the conversion setup file you edited in step 5.
7. Open the system catalog in DBA. Then open the Table list (Maintenance > Tables). The tables should have the correct path for the database. (See the Open filename column of the Table List.)
8. Repeat step 5 through step 7 for each database.
9. Test by accessing the databases. Make sure each DSN uses the correct connect file, and make sure the dictsource line in each connect file accesses the correct system catalog.

Using USR_DD_FILNAM to change replaceable characters

With this method, you'll create a routine that interprets replaceable characters in the Open filename field. Then you'll rebuild DBA, create a conversion setup file, and use the conversion setup file to invoke the USR_DD_FILNAM routine as you generate a system catalog for each database.

This method is particularly useful if your repository already has replaceable characters used by the RPS_FILNAM_METHOD in ReportWriter. For information on RPS_FILNAM_METHOD, see "Modifying Filenames at Runtime" in the "Customizing ReportWriter Routines" chapter of the *ReportWriter User's Guide*.

1. In the Open filename field for each file in the repository, enter a value with replaceable characters. (If your repository was designed to use RPS_FILNAM_METHOD, skip this step; the Open filename field will already have replaceable characters.)

   For example:

   ```
   c:\####\customer.ism
   ```

2. Create a routine to interpret the replaceable characters in the Open filename field, and then rebuild DBA. For information, see "Replacing the default USR_DD_FILNAM routine" on page 3-11. This routine will replace the default USR_DD_FILNAM routine.
3. Generate a system catalog.
4. Create a connect file for the system catalog.

5. Open the system catalog in DBA, and generate a conversion setup file. As the conversion setup file is generated, replaceable characters in the Open filename field are interpreted by the USR_DD_FILNAM routine you wrote. Check the generated conversion setup file to make sure the paths are correct.
6. Regenerate the system catalog using the conversion setup file as input.

Replacing the default USR_DD_FILNAM routine

1. Write your USR_DD_FILNAM routine and save it as a **.dbl** file (for example, **my_usr.dbl**). See **xfdbusr.dbl**, a file included in your Connectivity Series distribution, for an example of a customized USR_DD_FILNAM routine.
2. Make sure
   - **PATH** contains the location of **dbcreate**.
   - **DBLDIR** is set in **synergy.ini** to your Synergy root directory.
   - **WND** is set in **synergy.ini** to your UI Toolkit directory.
3. Move to the directory that contains the DBA program, which is **xfdba.dbr** (on Windows and UNIX) or **xfdba** (on OpenVMS). Typically this file is in the connect\synodbc\dba directory.
4. From the command line, compile the file you created in step 1. For example:

   ```
   dbl my_usr
   ```
5. Replace the USR_DD_FILNAM routine in the DBA program's object library. For example:

   ```
   dblibr -r xfdbalib.olb my_usr.dbo
   ```
6. Do *one* of the following:
   - Rebuild DBA by entering one of the following.

     On UNIX or Windows:

     ```
     xfdbabld
     ```

     On OpenVMS:

     ```
     $ @XFDBABLD
     ```

     This command executes a batch file, script, or DCL command file.
   - Use the **dblink** command at a command prompt. For example:

     ```
     dblink -o xfdba.dbr xfdbaprc.dbo xfdbalib.olb WND:tklib.elb
     ```

   When you use DBA to generate the conversion setup file and the system catalog, DBA will now use the USR_DD_FILNAM you wrote to convert open filenames in the conversion setup file.

System Catalog Generation Issues

Keep the following in mind as you prepare to create a system catalog from your repository.

Access levels

When generated, all tables have an initial read-only access level set at 100. For information on access levels, see "Setting Security Levels" on page 8-2.

Arrays

Because the current SQL API does not support arrays, each element in an array field is mapped to a separate column and given a name that consists of the array name, the element's position in the array, and pound signs (#) to delineate position values. For example, a [2,2] array with the name myarray will be mapped as the following columns: myarray#1#1, myarray#1#2, myarray#2#1, and myarray#2#2. These are the names you use to access data in myarray—for example:

```
SELECT myarray#2#2 FROM mytable WHERE myarray#1#1 = 100
```

This is also true of groups and struct fields that are arrays, except that for these, **dbcreate** also generates a read-only overlay field that includes all of the fields in the array. For example, if a repository has a group or struct field named myarray that's a [2,2] array with a single field, myfield, the group or struct field will be mapped to the following columns: myarray (the overlay field), myarray#1#1myfield, myarray#1#2myfield, myarray#2#1myfield, and myarray#2#2myfield.

Note the following:

- To use a character other than the pound sign (#) to delineate position values, use the SODBC_TOKEN environment variable. See "Changing the position delimiter used for arrays" on page 3-25 for information.
- For array fields (but not groups or struct arrays), you can instruct **dbcreate** to generate a single overlay column for all elements. See "Generating one column for an array field" on page 3-25.

AutoSeq and AutoTime fields

AutoSeq and AutoTime fields are 8-byte read-only ISAM key fields that are automatically populated with appropriate values by the ISAM layer. AutoSeq is a generated number that is guaranteed to be unique within an ISAM file, and AutoTime is a timestamp that records the last date and time that a record was inserted or modified. See "Keys in ISAM files" in the "Synergy DBMS" chapter of *Synergy Tools* for more information.

Data types

The **dbcreate** utility generates system catalog columns with the following SQL types:

Data Types		
Repository data type		**SQL type**
Alpha		SQL_VARCHAR
AutoSeq		SQL_BIGINT
AutoTime		SQL_TIMESTAMP
Binary		SQL_BINARY
Boolean		SQL_BIT
Date		SQL_TYPE_DATE
Decimal	**d1** and **d2**	SQL_TINYINT
	d3 and **d4**	SQL_SMALLINT
	d5 through **d9**	SQL_INTEGER
	d10 and higher (except **d16.6**)	SQL_DECIMAL
	d16.6	SQL_FLOAT
Enum		SQL_INTEGER
Integer	**i1**	SQL_TINYINT
	i2	SQL_SMALLINT
	i4	SQL_INTEGER
	i8	SQL_BIGINT
Time (HHMM or HHMMSS)		SQL_TYPE_TIME

Data Types (Continued)		
Repository data type		**SQL type**
User	Alpha	SQL_VARCHAR
	Binary	SQL_BINARY
	Date with YYYYMMDDHHMISS or YYYYMMDDHHMISSUUUUUU format in the User data field	SQL_TIMESTAMP
	Date with HHMM or HHMMSS	SQL_TYPE_TIME
	Date with any other value in the User data field	SQL_TYPE_DATE
	Numeric	SQL_DECIMAL

Note the following:

- For an enum field, **dbcreate** generates an integer system catalog column that provides access only to the value side of the enumeration. For example, if you have an enumeration with three members—"tree" with a value of 1, "shrub" with a value of 2, and "groundcover" with a value of 3—a query that includes this column will return only 1, 2, or 3. (It will not return the member names "tree," "shrub," and "groundcover.") Note that you can use the DECODE scalar function to simulate an enumeration—for example:

  ```
  DECODE(plant_type, 1, 'tree', 2, 'shrub', 3, 'groundcover')
  ```

- Time columns are returned as System.TimeSpan for ADO.NET. See "Time columns and ADO.NET" on page 9-20.
- Fields with the struct data type are treated as groups. See "Groups and struct fields" on page 3-16.
- Data in user fields can be manipulated by the *xf*ODBC routines for user-defined data types. See chapter 7, "Creating Routines for User-Defined Data Types."

For more information on data types in Repository, see "Basic field information" in the "Working with Fields" chapter of the *Repository User's Guide*.

Date and time fields

When a system catalog is generated, date and time columns are generated as described below. For information on how date and time data is returned from a database and how dates and times must be specified in SQL statements, see “Setting Runtime Data Access Options” on page 8-13.

SQL time columns are generated from repository fields with the Time type or with one of the following formats specified in the Repository “User data” field (e.g., ^CLASS^=HHMM), where HH is the hour (in 24-hour format), MM is the minutes, and SS is the seconds:

HHMM
HHMMSS

SQL date columns are generated from AutoTime fields and repository fields with one of the following formats specified in the Repository Class field or User data field:

1. YYMMDD
2. YYYYMMDD
3. YYJJJ
4. YYPP
5. YYYYPP
6. YYYYJJJ
7. DDMMYY
8. DDMMYYYY
9. MMDDYYYY
10. MMDDYY
11. DDMonYY
12. DDMonYYYY
13. MonDDYY
14. MonDDYYYY
15. YYMonDD
16. YYYYMonDD
17. JJJYY
18. JJJYYYY
19. JJJJJ
20. PPYY
21. PPYYYY
22. YYYYMMDDHHMISS
23. YYYYMMDDHHMISSUUUUUU

where YY is the last two digits of the year, YYYY is the year and century, MM is the month, Mon is the three-letter month abbreviation (Jan, Feb, etc.), DD is the day of the month, HH is the hour, MI is the minute, PP is the period, SS is the second, UUUUUU is the microsecond, JJJ is the Julian day count from the first of the year, and JJJJJ is the Julian day count from SYNBASEDATE or the default base date, which is 1752-09-14 (i.e., 14 September 1752). (For information on SYNBASEDATE, see “Setting the base date for Julian day conversions” on page 8-15.) Note the following:

- Date formats 1 through 6 can be specified in Repository by selecting the corresponding date format in the Class field. And date formats 7 through 23 can be selected by defining the field as “User” in the Type field in Repository and by including the following in the “User data” field (where *date_format* is one of the above formats):

  `^CLASS^=`*date_format*

  See “Basic field information” in the “Working with Fields” chapter of the *Repository User’s Guide* for more information.

- When generating a system catalog, each date field that doesn’t include a century (a YY date) is formatted as a date with a rolling century (an RR date). This enables the *xf*ODBC driver to display the date correctly. See “Converting dates returned without centuries” on page 8-14 for information on how *xf*ODBC converts RR dates as it accesses a database.

Groups and struct fields

By default, if a field is part of a group or struct field in the repository, the group name or the struct name is added to the beginning of the field name to create the name for the column in the system catalog. For instance, the field myfield in the group mygroup becomes mygroupmyfield in the system catalog, and a field named myfield that's part of a struct field named mystruct becomes mystructmyfield in the system catalog. These are the names you can use to access data in the group or struct field—for example:

```
SELECT mystructmyfield FROM mytable
```

Note the following:

- If the repository specifies a member prefix for the group, the member prefix is used instead of the group name. (Member prefixes do no apply to struct fields.)
- To omit group and struct field names from column names, use the SODBC_NOGROUPNAME environment variable. See "Removing group and struct names from column names" on page 3-25.
- If the group or struct field is an array, each element is mapped to a separate column and all array fields are included in a read-only overlay column as described in "Arrays" on page 3-12.

Open filename field (S/DE Repository)

For greater flexibility, use environment variables in the Open filename field of Repository file definitions to specify the location of your data files. These environment variables are stored in the system catalog and must also be set in the system in which the database is installed, generally in the environment setup file or the connect file. For information, see "Using an environment variable in the Open filename field" on page 3-9.

You can also use the USR_DD_FILNAM routine in the *xf*ODBC Database Administrator (DBA) program to customize the Open filename field when you generate a system catalog. For more information, see "Using USR_DD_FILNAM to change replaceable characters" on page 3-10.

Overlay fields

*xf*ODBC supports overlay fields.

Relations

Relations established between tables, as defined in Repository, are supported in *xf*ODBC. Use of relations is application dependent.

Structures and table names

The **dbcreate** utility generates a caution if you attempt to generate a system catalog from a structure assigned to more than one file definition unless you use the ODBC table name option in Repository. If you use this option to assign an ODBC table name to file/structure combinations, **dbcreate** will use the ODBC table names, rather than the structure names, in the generated system catalog. See the "Assigning Structures to Files" in the "Working with Files" chapter of the *Repository User's Guide* for information. (Note that this is *not* related to the SODBC_ODBCNAME environment variable, which enables you to use the field name specified in the Repository Alternate name field attribute.)

Temporary files

By default, when **dbcreate** generates a system catalog, it includes tables that describe temporary files (files for which the Repository Temporary flag is set). To omit tables that describe temporary files, set the SODBC_TMPOPT environment variable as described in "Excluding tables attached to temporary files" on page 3-26.

User-defined data types

For information on creating routines that manipulate data in user fields, see chapter 7, "Creating Routines for User-Defined Data Types."

Zeros, spaces, and null values

For information on how *xf*ODBC interprets zeros, spaces, and null values, and for information on how to prevent fields from being updated with null values (and other values that *xf*ODBC considers null), see "Preventing null updates and interpreting spaces, zeros, and null values" on page 3-27.

Other field attributes

The dbcreate utility and DBA also use the following repository settings as column attributes in the system catalog:

- Field size, type, and precision
- "Excluded by ReportWriter" settings (see "Including and omitting fields" on page 3-23)
- Alternate name, if SODBC_ODBCNAME is set when the system catalog is generated (see "Renaming columns for clarity" on page 3-24)
- Negative-allowed and range validation attributes, which are used to determine if numeric fields are signed or unsigned (see "Instructing dbcreate to ignore the "Negative allowed?" field in Repository" on page 3-24)

Setting Options and File Locations

The following sections list ways you can set file locations, generation options, and data conversion options. Many of these can be set with environment variables, some of which are set at installation. Other environment variables aren't set until you set them. Note the following:

- For information on system catalog generation settings you make with **dbcreate** or DBA options, see "Generating a system catalog from the command line" on page 4-3 and "Using DBA to generate a system catalog" on page 4-6.
- We use the term "environment variable" for all platforms, even though the term isn't generally used for OpenVMS. If you're on OpenVMS, substitute "logical" for "environment variable."
- For information on how to set environment variables and what we mean by "system-wide" and "the environment", see "Setting system-wide variables" on page 3-30 and "Setting variables in the environment" on page 3-31.
- See "Appendix A: Environment Variables" for a full list of *xf*ODBC environment variables.

Specifying file locations

Before you can generate a system catalog, you must specify the location of data files (see "Specifying the location of data files" on page 3-21). If you've set any environment variables in an environment setup file, you must also specify the location of that file (see "Specifying the location of an environment setup file" on page 3-20).

In addition, other file locations must be set before you can open a system catalog in DBA, use a conversion setup file, or access data with an ODBC-enabled application. See

- "Specifying the connect file location (GENESIS_HOME)" on page 3-19. This is set automatically, but you can change the setting.
- "Specifying a conversion setup file" on page 3-26. This is necessary only if you've created a conversion setup file.

Specifying the location of DBA and dbcreate

The location of the DBA program is automatically set. Although it's seldom necessary, you can change this setting. See "SODBC_DBA" on page A-7.

The location of **dbcreate** is automatically set system-wide when you install *xf*ODBC (except on 64-bit Windows). This enables you to run **dbcreate** from the directory that contains the system catalog.

- On 32-bit Windows and on UNIX, the location of **dbcreate** is added to the PATH setting.
- On OpenVMS, **dbcreate** is set as a DCL command when *xf*ODBC is installed.

Note that for client/server configurations, the location of **dbcreate** should be set on the server.

Specifying the connect file location (GENESIS_HOME)

The GENESIS_HOME environment variable must be set to the directory that contains the connect files, which specify the location of system catalogs and data files. (See chapter 5, "Setting Up a Connect File.") It is used by the *xf*ODBC driver, DBA, and **dbcreate**, and it's used when you generate system catalogs, when you modify system catalogs, and when you connect to the database. Note the following:

- GENESIS_HOME is required and is automatically set when you install *xf*ODBC. You can, however, change this setting. Note that because GENESIS_HOME is set at the system level, if you install both 32-bit and 64-bit versions of Connectivity Series on the same 64-bit Windows machine, the last version installed determines the GENESIS_HOME setting by overwriting the previous setting. In this case, we recommend that you set GENESIS_HOME to a location outside Program Files and use it for both 32-bit and 64-bit.
- For stand-alone configurations on Windows and UNIX, GENESIS_HOME must be set in the environment.
- For stand-alone configurations on OpenVMS, GENESIS_HOME must be set in **CONNECT_STARTUP.COM**.
- In client/server configurations, GENESIS_HOME must be set in **opennet.srv** on the server, and it must be set before starting the SQL OpenNet server (which only uses settings made before it's started and doesn't use settings in **synergy.ini**). Note that as distributed, **opennet.srv** already defines GENESIS_HOME. Additionally, GENESIS_HOME must also be set in the environment on the client unless you set GENESIS_MSG_FILE.
- For Windows Vista and higher, we recommend that you change the GENESIS_HOME setting and move files for the sample database and repository to a writable location outside of Program Files so that files can be created and updated.

If you change the setting for GENESIS_HOME, *xf*ODBC may not be able to locate the error message file, which is required to generate a system catalog. If the GENESIS_MSG_FILE environment variable is not set, *xf*ODBC looks for the error message file in GENESIS_HOME\lib. See "Specifying the name and location of the error message file" below for more information.

Specifying the name and location of the error message file

To generate a system catalog, *xf*ODBC must be able to locate the error message file. If the GENESIS_MSG_FILE environment variable is set, **dbcreate** uses this setting to locate the file. If this variable is *not* set, **dbcreate** attempts to locate the default error message file, **sql.msg**, in the GENESIS_HOME\lib directory. Note the following:

- When you install Connectivity Series, GENESIS_MSG_FILE is automatically set to **sql.msg**, the default error message file. Note, however, that if you install both 32-bit and 64-bit versions of Connectivity Series on a 64-bit Windows machine, the last version installed determines which **sql.msg** file this is set to. (The 32-bit installation and the 64-bit installation each have an **sql.msg** file.)
- If you set GENESIS_MSG_FILE, set it in the environment, and set it to the path and filename of the error message file. For client/server configurations, it must be set in the environment on the client and on the server. (For services such as web servers that use the *xf*ODBC driver, you can use the Env. variables field in the xfODBC Setup window to set this on the client. For information, see "Adding a user or system DSN" on page 8-5.)

For information on editing the **sql.msg** file, see "Editing the SQL Message File" on page 11-10.

Specifying the location of an environment setup file

An environment setup file is a file that you write to define environment variables that are used by *xf*ODBC when locating Synergy data files. It typically has an **.ini** filename extension and is placed in the GENESIS_HOME directory, although these are not requirements. (For information on environment setup files, see "Setting environment variables in an environment setup file" on page 3-34.)

To use an environment setup file, the SODBC_INIFIL environment variable must be set to the path and filename of the environment setup file. Note the following:

- SODBC_INIFIL is used by the *xf*ODBC driver when you connect to a database. (It is not used by DBA or **dbcreate**, so it is not used when you create or modify a system catalog.)
- In a stand-alone configuration, set SODBC_INIFIL in a connect file or in the environment.
- In a client/server configuration, set SODBC_INIFIL in the environment on the server or in the **opennet.srv** file (Windows only).
- SODBC_INIFIL is *not* set during installation.

If SODBC_INIFIL is set in the environment when you access your Synergy data, *xf*ODBC will ignore environment variables set in the connect file. To use environment variables set in the connect file, either make sure SODBC_INIFIL is *not* set or is set in the connect file.

Specifying the location of data files

For repository files, you can specify the path and filename

- at the command-line for **dbcreate**. (See "Generating a system catalog from the command line" on page 4-3.)
- in the Generate System Catalog window of the DBA program. (See "Using DBA to generate a system catalog" on page 4-6.)

For Synergy data files, you can specify

- the filename or the path and filename in the Open filename field of Repository. If the path is not specified here, *xf*ODBC uses the path in the datasource line of the connect file.
- the path in the dictsource or datasource line of the connect file. Settings in these lines are used if no path is specified in the Repository Open filename field, and the datasource setting is used only if no data files exist in the dictsource directory. *xf*ODBC always gets the data file name from the Open filename field. (See "The dictsource and datasource Lines" on page 5-3.)

  Note that we recommend using an environment variable to specify the path in Open filename field of the repository rather than relying on dictsource or datasource.

There are two types of environment variable you can use to specify the location of data files:

- *User-created data location variables*—i.e., environment variables you create and use in the place of hard-coded paths and filenames.
- *Repository file location variables*. These are RPSDAT, RPSMFIL, and RPSTFIL. These environment variables are automatically set by the installation (except on 64-bit Windows), and in some cases *xf*ODBC automatically uses their settings.

User-created data location variables

Rather than hard-coding a path in the Open filename field of Repository, you can define your own data-location environment variables from the command line for **dbcreate** or with the Generate System Catalog window of DBA. Note the following:

- These variables can be used for both system catalog generation and data access. For example, if you use one of these variables at the command line for **dbcreate** or in the Generate System Catalog window of DBA, the variable is used to locate the repository files used to generate the system catalog. On the other hand, if you use one of these variables in the Open filename field of Repository or in the datasource line of the connect file, *xf*ODBC uses the variable to locate database files for data access.
- User-created data location variables can be defined in the connect file, in an environment setup file, and in the environment.
- For client/server configurations, user-created data location variables must be defined on the server.

- To access data, *xf*ODBC uses the following steps to resolve data-location variables that point to database files:
  1. If SODBC_INIFIL is *not* set or is set in the connect file, *xf*ODBC first considers variables defined in the connect file. (If SODBC_INIFIL is set in the environment, *xf*ODBC ignores environment variables set in the connect file.)
  2. Next, *xf*ODBC searches the environment setup file for variable definitions.
  3. *xf*ODBC then considers variables defined in the current environment, and then it considers variables defined system-wide (for both stand-alone and client/server configurations).
  4. Finally, if *xf*ODBC is still not able to resolve the variable after checking the connect file, the environment setup file, and the environment and system-wide definitions, *xf*ODBC returns an error message indicating that the file was not found.

If you use a hard-coded path in the Open filename field of a Repository file definition, at the command line for DBA or **dbcreate**, or in the Generate System Catalog Window of the DBA, this path is the only one *xf*ODBC considers when attempting to locate the data file.

See "Create a connect file" on page 2-5 for an example data location variable, XFDBTUT, that works with the tutorial. See step 4 on page 4-7 for an example used in the Generate System Catalog window of DBA.

Specifying repository file locations

When locating repository files for generating the system catalog, DBA and **dbcreate** search in different ways:

- The **dbcreate** utility looks first to the command line and uses any repository files specified by the **-r** option. If the command line doesn't include the **-r** option, **dbcreate** uses the RPSMFIL and RPSTFIL environment variable settings. If these aren't set, **dbcreate** looks for the **rpsmain.ism** and **rpstext.ism** files in the directory specified by the RPSDAT environment variable. If these files don't exist in this location, **dbcreate** returns an error.
- DBA looks first to the RPSMFIL and RPSTFIL environment variable settings. If these have been set, DBA uses the settings to pre-fill the Main repository and Text repository fields of the Generate System Catalog input window. If you change these fields, the new values will override the RPSMFIL and RPSTFIL settings. If RPSMFIL and RPSTFIL are blank, the Main repository and Text repository fields will be blank when the Generate System Catalog input window opens.

Note the following:

- RPSDAT specifies the location (path only, not filename) of the repository files. This variable works only if the repository files are named **rpsmain.ism** and **rpstext.ism**. **dbcreate** uses this variable, but DBA does not.
- RPSMFIL specifies the path and filename of the repository main file. Both **dbcreate** and DBA use this variable.

- RPSTFIL specifies the path and filename of the repository text file. Both **dbcreate** and DBA use this variable.
- The installation automatically sets RPSDAT, RPSMFIL, and RPSTFIL (except on 64-bit Windows), but they're not required for *xf*ODBC. If your repository files are not in one of these directories, you must hand-code the paths to the repository files to generate or modify the system catalog.
- On Windows, these variables must be set in the environment or in **synergy.ini**. (Note that **dbcreate** doesn't use environment variables set in **synergy.ini**.)
- For UNIX and OpenVMS, these variables must be set in the environment.

Setting catalog generation options

In addition to the options you set at the command line for **dbcreate** or in the Generate System Catalog window of DBA, you can specify some system catalog generation options by setting environment variables and S/DE Repository options. You are not required to set any of these options, but if you set any of the environment variables documented here, you must set them in the environment. We recommend creating a batch file, shell script, or DCL command file, setting the environment variables in this file, and then running the file before using the DBA program or **dbcreate**.

See "Setting Runtime Data Access Options" on page 8-13 for information on options that affect the way *xf*ODBC behaves as it accesses data, including more settings that determine how the *xf*ODBC driver interprets data.

Including and omitting fields

If the S/DE Repository option "Excluded by ReportWriter" is checked for a field, **dbcreate** will *not* include the field in the system catalog, so the field will not be available to ODBC-enabled applications. To include the field in the system catalog, clear this option in Repository. (See "Defining a New Field" in the "Working with Fields" chapter of the *Repository User's Guide* for information.) However, if you want all fields to be included in the system catalog, regardless of their "Excluded by ReportWriter" settings, set the SODBC_CNVOPT environment variable.

Note the following:

- To include all fields, set SODBC_CNVOPT to 1.
- Set this variable in the environment. For client/server configurations, set it where you run **dbcreate**.
- If a field is used as a *structure tag* or *key segment*, it's automatically included, regardless of the report exclusion flag or the SODBC_CNVOPT setting.

Omitting keys

By default all keys defined in the repository are used to define indexes in the system catalog. To omit a key from the system catalog, set the S/DE Repository option “Excluded by ODBC” for the key definition. (See “Defining Keys” in the “Working with Files” chapter of the *Repository User’s Guide* for more information.) Note that this option does not affect the inclusion of fields in the system catalog (even fields specified in the key definition), just the key. Every field that is a key or key segment is included in the system catalog. See “Tags and optimization” on page 10-8 for recommendations that utilize this option.

Instructing dbcreate to ignore the “Negative allowed?” field in Repository

The **dbcreate** utility and the *xf*ODBC driver distinguish between signed and unsigned numeric fields. When **dbcreate** generates a system catalog, it checks the “Negative allowed?” repository setting to determine if the resulting column will be signed or unsigned.

- If the “Negative allowed?” setting is No, the resulting column will be unsigned.
- If the “Negative allowed?” setting is Only, OrZero, or Yes, the resulting column will be signed unless a range that includes only positive values is assigned to the field, in which case the column will be unsigned.

Prior to Connectivity Series version 8.3, **dbcreate** ignored the “Negative allowed?” field and set all fields to signed unless they had validation ranges that were limited to positive values (in which case the resulting columns were unsigned). To revert to this behavior, set the SODBC_NOUNSIGNED environment variable to any value.

- By default, SODBC_NOUNSIGNED is not set.
- Set SODBC_NOUNSIGNED in the environment. For client/server configurations, set it where you run **dbcreate**.
- Set SODBC_NOUNSIGNED *before* generating the system catalog.

Renaming columns for clarity

Repository field names that are short and cryptic may not make good column names. As an alternative, you can use Alternate name field values (specified in the repository) as column names by setting the SODBC_ODBCNAME environment variable. When SODBC_ODBCNAME is set, *xf*ODBC uses a field’s Alternate name value if it’s set; otherwise it uses the field’s name. Note the following:

- To use the values in the Repository Alternate name field as column names, set SODBC_ODBCNAME to 1.
- Set SODBC_ODBCNAME in the environment. For client/server configurations, set it where you run **dbcreate**.

Note that this is *not* related to the Repository ODBC table name option, which enables you to assign ODBC *table* names to file/structure combinations.

Generating one column for an array field

By default, each element in an array field is mapped to a separate column in the system catalog (see "Arrays" on page 3-12). You can, however, use the SODBC_COLLAPSE environment variable to instruct **dbcreate** to map all elements of an array field to a single system catalog column if the number of elements in the array is greater than or equal to the limit you specify—that is, the number you set SODBC_COLLAPSE to. (Note that you should use SODBC_COLLAPSE if a system catalog table will have more than 254 columns. Some ODBC-enabled applications do not permit tables with more than 254 columns.)

For example, if you set SODBC_COLLAPSE to 10 and your repository has a structure with three array fields—one with six elements, one with eight elements, and one with 10 elements—the corresponding table in the system catalog will have 15 columns: six for the first array, eight for the second array, and 1 for the third array (because it reached the limit set by SODBC_COLLAPSE).

Note the following:

- If you set SODBC_COLLAPSE, set it in the environment. For client/server configurations, set it in the environment where you run **dbcreate**.
- SODBC_COLLAPSE does not affect group arrays. Group arrays cannot be collapsed.

Changing the position delimiter used for arrays

When you generate a system catalog for a repository that has an array field, each element in the array is mapped as a separate system catalog column with a name that consists of the array name, the element's position in the array, and pound signs (#) to delineate position values. (See "Arrays" on page 3-12) For example, a [2,2] array field with the name myarray will be mapped to the following: myarray#1#1, myarray#1#2, myarray#2#1, and myarray#2#2.

You can, however, change the character used to delineate position values by setting the SODBC_TOKEN environment variable to the character you want to use. (Make sure you set it to a valid SQL identifier value for your ODBC applications.) For example, if you set SODBC_TOKEN=_, the myarray field described above would result in the following system catalog columns: myarray_1_1, myarray_1_2, myarray_2_1, and myarray_2_2.

If you set SODBC_TOKEN, set it in the environment. For client/server configurations, set it where you run **dbcreate**.

Removing group and struct names from column names

By default, if a field is part of a group or struct field in the repository, the group or struct name is added to the field name to create the column name for the system catalog. (See "Groups and struct fields" on page 3-16 for information.) To omit group and struct names from column names, use the SODBC_NOGROUPNAME environment variable—but do this only if you are certain the resulting column names will be unique.

Note the following:

- Set SODBC_NOGROUPNAME to any value to omit group and struct names.
- Set this variable *before* generating your system catalog.
- Set this variable in the environment. For client/server configurations, set it where you run **dbcreate**.

If a group or struct field is an array, **dbcreate** generates a column for each element in the array (in addition to an overlay column), using the naming convention documented in "Arrays" on page 3-12. However, if SODBC_NOGROUPNAME is set, instead of using the group or struct name as part of the names for these columns, the names will start with "GR". For example, if a repository has a group or struct field named myarray, that is a [2,2] array with a single field, myfield, the group or struct field will be mapped to the following columns: myarray (the overlay field), GR#1#1MYFIELD, GR#1#2MYFIELD, GR#2#1MYFIELD, and GR#2#2MYFIELD.

Excluding tables attached to temporary files

By default, when **dbcreate** generates a system catalog, it includes tables that describe temporary files (files for which the Repository Temporary flag is set). To exclude tables attached to temporary files from the system catalog, set SODBC_TMPOPT to 1. Set SODBC_TMPOPT in the environment. For client/server configurations, set it where you run **dbcreate**.

Specifying a conversion setup file

Conversion setup files are used when you regenerate a system catalog. They enable you to make changes to system catalog settings, such as the paths and filenames for data files, access levels for tables, and so forth. To use a conversion setup file, you must generate one and then specify it *before* or *as* you use DBA or **dbcreate** to regenerate the system catalog. (For information on what you can do with a conversion setup file and on generating one, see "Generating and Editing a Conversion Setup File" on page 6-27.)

To specify a conversion setup file *as* you use **dbcreate** or DBA, use the **-i** command line option for **dbcreate**, or use the Conversion setup field in the Generate System Catalog window of the DBA. For information on **dbcreate** command line options, see "Generating a system catalog from the command line" on page 4-3. For information on the Conversion setup field, see "Using DBA to generate a system catalog" on page 4-6.

To specify the file *before* you use DBA or **dbcreate**, set the SODBC_CNVFIL environment variable to the path and filename of the conversion setup file. If SODBC_CNVFIL is set, DBA and **dbcreate** automatically use the conversion setup file whenever you regenerate the system catalog. You won't need to set a command line option, and the Generate System Catalog window of DBA will automatically specify the conversion setup file. In addition, if SODBC_CNVFIL is set, tables you delete in DBA will also be marked for deletion in the conversion setup file.

Note the following:

- The SODBC_CNVFIL environment variable should not be set until the conversion setup file has been created.
- If the SODBC_CNVFIL environment variable is set, it must be set in the environment.
- For client/server configurations, set SODBC_CNVFIL where you run **dbcreate**, and put the conversion setup file where it can be accessed by **dbcreate**.
- If you use the conversion setup file command line option (**-i**) for **dbcreate** *without* specifying a filename, the conversion setup file is *not* used—even if SODBC_CNVFIL is set.

While in DBA, changes you make to tables automatically and immediately update *both* the system catalog *and* the conversion setup file specified by the SODBC_CNVFIL environment variable. Be careful to use SODBC_CNVFIL to specify the exact conversion setup file for the system catalog you are modifying before you open DBA.

Using decimal information in the repository format string

If your repository has a field that's not an implied decimal, but has a format string with a decimal point, you can instruct **dbcreate** and DBA's Generate option to use the decimal information in the format string to create an implied decimal column in the system catalog. To do this, set the SODBC_USEFORMAT environment variable to 1 *before* you generate the system catalog. For example, if SODBC_USEFORMAT is set to 1 and your repository has a **d5** field with an XXX.XX format string, the field will appear as a **d5.2** column in the system catalog.

Note the following:

- To use the decimal information in the format string, set SODBC_USEFORMAT to 1.
- Set SODBC_USEFORMAT in the environment. For client/server configurations, set it where you run **dbcreate**.

Preventing null updates and interpreting spaces, zeros, and null values

*xf*ODBC uses the "Null allowed" setting for a column to determine the following:

- Whether an alpha, decimal, date, or time column can be updated with a null value or some other value that *xf*ODBC considers equivalent to null (spaces for alpha fields and alpha dates, and zeros for decimal and time fields).
- How nulls and spaces in some columns are interpreted. See "How spaces, null values, and zeros are interpreted when read from a database" on page 3-29.

Note that if "Null allowed" is set to no for a column, the column must be included in every INSERT statement for the table.

To see what this setting is for a column, use SQLDescribeCol to get the NullablePtr setting:

- If NullablePtr=SQL_NULLABLE=1, nulls are allowed for the column (i.e., "Null allowed" is set to yes).
- If NullablePtr=SQL_NO_NULLS=0, nulls are not allowed (i.e., "Null allowed" is set to no).

You can also use the DBA program to view the "Null allowed" setting for the column (see "Viewing information about a column" on page 6-24).

Setting "Null allowed" for a column

To set this property for a column, set the "Null allowed" Repository option for the field before generating the system catalog. (See "Validation information" in the "Working with Fields" chapter of the *Repository User's Guide*.)

- If the "Null allowed" Repository option is set to Yes or No for a repository field, **dbcreate** uses this setting for the system catalog column it generates for that field. For alpha, date, decimal, and time fields, you can set this option to Yes, No, or Default. For other fields, it can be set only to No or Default.
- If the "Null allowed" Repository option is set to Default for a repository field, the system catalog column generated for that field will be set to allow nulls unless it is a Boolean, binary, or integer field, or a non-date field that is part of the definition for the first key for the table. (The SODBC_NONULL environment variable changes this behavior, but this environment variable is deprecated. See SODBC_NONULL in the "Environment Variables" chapter of *Environment Variables & System Options* for more information.)

We recommend that you set every repository field used in a key definition to preclude nulls (i.e., set the Repository "Null allowed" option for the field to No), except those fields that must actually be able to accept null values. This is particularly important if you access your data in a .NET environment. The ADO.NET Entity Framework does not use nullable fields to optimize queries and ignores primary keys that contain nullable fields.

The "Null allowed" setting is yes for columns that are part of a table added with CREATE TABLE unless you use NOT NULL in the CREATE TABLE statement. See CREATE TABLE on page B-53.

How spaces, null values, and zeros are interpreted when read from a database

When reading from a database, the *xf*ODBC driver interprets spaces and null values differently for some columns depending on how "Null allowed" is set. Note the following:

- *xf*ODBC interprets zero-length strings as nulls.
- *xf*ODBC interprets invalid data as null, unless you set the convert_error option (see "Setting the convert_error Option" on page 5-4).
- The behavior for user types is the same as their base types. For example, see the Alpha row in the table below for information on how the "Null allowed" property affects user-defined alpha fields.

Data type	Column value	If "Null allowed" is yes,[a] value read as...	If "Null allowed" is no,[b] value read as...
Alpha	Spaces	Null	Spaces
	Null	*Spaces (filled to max length)*	*Spaces (filled to max length)*
Binary	Zero	Zero	Zero
Boolean	Zero	False	False
Decimal	Spaces	Null	Zero
	Zero	Zero	Zero
	Null	Null	Zero
Date	Spaces	Null	Null
	Zero	Null	Null
	Null	Null	1-1-0001
Integer	Zero	Zero	Zero
Time	Spaces	Null	Null
	Zero	Null	Null
	Null	Null	00:01

a. I.e., if SQL DescribeCol returns NullablePtr=SQL_NULLABLE=1 for the column.
b. I.e., if SQL DescribeCol returns NullablePtr=SQL_NO_NULLS=0 for the column.

Setting environment variables

This section describes the methods you can use to set environment variables. The method you should use depends on the configuration of *xf*ODBC (client/server or stand-alone), what programs you want the setting to affect, and whether you are using Windows, UNIX, or OpenVMS. You can set environment variables by

- typing them at a Windows, UNIX, or OpenVMS prompt. See page 3-31.
- writing a batch file, shell script, or DCL command file that runs in the same environment you run **dbcreate** in. See page 3-32.
- entering them in the connect file. (This is especially useful for data file locations.) See page 3-33.
- entering them in the xfODBC Setup window. See page 3-34.
- creating an environment setup file, setting the variables in this file, and setting SODBC_INIFIL to the name of this setup file. See page 3-34.
- entering them in **synergy.ini**. See page 3-34.
- entering them in **opennet.srv**. See page 3-35.
- entering them in **net.ini**. See page 8-26.
- entering them in System Properties on Windows. See page 3-35.
- entering them in a log-in file on UNIX. See page 3-35.
- entering them in a log-in file on OpenVMS. See page 3-35.

For client/server configurations, set environment variables only on the server. Environment variables set on the client are not recognized.

Environment variables are capitalized in our documentation to distinguish them from filenames and paths, which are usually lowercase or a combination of uppercase and lowercase. You are *not* required to follow the case in our examples when you enter commands and statements on Windows or OpenVMS systems, but you must for UNIX.

Setting system-wide variables

On Windows and OpenVMS, environment variables for services and ODBC-enabled applications must be set system wide. On UNIX, which doesn’t support system-wide environment variables, you must set environment variables before starting the program or service that uses them. (Often it’s convenient to do this with a shell script.)

To set a system-wide environment variable,

- on Windows, set it in the Environment Variables window or on the Environment tab. See "Setting environment variables in System Properties" on page 3-35.
- on OpenVMS, use the /SYS command when you set the environment variables. For an example, see "Setting environment variables in a batch file, shell script, or DCL command file" on page 3-32.

Setting variables in the environment

When we use the term "the environment" to describe where an environment variable should be set, you should set it

- at a Windows, UNIX, or OpenVMS prompt.
- in a batch file, shell script, or DCL command file.
- in a log-in file on UNIX or OpenVMS.
- in the Environment Variables window or on the Environment tab. See "Setting environment variables in System Properties" on page 3-35.

Note that this list does not include environment variables set in a connect file, environment setup file, **synergy.ini**, or **opennet.srv**. This is because variables set in these locations are available only to the Connectivity Series products that use them. They're not available to other programs running in the environment.

Setting environment variables from a Windows, UNIX, or OpenVMS prompt

Environment variables set at a command prompt are temporary. They exist only in the current environment and do not apply when you leave the environment. For example:

- At a Windows prompt, enter

  ```
  SET DATA=%GENESIS_HOME%\dat
  ```

- At a UNIX prompt, enter

  ```
  DATA=$GENESIS_HOME/dat; export DATA
  ```

- At an OpenVMS prompt, enter

  ```
  DEFINE DATA GENESIS_HOME:[DAT]
  ```

In these examples, the environment variable DATA is set to a directory named "dat", a subdirectory of the GENESIS_HOME directory.

Setting environment variables in a batch file, shell script, or DCL command file

You can also set environment variables from a batch file (Windows), shell script (UNIX), or DCL command file (OpenVMS). The advantage of this method is that the variables are set only when necessary. Moreover, they are no longer in effect once the current process terminates—for example, when you close a Command Prompt window in Windows, log off in OpenVMS, or exit a shell on UNIX. (Note that on UNIX, an environment variable must be exported to the shell for it to be available to additional programs run from the shell.) This is a particularly good way to store variables that need to be set only for system catalog generation.

1. Using a text editor, create a batch file, shell script, or DCL command file and set *xf*ODBC environment variables in the file. Use standard environment variable syntax. For example:

   - Windows:

     ```
     set SODBC_CNVOPT=1
     ```

   - UNIX:

     ```
     SODBC_CNVOPT=1 ;export SODBC_CNVOPT
     ```

   - OpenVMS:

     ```
     DEFINE/SYS SODBC_CNVOPT 1
     ```

     (/SYS is optional. It sets the environment variable as a system-wide logical on OpenVMS.)

2. From the command line, run the batch file, script, or DCL command file.

3. After running the file, run **dbcreate** or DBA from the command line in the same environment.

   On UNIX and OpenVMS, your *xf*ODBC distribution includes shell scripts (**setodbc** and **startnet**) or a DCL command file (**STARTNET.COM**) you can use to specify environment variables. Note the following:

   - These files are replaced every time you install Connectivity Series. So if you set an environment variable in one of these files, you'll have to reset it after installing.
   - To use **setodbc** on UNIX, you must have configured your session for Synergy/DE by running **setsde**. For information, see "SQL Connection and xfODBC on UNIX" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.
   - If you use **startnet** or **STARTNET.COM**, environment variable settings should precede the line that starts the SQL OpenNet server. On UNIX, this is the **vtxnetd** command. On OpenVMS, this is the command that runs **NET.COM**. For example:

     ```
     SODBC_ODBCNAME=1; export SODBC_ODBCNAME
     nohup vtxnetd -p1958 log &
     sleep 1
     ```

Setting environment variables in a connect file

You can also use your connect file to define environment variables for the *xf*ODBC driver. These definitions are read by the *xf*ODBC driver when connecting to a database and remain in effect for the duration of the connection. Environment variables set in the connect file are used to locate data files and set some data access options. Note the following:

- Do *not* include the SET (Windows), EXPORT (UNIX), or DEFINE (OpenVMS) commands in the environment variable definition, and do not use other environment variables in the definition. For example, the following illustrates the syntax used to set an environment variable on a Windows system.

  ```
  XFDBTUT=C:\Program Files\Synergex\SynergyDE\connect\synodbc\dat
  ```

- If SODBC_INIFIL is set in the environment, *xf*ODBC won't use environment variable settings in the connect file. If SODBC_INIFIL is set in the connect file, *xf*ODBC looks to the connect file for environment variable definitions before it looks in the environment setup file. (See "Specifying the location of an environment setup file" on page 3-20 for more information.)
- If you define more than one environment variable, put them on separate lines. For example:

  ```
  CUST=c:\data\customer.ism
  ORDER=c:\data\orders.ism
  PLANTS=c:\data\plants.ism
  ```

The following connect file examples set an environment variable named XFDBTUT (an environment variable used in the sample Synergy database and set in the sample connect file):

- Windows:

  ```
  dictsource "C:\Program Files\Synergex\SynergyDE\connect\synodbc\dict\"
  datasource ";C:\\Program Files\\Synergex\\SynergyDE\\connect\\syn-
  odbc\\dat;"
  XFDBTUT=C:\Program Files\Synergex\SynergyDE\connect\synodbc\dat
  ```

- UNIX:

  ```
  dictsource  /usr/synergyde/connect/synodbc/dict
  datasource ;/usr/synergyde/connect/synodbc/dat;
  XFDBTUT=/usr/synergyde/connect/synodbc/dat
  ```

- OpenVMS:

  ```
  dictsource  DKA300:[SYNERGYDE.CONNECT.SYNODBC.DICT]
  datasource ;DKA300:[SYNERGYDE.CONNECT.SYNODBC.DATA];
  XFDBTUT=DKA300:[SYNERGYDE.CONNECT.SYNODBC.DATA]
  ```

If you set SYNCENTURY and SYNBASEDATE, you must set them in the connect file if the connect file is on the server.

For more information on connect files, see chapter 5, "Setting Up a Connect File."

Setting environment variables in the xfODBC Setup window

The xfODBC Setup window has two fields you can use to set environment variables used for data access (not system catalog generation).

- The "Appended to connect string" field, which enables you to define environment variables used by the target database on a server. Note that to use this field, you must use SQL OpenNet (i.e., you must select Net in the "Vortex driver" field of the xfODBC Setup window).
- The "Env. variables" field, which enables you to set environment variables used by the *xf*ODBC driver on the client. This is the only place these environment variables can be set for services such as web servers that use the *xf*ODBC driver.

For information on the xfODBC Setup window and the syntax for setting environment variables in these fields, see "Adding a user or system DSN" on page 8-5.

Setting environment variables in an environment setup file

An environment setup file is one way to store and activate all of the data environment variables you need when connecting to a Synergy database. An environment setup file is a text file you write; it typically has an **.ini** filename extension and is placed in the GENESIS_HOME directory, though these are not requirements. Note the following:

- Define environment variables in the same way you define them in a connect file (except that you can use environment variables in the definition)—for example:

  ```
  XFDBTUT=%CONNECTDIR%synodbc\dat
  ```

- Assign the environment setup file's path and filename to SODBC_INIFIL in the environment. For stand-alone configurations, it can also be set in the connect file. For client/server configurations, it must be set on the server in the environment or in the **opennet.srv** file (Windows only). *xf*ODBC is coded to look for a file assigned to this variable and to read it for data environment variables.

Setting environment variables in synergy.ini (Windows)

Because the DBA program, **xfdba.dbr**, is a Synergy application and can read **synergy.ini**, you can use this to set environment variables used by DBA. This is read every time you run **dbr**, the Synergy runtime.

The *xf*ODBC driver and **dbcreate** don't recognize environment variables set in **synergy.ini**.

The following is an example of an environment variable set in **synergy.ini**:

```
SODBC_DBA=%CONNECTDIR%synodbc\dba
```

Note that in **synergy.ini**, SET is not included in the syntax, and an environment variable can be used as part of another environment variable's definition.

Setting environment variables in opennet.srv (Windows)

Environment variables used by the SQL OpenNet server can be defined in the **opennet.srv** file. For example, as distributed, the **opennet.srv** file defines GENESIS_HOME before starting the Synergy/DE OpenNet Server service (**SynSQL**):

```
GENESIS_HOME=%CONNECTDIR%synodbc\
.
.
.
vtxnetd.exe -k67834 -p1958 log
```

This sets the environment variable and then launches the service, overriding any system-set variable. Note that SET is omitted.

For more information on **opennet.srv**, see "Customizing the opennet.srv file" in the Windows section of the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.

Setting environment variables in System Properties

On Windows, you can set system-wide environment variables from System Properties. Select System, available from Windows Control Panel. (You may also need to select "Advanced System Settings" or click "Change settings.") Go to the Advanced tab of the System Properties window, click the Environment Variables button, and then click the New button under the System variables area of the dialog box.

Setting environment variables in log-in files on UNIX

On UNIX systems, you can set environment variables in the log-in file. The log-in file for UNIX systems is **.profile** if you're using Bourne or Korn shell; if you're using C Shell, the log-in file is **.login**. After setting an environment variable on UNIX, you must export it unless you have the "auto-export" feature turned on in your shell. (Refer to your UNIX reference manual for details on auto-export. Not all UNIX systems offer this option.)

For example:

```
GENESIS_HOME=$CONNECTDIR/synodbc
export PATH GENESIS_HOME
```

Setting environment variables in log-in files on OpenVMS

On OpenVMS systems, you can set environment variables (logicals) in your user log-in file or in the system-wide log-in file located in SYS$MANAGER. You can also set system-wide environment variables in **SYS$MANAGER:CONNECT_STARTUP.COM**, but remember that this file is replaced every time you install Synergy products. The following is an example of an environment variable set in a log-in file:

```
$ DEFINE GENESIS_HOME SYNERGYDE$ROOT:[CONNECT.SYNODBC]
```

4

Creating a System Catalog

To make Synergy data accessible to ODBC-enabled applications, you must create a system catalog. The system catalog is generated from repository definitions and provides the information the *xf*ODBC driver needs to access the data files.

Generating the System Catalog

Before generating a system catalog,

- install *xf*ODBC. See "xfODBC requirements and installation" on page 1-8.
- define your database's schema in a repository. See "Setting Up a Repository" on page 3-2.
- set any needed environment variables for file locations, data conversion options, and other system catalog generation options. See "System Catalog Generation Issues" on page 3-12 and "Setting Options and File Locations" on page 3-18.

  You can set some system catalog generation options when you generate. See "Generating a system catalog from the command line" on page 4-3 and "Using DBA to generate a system catalog" on page 4-6.

You should also compare the repository definitions to the Synergy database files. See "Validate, verify, and compare" on page 3-7.

Once you've done these things, you're ready to generate the system catalog. you can generate the system catalog from the command line using the **dbcreate** utility, or you can generate it from the *xf*ODBC Database Administrator (DBA), a program that you can also use to modify the system catalog. We recommend you use **dbcreate**. The **dbcreate** utility enables you to see messages that document the system catalog generation process.

When you generate a system catalog, **dbcreate** or DBA will

- read file/structure combinations defined in your repository.
- create a system catalog that consists of the ISAM files listed in "System catalog" on page 1-5.
- create a unique data entry in the system catalog for each file/structure combination in your repository.
- create an initial set of users and groups. Note that these will be created only if you set the initialization option in **dbcreate** or DBA. Without the initial set of users and groups, you won't be able to customize the system catalog. See "Initializing users and groups" on page 6-13 for details on the initial set of users and groups.

To generate a system catalog, *xf*ODBC must be able to locate the error message file. For information, see "Specifying the name and location of the error message file" on page 3-20.

For information on generating a system catalog or a set of system catalogs for databases that share a repository, see "Handling a repository shared by multiple databases" on page 3-7.

Generating a system catalog from the command line

You can run **dbcreate** from the command line on a Windows, UNIX, or OpenVMS system. For client/server configurations, **dbcreate** must be run from the server. **Dbcreate** is in the connect subdirectory of the main Synergy/DE installation directory. (For information on options used to regenerate a system catalog, see "Regenerating the system catalog with dbcreate" on page 4-9.)

Use the following syntax:

`dbcreate` *[option] [...]*

where *option* is one of the following:

-? Displays online help for **dbcreate** command-line options. This option is the same as **-h**).

-h Displays online help for **dbcreate** command-line options. This option is the same as **-?**).

-c Generates a system catalog from a repository. If it's generated to a directory that already contains a system catalog, the system catalog is overwritten. (User and group files, however, are not created or overwritten unless this option is used in conjunction with **-p**.) This option is the default if you generate to a directory that doesn't have a system catalog.

If you use a conversion setup file with the **-i** option, tables marked as OUT are omitted from the system catalog.

-x Updates a system catalog. If the repository has new structures, the new structures are added to the system catalog as new tables. If the repository has structures that are different than the corresponding tables in the system catalog, these tables are updated. (They wouldn't be updated if you used the **-u** option.) User and group files are not affected unless this option is used with **-p**.

If you use a conversion setup file, tables marked as OUT are *not* overwritten or removed. Settings in the conversion setup file are applied for tables marked as IN.

-u Adds new structures as new tables in the system catalog. Tables that are already part of the system catalog are *not* updated. (User and group files are not affected unless this option is used in conjunction with **-p**.) This is the default if you generate to a directory that already contains a system catalog.

Conversion setup file settings are ignored if you use this option.

-r Specifies the location and name of the repository main file and repository text file. Use the following syntax:

`-r` *repository_main_file repository_text_file*

This option overrides RPSMFIL, RPSTFIL, and RPSDAT settings.

-d Specifies the directory where the system catalog will be created. Use the following syntax:

```
-d target_directory
```

If you don't use this option to specify a location, the system catalog will be created in the working directory.

-p When used with **-i**, this option creates user and group system catalog files, if they don't already exist, initializing them with default values. (If user and group files already exist, **-i** will prevent **-p** from initializing them.) When used without **-i**, this option initializes existing users and groups, but won't create user and group files if they don't already exist.

The **-p** option does not affect system catalog files other than the user and group files, and it may be used in conjunction with the **-c**, **-x**, **-u**, and **-n** options (in addition to **-i**). It can also be used alone after other system catalog files have been created, but **dbcreate** will attempt to regenerate the system catalog.

For more information, see "Initializing users and groups" on page 6-13.

-n Prevents changes to existing users and groups, but enables **dbcreate** to create user and group system catalog files if they don't already exist. This option is meaningful only when used with the **-p** option.

-i Specifies a conversion setup file or specifies that no conversion setup file is to be used. Use the following syntax:

```
-i [conversion_setup_file]
```

where *conversion_setup_file* (optional) specifies the name and, optionally, the path of the conversion setup file. If you use the **-i** option without specifying a *conversion_setup_file*, **dbcreate** does *not* use a conversion setup file—even if the SODBC_CNVFIL environment variable is set. If you specify a filename without specifying a path, **dbcreate** looks for the file in the current working directory. This option overrides the SODBC_CNVFIL setting.

For client/server configurations, the conversion setup file must be on the server.

-l Creates a log file to record the messages generated by the **dbcreate** command. Use the following syntax:

```
-l log_file
```

where *log_file* specifies the path and filename for the log file. If you don't specify a path, the file is saved to the current working directory. If the **-l** option is not used, messages are printed to the screen rather than a log file.

If you use the **-l** option without specifying the **-v** option, **dbcreate** logs only cautions and errors. For structure and other pertinent information, use the **-v** option with the **-l** option.

Do not use an environment variable in the path specification for the log file. If you do, no log file will be generated. Also, you must include a filename extension. If you don't, you'll get an error and the system catalog won't be generated.

-v Creates a more detailed log than the **-l** option alone. (The log will include informational messages). When used in conjunction with the **-l** option, writes the log to a log file. (When used *without* the **-l** option, the log is written to the screen.) This option can be used with **-c**, **-u**, or **-x**. (Note that if you use **-v** with **-x** or **-u** and a conversion setup file, the log incorrectly indicates that conversion setup file settings are applied. Instead, none are applied if used with **-u**. If used with **-x,** only settings for tables marked as IN are applied.)

The syntax is the same on Windows, UNIX, and OpenVMS systems. The only difference is the way paths are specified. Paths must conform to the platform's standard (for example, c:\my_data on Windows systems versus /usr/my_data on UNIX systems). Note the following:

- On OpenVMS, the **dbcreate** utility is set up as a verb. So if you pass more than eight parameters to **dbcreate**, you must enclose the parameters in quotes. Each option counts for one parameter, and each path specification counts for a parameter. The following command, for example, has nine parameters, so they must be enclosed in quotes:

  ```
  $ DBCREATE "-C -R DATA:RPSMAIN DATA:RPSTEXT -P -D CAT: -l LG.TXT"
  ```

- If you use **dbcreate** without the **-r** option, **dbcreate** will use the RPSMFIL, RPSTFIL, and RPSDAT environment variables to locate the repository main file and the repository text file.
- You don't have to specify **-c**, **-x**, or **-u**, but if you do, you must specify only one of these options. If you don't specify any of them, **dbcreate** uses the default option, which is **-u** if the system catalog exists or **-c** if the system catalog doesn't already exist.

When you first use **dbcreate** to generate a new system catalog from your repository definitions, you must add the **-p** option to initialize users and groups. This command creates user and group files (**sodbc_users.*** and **sodbc_groups.***) along with other system catalog files. Without these files there is no user or password validation when connecting to the database, and all connected users have read-only access to all tables. Be sure to generate these files and keep them with the other system catalog files.

Examples

For the following examples, assume your repository files are named **rpsmain** and **rpstext** and are located in a directory defined by the environment variable DATA. In addition, assume that you want the system catalog files created in a directory defined by the environment variable TARGET. The first example works on Windows and UNIX; the second works on OpenVMS:

```
dbcreate -c -r DATA:rpsmain DATA:rpstext -p -d TARGET:
```

```
$ DBCREATE -C -R DATA:RPSMAIN DATA:RPSTEXT -P -D TARGET:
```

Using DBA to generate a system catalog

This section describes how to generate a system catalog from the DBA program. We recommend, however, that you generate system catalogs from the command line using the **dbcreate** utility. The **dbcreate** utility enables you to see messages that document the generation process.

For client/server configurations, DBA must be run from the server.

1. Start DBA by opening Windows Control Panel, selecting Synergy Control Panel, and clicking "xfODBC DBA". Or start it by the following at a Windows or UNIX prompt:

   ```
   dbr SODBC_DBA:xfdba.dbr
   ```

   To open DBA from an OpenVMS prompt, type the following:

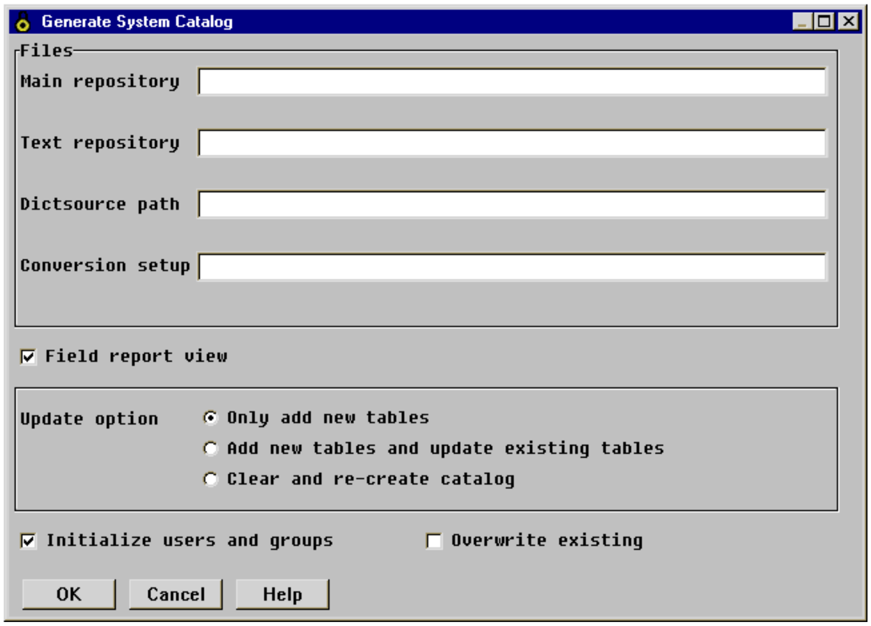
   ```
   $ RUN SODBC_DBA:XFDBA.EXE
   ```

2. From the Catalog menu, select Generate. (For information on using the menus and windows in DBA, see "Understanding DBA, the Customization Program" on page 6-2.)

   If you have not yet opened a system catalog, a window is displayed with the following message:

   No system catalog connected.

3. Click OK or press ENTER. The Generate System Catalog window is displayed. (See figure 4-1.)



Generate System Catalog

Files

Main repository	
Text repository	
Dictsource path	
Conversion setup	

[✓] Field report view

Update option: [✓ Only add new tables] / Add new tables and update existing tables / Clear and re-create catalog

[✓] Initialize users and groups   [ ] Overwrite existing

Figure 4-1. The Generate System Catalog window.

4. Fill in the fields in the Generate System Catalog window with the appropriate file locations and filenames, and select the appropriate options:

   Main repository. Enter the name and location of your repository main file (for example, **rpsmain.ism**). If the RPSMFIL environment variable is set, this field is automatically populated with the RPSMFIL setting (a path and filename). If RPSMFIL is not set, but RPSDAT is, this field is automatically populated with the RPSDAT setting (a path) and the **rpsmain.ism** filename.

   Text repository. Enter the name and location of your repository text file (for example, **rpstext.ism**). If the RPSTFIL environment variable is set, this field is automatically populated with the RPSTFIL setting (a path and filename). If RPSTFIL is not set, but RPSDAT is, this field is automatically populated with the RPSDAT setting (a path) and the **rpstext.ism** filename.

   Dictsource path. Enter the directory in which you'd like to place the system catalog files. If you're regenerating a system catalog and you used the **-c** command-line option when starting DBA, this field is filled with the path specified in the dictsource line of the connect file.

   Conversion setup. If you're regenerating an existing system catalog and you want to use a conversion setup file as input, enter the path and filename of the conversion setup file here. If you've set the SODBC_CNVFIL environment variable, DBA automatically fills this field with the setting for this variable. If you have not set this variable and you don't enter the filename in this field, DBA will not use a conversion setup file. Note that for client/server configurations, the conversion setup file must be on the server.

   Field report view. Clear this option if you want DBA to ignore repository report view flag settings and include all fields—both viewable and non-viewable. Select this option if you want DBA to omit fields with report view flag settings of non-viewable. Note, however, that fields defined as keys or tags in Repository are *always* included in the system catalog.

   Update option. If you're generating a new system catalog, leave the default option, Clear and re-create catalog, selected. (This isn't the default if you opened a system catalog before step 2.)

   - To update a system catalog by generating only tables that do not already exist, select the first option, "Only add new tables." Existing system catalog files are not changed. (This is equivalent to using the **-u** option when using **dbcreate** from the command line.)
   - To update a system catalog based on changes to the repository, select the second option, "Add new tables and update existing tables." This option overwrites existing tables in the system catalog, and it adds any new table definitions. By itself, it does *not* create or overwrite user or group files. If you use a conversion setup file, tables marked as OUT are *not* overwritten or removed. (This is equivalent to using the **-x** option when using **dbcreate** from the command line.)
   - To clear the system catalog and then regenerate it, select the "Clear and re-create catalog" option. This option creates tables in the system catalog as they are defined in the repository. If you use a conversion setup file, tables marked OUT are omitted from the system catalog. (This is equivalent to using the **-c** option when using **dbcreate** from the command line.)

Initialize users and groups. Select this option to create or restore the default set of users and groups. If you select this option *without* selecting the Overwrite existing option, DBA creates the default set of users and groups only if the system catalog has no users and groups. However, if you select this option *and* the Overwrite existing option, DBA creates the default users and groups, even if the system catalog already has users and groups. In this case, DBA regenerates the **sodbc_users** and **sodbc_groups** files in the directory specified as the dictsource path in the connect file, and all changes you've made to users and groups are lost.

When you first generate a new system catalog from your repository definitions, you must use the "Initialize users and groups" option, which creates user and group files (**sodbc_users.*** and **sodbc_groups.***). Without these files there is no user or password validation when connecting to the database, and all connected users have read-only access to all tables. Be sure to generate these files and keep them with the other system catalog files.

For information on the default set of users and groups, see "Initializing users and groups" on page 6-13.

Overwrite existing. Select this option to overwrite existing users and groups. This option is only available when the Initialize users and groups option is selected. If you select this option, all changes you've made to users and groups are lost.

5. Click OK or press ENTER in the Generate System Catalog window to start the generation.

   Unless an error occurs, an information window will open and display the message "System catalog generated." If there's an error, a log file (**ConvErrs.log**) will be created in the connect\synodbc directory (for Windows versions prior to Vista) or in your TEMP directory (for Vista and higher versions of Windows).

6. Make sure the system catalog has a table for each structure you want included. If any are missing, see "Errors and Troubleshooting" on page 4-14.

Regenerating the System Catalog

Typically, you will need to regenerate the system catalog if you have

- manually made changes to the conversion setup file.
- changed the location of the Synergy data tables.
- used S/DE Repository to make changes to the repository files.

You may also consider regenerating if you are distributing several system catalogs all based on the same repository definitions. For more information, see "Handling a repository shared by multiple databases" on page 3-7.

You do *not* need to regenerate for changes made to tables with DBA. These changes automatically update the relevant catalog files. Rather, regenerating the system catalog may *reverse* these changes if you're not careful choosing generation options and fail to use a conversion setup file when necessary.

Regenerating the system catalog with dbcreate

To regenerate the system catalog with **dbcreate**, use the syntax and options listed in "Generating a system catalog from the command line" on page 4-3. Use the same repository files you used to generate the system catalog, and make sure the target directory is the directory that contains the existing system catalog files. By default, **dbcreate** uses the **-u** option, but there are other options that control the ways tables are updated, enable you to initialize users and groups, and enable you to use a conversion setup file. See "Examples" on page 4-11.

Controlling updates to tables

When you regenerate a system catalog from the command line, there are three options for updating tables. The first two are

- the **-c** option—use this option to *clear* the tables in the system catalog and then regenerate it.
- the **-x** option—use this option to *overwrite* the tables in the system catalog based on changes to the repository.

The difference between **-c** and **-x** is that **-c** creates the system catalog from scratch. It clears the system catalog and then generates a new one from the repository files. If a table in the existing system catalog is not part of the repository, it won't be part of the new system catalog. In addition, if you use a conversion setup file, all tables marked as OUT are omitted. On the other hand, **-x** overwrites existing table definitions in the system catalog files *only* when a change has been made to a table in the repository. If tables have been added to the repository, they are added to the system catalog. If you use a conversion setup file, tables marked as OUT are not overwritten or removed.

The third option is **-u**. This is the default setting when you regenerate a system catalog. If you explicitly specify the **-u** option, or if you type **dbcreate** without either the **-x** or the **-c** options, the system catalog will be updated, but only *new* tables will be added. Tables that already exist will not be changed or removed.

If you regenerate the system catalog after making changes to the repository files, we suggest you use the **-c** option, which clears the tables in the system catalog before regenerating it. Other options may not fully update your the system catalog contents.

Initializing users and groups with dbcreate

Users and groups are not automatically initialized when you regenerate a system catalog. To initialize them as you regenerate, use the **-p** option for **dbcreate**. If no users or groups exist for the system catalog, use **-p** and **-n**. (The **-n** option instructs **dbcreate** to create user and group system catalog files if there are none. if there are user and group system catalog files, it instructs **dbcreate** not to make changes to them.) See "Generating a system catalog from the command line" on page 4-3 for more information on these options.

Initializing users and groups removes users and groups you've added, removes modifications you've made to users and groups, and restores users and groups to their default settings.

For more information, see "Initializing users and groups" on page 6-13.

Using a conversion setup file

Note that both the **-i** option and the SODBC_CNVFIL environment variable affect the way **dbcreate** uses conversion setup files.

- If you set the SODBC_CNVFIL environment variable, **dbcreate** automatically uses the conversion setup file that this variable is set to. If SODBC_CNVFIL is not set, you must use the **-i** option (with a filename) to use a conversion setup file. For information on the SODBC_CNVFIL environment variable, see "Specifying a conversion setup file" on page 3-26.
- If you use the **-i** option without specifying a filename after the **-i**, **dbcreate** does not use a conversion setup file—even if the SODBC_CNVFIL environment variable is set.
- The **-i** option overrides the SODBC_CNVFIL setting.

For information on creating conversion setup files, see "Generating and Editing a Conversion Setup File" on page 6-27.

Examples

For the following examples, assume your repository files are named **rpsmain** and **rpstext** and are located in a directory defined by the environment variable DATA. In addition, assume that you want the system catalog files created in a directory defined by the environment variable TARGET.

Adding a new table definition to an existing system catalog

To add a new table definition to an existing system catalog, create the new structure definition in Repository; then enter one of the following at the command line:

- On Windows or UNIX:

```
dbcreate -u -r DATA:rpsmain DATA:rpstext -d TARGET:
```

- On OpenVMS:

```
$ DBCREATE -U -R DATA:RPSMAIN DATA:RPSTEXT -D TARGET:
```

Changing an existing table definition in a system catalog

To change an existing table definition in a system catalog, change the existing structure definition in Repository; then enter one of the following at the command line:

- On Windows or UNIX:

```
dbcreate -x -r DATA:rpsmain DATA:rpstext -d TARGET:
```

- On OpenVMS:

```
$ DBCREATE -X -R DATA:RPSMAIN DATA:RPSTEXT -D TARGET:
```

Creating the default user and group files in a specified directory

To create the default user and group files in a specified directory, enter one of the following at the command line:

- On Windows or UNIX:

```
dbcreate -r DATA:rpsmain DATA:rpstext -p -d TARGET:
```

- On OpenVMS:

```
$ DBCREATE -R DATA:RPSMAIN DATA:RPSTEXT -P -D TARGET:
```

Creating the default user and group files if they don’t already exist

To create default user and group files in current directory—but only if they don’t already exist, enter one of the following at the command line:

- On Windows or UNIX:

```
dbcreate -r DATA:rpsmain DATA:rpstext -p -n
```

- On OpenVMS:

```
$ DBCREATE -R DATA:RPSMAIN DATA:RPSTEXT -P -N
```

Removing a single table from a system catalog

To remove a single table from a system catalog, remove the structure definition in Repository; then enter one of the following at the command line:

- On Windows or UNIX:

```
dbcreate -c -r DATA:rpsmain DATA:rpstext -d TARGET:
```

- On OpenVMS:

```
$ DBCREATE -C -R DATA:RPSMAIN DATA:RPSTEXT -D TARGET:
```

Regenerating the system catalog with DBA

Although this section describes how to regenerate a system catalog with DBA, we recommend that you use **dbcreate** instead. The **dbcreate** utility enables you to see messages that document the generation process.

1. Start DBA from either Synergy Control Panel (in Windows Control Panel) or the command line. See "Command line options" on page 6-2.
2. Open the system catalog in DBA. See "Opening the System Catalog in DBA" on page 6-10.
3. From the Catalog menu, select Generate.
4. If necessary, change any of the default settings in the Generate System Catalog window. If you want to use a conversion setup file, for example, make sure the Conversion setup field is filled. If you've set the SODBC_CNVFIL environment variable, this field is automatically filled with this variable's setting. For information on options in the Generate System Catalog window, see step 4 on page 4-7.
5. Click OK or press ENTER.
6. When the process is complete, press ENTER.

   Unless an error occurs, an information window will open displaying the message "System catalog generated." If there's an error, a log file (**ConvErrs.log**) is created in the connect\synodbc directory (for Windows versions prior to Vista) or in your TEMP directory (for Vista and higher versions of Windows).

Preserving views

To preserve views in a system catalog when you regenerate, do one of the following:

- Use the **-u** or **-x** options (rather than **-c**) when you regenerate the system catalog with **dbcreate**.
- Use the “Only add new tables” option or the “Add new tables and update existing tables” option when using the DBA program.
- Write a program that saves view information, and run that program before you regenerate your catalog. Then, after the catalog has been regenerated, use the output from the program to re-create the views. The Connectivity Series distribution includes an example SQL Connection program, **exam_saveviews.dbl** (in the connect\synsqlx directory), that illustrates this.

Errors and Troubleshooting

Troubleshooting

If you are having difficulty generating a system catalog from your Synergy database, the problem may lie in the repository files themselves or possibly in the generation process. In addition to this section, you may want to read, "If the system catalog won't open..." on page 6-11 and "Verifying the System Catalog" on page 6-31.

If you're encountering problems when you attempt to access data with an ODBC-enabled application, see "Troubleshooting Data Access" on page 9-4. For information on troubleshooting system catalog caching, see "Troubleshooting system catalog caching" on page 8-22.

Check your repository files

To find problems (such as missing keys) in the repository files themselves, you may need to look at them using S/DE Repository. For information on how repository files should be set up and checked, see "Setting Up a Repository" on page 3-2.

Check file locations

- Make sure your repository files are in the location you expect.
- Is RPSMFIL set in the environment? Does it point to the directory where your repository main file is located?
- Is RPSTFIL set in the environment? Does it point to the directory where your repository text file is located?
- If you are using repository files other than those indicated by the RPSMFIL and RPSTFIL environment variables, have you specified their filenames and paths at the command line or in DBA?

Check the conversion setup file

If you have created a conversion setup file and are using it as input for the creation of the system catalog,

- is it located in the directory indicated by the SODBC_CNVFIL environment variable, and is SODBC_CNVFIL set in the environment?
- are all of the IN|OUT arguments set to include the tables you want in the system catalog?
- does it include environment variables indicating table locations? Are these variables set in the environment?

If you are using **dbcreate** to generate the system catalog and you want to input a conversion setup file other than the one specified by the SODBC_CNVFIL environment variable,

- did you indicate the filename of the new conversion setup file as an argument?

Setting variables for generation options

- To ensure that your data is read correctly by an ODBC-enabled application, have you set the necessary variables for generation options before attempting to generate the system catalog? These variables may include SODBC_CNVFIL, SODBC_CNVOPT, SODBC_COLLAPSE, SODBC_INIFIL, SODBC_ODBCNAME, and SODBC_TMPOPT.
- See "System Catalog Generation Issues" on page 3-12 for information on potential data conversion issues.

Dbcreate error and warning messages

This section lists errors and warnings you may encounter when using the **dbcreate** utility. For information on DBA errors, see "DBA error messages" on page 6-7. For information on data access errors, see "Data Access Errors" on page 11-11.

ASCII Structure *structure_name*: no keys found

ASCII files do not have keys. This is an informational message, not a warning or error.

Cannot open logfile *log_file*

The specified log file name or path cannot be used. Check the filename, path, file permissions, and available disk space.

Column *column_name* maximum name length exceeded

The name *column_name* has exceeded 30 characters and has been truncated. This is most likely to occur with groups, struct fields, and arrays. Use the group prefix or Alternate name to create shorter names.

Column *column_name*, unsupported date type: *coltype*

The column uses an unsupported date format mask (*YYPP*, *YYYYPP*, or *RRPP*). Define the field as user-defined in the repository and modify **xfodbcusr.dll** for Windows, **XFODBCUSR.so** for UNIX or **xfodbcusr_so.exe** for OpenVMS to properly handle columns with these data types. See chapter 7, "Creating Routines for User-Defined Data Types," for more information.

Caution: Table *table_name*, key *key_name*, literal *literal_name* value does not match table tag value, relation dropped

A relation has a literal segment whose value does not match the tag criteria for the related segment. Change the literal segment to match tag criteria.

DDINFO-ERR: Cannot open *filename*

The specified file could not be found.

DDINFO-ERR: ddc_keyx called with a null

This is an internal error. Call Synergy/DE Developer Support.

DDINFO-ERR: ddc_keyx dc_io() failure

This is an internal error. Call Synergy/DE Developer Support.

DDINFO-ERR: Error opening conversion setup file *filename*

The specified conversion setup file cannot be found.

DDINFO-ERR: Field *field_name* exceeds maximum 2000: *length*

The length (*length*) of field *field_name* exceeds the maximum allowable length of 2000 bytes. Shorten the field name.

DDINFO-ERR: Field *field_name* is used as a key segment

The arrayed field *field_name* is a key segment in the repository. This is not supported. To work around this, create an overlay that consists of the entire array.

DDINFO-ERR: No fields defined for structure: *structure_name*

The structure *structure_name* does not contain any field definitions. Check your repository.

DDINFO-ERR: No keys defined for structure: *structure_name*

Each structure used to generate the system catalog must have at least one key, but the structure *structure_name* has no key. Define a key for the structure.

DDINFO-ERR: No repository or schema file found

This is an internal error. Call Synergy/DE Developer Support.

DDINFO-ERR: version mismatch: cannot convert version *repository_version*

The repository was created with an unsupported version of Synergy/DE Repository. *xf*ODBC supports repositories created with Repository version 7.*x* or greater. Re-create the repository.

DDINFO-WARN: Field *field_name* exceeds maximum 2000 - MSACCESS may fail.

The field *field_name* is too long and may cause Microsoft Access to fail.

DDINFO-WARN: Field *field_name* Precision changed to *precision* based on format

The repository format string for field *field_name* caused **dbcreate** to assign a different precision for the field in the system catalog.

DDINFO-WARN: Field *field_name* (odbc field *alt_name*) not found

Dbcreate could not find the field *field_name* (with the alternate name of *alt_name*).

DDINFO-WARN: Overlay Key *key_name* field *field_name* has an overlay offset - GENIX overlay key not created

An index was not created for *field_name* because **dbcreate** does not create an index for an overlay field that has an offset. See "Indexes for overlay segments" on page 10-3.

DDINFO-WARN: Key *key_name* segment *seg_name* length is less than total length of overlaid fields

The key segment *seg_name* is an overlay field whose length is less than the total length of the fields it overlays. **Dbcreate** was probably not able to use the segment as defined. This generally indicates that there is a problem with the repository or file definition because a key should never partially encompass an overlaid field. If **dbcreate** was not able to use the segment, it and subsequent segments are omitted from the index.

DDINFO-WARN: Key *key_name* segment *seg_name* is an overlay containing field *field_name* which exceeds key length.

The key segment *seg_name* overlays a field (*field_name*) that exceeds the key length. **Dbcreate** was probably not able to use the segment as defined. This generally indicates that there is a problem with the repository or file definition because a key should never partially encompass an overlaid field. If **dbcreate** was not able to use the segment, it and subsequent segments are omitted from the index.

DDINFO-WARN: *key_name* not equal to length of all segments.

The length of the key *key_name* is not equal to the length of all of its segments. **Dbcreate** may have been able to use the segments as defined, but this generally indicates that there is a problem with the repository or file definition because a key should never partially encompass an overlaid field. If **dbcreate** was not able to use a segment, that and subsequent segments are omitted from the index.

DDINFO-WARN: *seg_name* exceeds field *field_name* length. Segment ignored.

The length of the segment *seg_name* exceeds the length of the field (*field_name*) for the key. **Dbcreate** was probably not able to use the segment as defined. This generally indicates that there is a problem with the repository or file definition because a key should never partially encompass an overlaid field. If **dbcreate** was not able to use the segment, it and subsequent segments are omitted from the index.

DDINFO-WARN: Overlay exceeds *key_name* length. Segment *seg_name* ignored.

Segment *seg_name* is an overlay field whose fields exceed the length of the key (*key_name*). **Dbcreate** was probably not able to use the segment as defined. This generally indicates that there is a problem with the repository or file definition because a key should never partially encompass an overlaid field. If **dbcreate** was not able to use the segment, it and subsequent segments are omitted from the index.

DDINFO-WARN: Overlay not contiguous

A key overlays fields that are not contiguous, so **dbcreate** is not able to use this key to create an index.

Duplicate column *column_name*

Either the same alternate name has been assigned to more than one field, or truncated names are the same.

- Alternate names must be unique.
- Column names are truncated if they exceed 30 characters. If you have two column names whose first 30 characters are identical, *xf*ODBC will see them as identical names. This is most likely to occur with groups, struct fields, or arrays. To see if this is the case, use the **-v** option with **dbcreate**. If truncated column names are the cause, use the group prefix or Alternate name to create shorter names.

If neither of these is the cause, call Synergy/DE Developer Support.

ERROR: Cannot open files (ddc_init:*data*)

An error occurred while opening the repository files. Call Synergy/DE Developer Support.

ERROR: GENESIS_FORKEYS entry, *primary_table, primary_key, key, primary_column, column, foreign_table*

There are duplicate relations for the table *primary_table*.

ERROR: GENESIS_HOME environment variable not found

The GENESIS_HOME environment variable is not set. See "Specifying the connect file location (GENESIS_HOME)" on page 3-19 for information on setting this variable.

ERROR: Index *index_name*, column *column_name* not found

The given index references a field that is not defined. Often, this is caused by a field that is incorrectly defined as invisible. If not, call Synergy/DE Developer Support.

ERROR: sdms_init failed

An internal function failed. Call Synergy/DE Developer Support.

ERROR: *table_name*: SDMS error

A Synergy DBMS error occurred while accessing the table. Notify your system administrator.

ERROR: Table lookup error: (ddc_fname: *data*)

An error occurred while retrieving the list of table definitions. Use the DBA program to compare the system catalog to the repository definitions. For information, see "Comparing the system catalog to repository definitions" on page 6-31.

Index *index_name*, column *column_name* datatype mismatch

The data type of the index segment does not match the one defined in the corresponding column definition. Verify your repository definition for this index and its segments.

Index *index_name*, column *column_name* decimal datatype mismatch

A column is defined as type decimal but is defined as another data type when specified as a segment of the key. If the key value can be negative, change the file key type. If the key cannot be negative, change repository definition of the key segment to an alpha data type. See "Optimizing with Keys" on page 10-2.

Index *index_name*, column *column_name* not found

The index references a field that is not defined. Verify the repository.

Index *index_name*, column *column_name* not used due to date windowing

An ordered key could not be created because the date column contains a two-digit century.

Index *index_name*, column *column_name* not used due to unordered date

The column *column_name* uses a user-defined date type (one specified with ^CLASS^) that does not start with the year. See chapter 7, "Creating Routines for User-Defined Data Types," for information on creating routines to handle the date.

Index *index_name*, column *column_name*, unsupported segment type *seg_type*

The segment data type for column *column_name* in index *index_name* is not supported. Subsequent segment definitions for this index will be ignored. Verify your repository definition for this index and its segments. Unsupported data types include external, user-defined, case-insensitive, and unsigned integer. The literal data type is supported for foreign keys only, and then only when specified as the first segment.

Index *index_name* dropped

The given index definition was dropped because the first segment has an unsupported segment type. Verify your repository definition for this index and its segments.

Index *index_name*, duplicate column *column_name*

There is a duplicate column name in the specified index. Check the repository definition for the specified index.

Index *index_name*, overlay index created

A segment within a key is an overlay. *xf*ODBC will optimize by creating another key with all the columns, including the columns defined by the overlay.

Invalid option Expected LOGFIL parameter

You must specify a log file when you use the **dbcreate -l** option. Use the following syntax:

`-l` *log_file*

where *log_file* specifies the path and filename for the log file.

Invalid option Expected PATH parameter

You must specify a path when you use the dbcreate **-d** option. Use the following syntax:

`-d` *target_directory*

where *target_directory* is the directory where the system catalog files will be created.

Invisible field *field_name* ignored

The invisible field definition *field_name* is ignored. This is an informational message.

Maximum command line length (*max_len*) exceeded

There are too many characters in the fields of the Generate System Catalog window. The DBA program resolves any environment variables, combines the fields in this window into a single command line, and sends the command line to **dbcreate**.

Message file not available. ERRNO: *number*

*xf*ODBC cannot find the error message file. (By default, the error message file is named **sql.msg**.) Set the GENESIS_MSG_FILE environment variable to the path and filename of the error message file. Alternatively, you can put the **sql.msg** file in a directory named lib that's in the directory GENESIS_HOME is set to (in other words, GENESIS_HOME:lib).

No file link found for related structure

One of the structures for a relation has not been assigned to a file.

No indexes defined for table *table_name*

Index definitions for *table_name* are invalid. Verify the repository and use **fcompare** to compare the repository definitions to your Synergy database files. See "Validate, verify, and compare" on page 3-7 for information.

Null key '*index_name*' ignored

Null keys are ignored.

Null key *index_name*, optimization reduced

Null keys can be used only for equality operations. Whenever possible, declare keys as unique with no duplicates, non-null values, or negative values. This enables *xf*ODBC to optimize ORDER BY statements.

Partial catalog exists, specify -c option

The catalog generation failed and the system catalog is not complete. Regenerate the system catalog using **dbcreate** with the **-c** option.

Primary key *from_table.key.column*(*ordinal_position*) is smaller than foreign key *to_table.key.column*(*ordinal_position*)

The primary key segment size is smaller than the foreign key it references. Verify the repository definitions for the primary and foreign keys to ensure the column counts match.

Primary key *table_name.key_name*, Foreign key *table_name.key_name*, column count mismatch*[.* Relation dropped*.]*

The number of segments in the primary key does not match the number of columns in the foreign key. Verify the repository definition for both the primary and foreign key. If the first segment is a literal and there is not more than one segment in the foreign key table, the relation will be dropped.

Primary key index *index_name* not found

A primary key is missing. *xf*ODBC may have found an error when attempting to create the key. Check the repository definition for *index_name* and/or regenerate the system catalog with the verbose (**-v**) and log file (**-l** *log_file*) options to determine if there is a problem with *index_name*.

Structure *structure_name*: No fields found

No field definitions were found for the structure *structure_name*. Verify the repository and use **fcompare** to compare the repository definitions to your Synergy database files. See "Validate, verify, and compare" on page 3-7 for information.

Structure *structure_name*: No keys found

No keys were found for the structure *structure_name*. Verify the repository and use **fcompare** to compare the repository definitions to your Synergy database files. See "Validate, verify, and compare" on page 3-7 for information.

Structure *structure_name*: Not found

The structure *structure_name* could not be found. Verify the repository and use **fcompare** to compare the repository definitions to your Synergy database files. See "Validate, verify, and compare" on page 3-7 for information.

Structure *structure_name*: Unknown error

An unknown error was returned for the given structure. Verify the repository and use **fcompare** to compare the repository definitions to your Synergy database files. See "Validate, verify, and compare" on page 3-7 for information.

Table *table_name* is already defined

A duplicate table name was found when updating the system catalog. Regenerate the system catalog using **dbcreate** with the **-c** option.

Table *table_name*, column *column_name* not found

The column (*column_name*) was not found in the table (*table_name*). Check the repository field attributes for *table_name*.

Table *table_name*, primary index *index_name*, column *column_name* not found

An index definition references a column that does not exist. Verify the index and column definition using repository and/or regenerate the system catalog using **dbcreate** with the logging (**-l** *log_file*) and verbose (**-v**) options.

5

Setting Up a Connect File

For each system catalog you create, you must create a unique connect file. You'll need a connect file to open a system catalog in the *xf*ODBC Database Administrator (DBA) program, and you'll need a connect file to access data. The connect file contains information on the location of the Synergy data files and the system catalog. You can also use it to set the convert_error option, Synergy driver logging, and environment variables for the *xf*ODBC driver. You can create connect files in Windows, UNIX, and OpenVMS environments.

Creating the Connect File

Use a text editor to create a connect file, and then save the file in your GENESIS_HOME directory with whatever name you choose. Connect files should reside in the directory specified by the GENESIS_HOME environment variable even if the data and system catalog files are located in other directories. (See "Specifying the connect file location (GENESIS_HOME)" on page 3-19.)

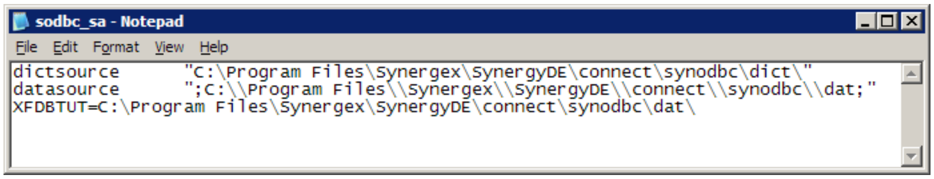
Connect files must contain

- a dictsource line, which specifies the directory that contains, or will contain, your system catalog.
- a datasource line, which specifies the directory or directories that contain your Synergy data files.

Connect files can also contain

- the convert_error setting. (See "Setting the convert_error Option" on page 5-4.)
- commands that invoke Synergy driver logging for system catalog caching. (See "Synergy Driver Logging" on page 5-5.)
- environment variable settings. (See "Setting environment variables in a connect file" on page 3-33.)

Figure 5-1 shows the connect file for the sample database on Windows. The sample connect file and database are included in the Connectivity Series distribution.



```
dictsource      "C:\Program Files\Synergex\SynergyDE\connect\synodbc\dict\"
datasource      ";C:\\Program Files\\Synergex\\SynergyDE\\connect\\synodbc\\dat;"
XFDBTUT=C:\Program Files\Synergex\SynergyDE\connect\synodbc\dat\
```

Figure 5-1. The sample connect file for Windows.

The dictsource and datasource Lines

The *dictsource* line must specify the full path to the directory that contains (or will contain) your system catalog and, optionally, your data files.

The *datasource* line must specify the full path to the directory or directories that contain your Synergy data files. However, it is used only if a filename, but no path, is set in the Open filename field of the Repository file definition and no data files exist in the dictsource directory. For example, if you specified only a filename, such as **mydata.ism**, in the Open filename field, and there were no data files in the dictsource directory, the datasource specification will by used to locate **mydata.ism**. Note that this line is required even if it isn't used.

We don't recommend using dictsource or datasource lines for the location of data files. Instead, use an environment variable for the path in the Repository Open filename field.

The following is a Windows example. The dictsource specification is straightforward, but notice the format of the datasource line: the path must begin and end with a semicolon, and double slashes are used to represent a single backslash in the path.

```
dictsource "C:\Program Files\Synergex\SynergyDE\connect\synodbc\dict\"
datasource ";C:\\Program Files\\Synergex\\SynergyDE\\connect\\syn-
odbc\\dat;"
```

The syntax for dictsource and datasource in a UNIX or OpenVMS connect file is the same, but paths must conform to the platform's standard.

When using SQL OpenNet, don't use UNC (uniform naming convention) paths or paths that include mapped drives. The service may not have the privilege to map the drive.

The following is a UNIX example:

```
dictsource       /usr/synergyde/connect/synodbc/dict
datasource      ;/usr/synergyde/connect/synodbc/dat;
```

The following is an OpenVMS example:

```
dictsource       DKA600:[SYNERGYDE.CONNECT.SYNODBC.DICT]
datasource      ;DKA600:[SYNERGYDE.CONNECT.SYNODBC.DICT];
```

If the data files are in more than one directory, separate directories with a semicolon. For example:

```
datasource      ;c:\\syndata;c:\\syndata2;c:\\syndata3;
```

If there are spaces in the path, the path (including semicolons for datasource) must be enclosed in double quotes. For example:

```
dictsource    "c:\my files\dict"
datasource    ";c:\\my files\\dat;"
```

Setting the convert_error Option

By default, the *xf*ODBC driver treats invalid dates as null. You can, however, use the convert_error option to instruct the *xf*ODBC driver to allow SELECT statements to fail if they encounter invalid dates rather than treating them as null.

- If you don't set convert_error, or if you set it to no, invalid dates are treated as null data unless they are in user-defined date columns.
- If you set convert_error to yes, SELECT statements will fail if they encounter invalid date data.

To set convert_error, add a line with the following syntax to the end of the connect file:

```
convert_error yes|no
```

For example:

```
dictsource "C:\Program Files\Synergex\SynergyDE\connect\synodbc\dict\"
datasource ";C:\\Program
Files\\Synergex\\SynergyDE\\connect\\synodbc\\dat;"
convert_error yes
```

Note the following:

- Use all lowercase characters for yes and no.
- Do *not* use an equal sign (=) or any other symbol.

Synergy Driver Logging

You can use Synergy driver logging to determine if a system catalog is cached. By adding the following lines to your connect file, *xf*ODBC will log the path and name for the shared memory file and list errors encountered while attempting to use shared memory.

```
logfile file_spec
loglevel 1
```

where *file_spec* is the path and filename of the log file you want to create.

Note that we don't recommend adding these lines before starting the SQL OpenNet server. See "System Catalog Caching" on page 8-18 for more information.

6

Viewing and Customizing the System Catalog

Once you've generated a system catalog and created a connect file, you can view and customize the system catalog. The *xf*ODBC Database Administrator (DBA) program enables you to view and customize users and groups; view tables, columns, indexes, and segments; and delete tables and columns. A conversion setup file enables you to change the access level of a table or add a table back into the system catalog. Additionally, if you use DBA to delete a table from the system catalog, you can use a conversion setup file to preserve that change when you regenerate.

Understanding DBA, the Customization Program

DBA enables you to view and customize elements of the system catalog. This section explains how to use DBA command-line options, menus, windows, and lists. For information on opening a system catalog in DBA, see "Opening the System Catalog in DBA" on page 6-10.

Starting DBA

To run DBA on a UNIX system, you must have configured your session for the Synergy/DE runtime by running **setsde**. For information, see "UNIX Requirements" in the "Requirements and Considerations" chapter of the *Installation Configuration Guide*.

To start DBA, do one of the following:

- In Windows Control Panel, select Synergy Control Panel, and then click "xfODBC DBA."
- Use the following syntax at a Windows or UNIX prompt:

  `dbr SODBC_DBA:xfdba.dbr` *[option] [...]*

- Use the following syntax on an OpenVMS system:

  `XFDBA` *[option] [...]*

Command line options

-c (optional) Specifies the user name, password, and connect file. If you use this option, DBA opens the system catalog specified in the connect file. Use the following syntax:

`-c` *connect_string*

where *connect_string* has the following format:

username/password/connect_filename

For information and examples, see "Opening a system catalog from the command line" on page 6-11.

-g (optional) Generates a conversion setup file. Use the following syntax:

`-g` *conversion_setup_file*

where *conversion_setup_file* specifies the path and filename of the generated file. This setting overrides the SODBC_CNVFIL environment variable setting. Note that for client/server configurations, the conversion setup file must be on the server.

-i (optional) Initializes existing user and group files to default values. For information and examples, see "Initializing users and groups" on page 6-13.

-v (optional) Creates a verification log that lists each table and compares stored counts with actual record counts. Use the following syntax:

`-v` *verification_log*

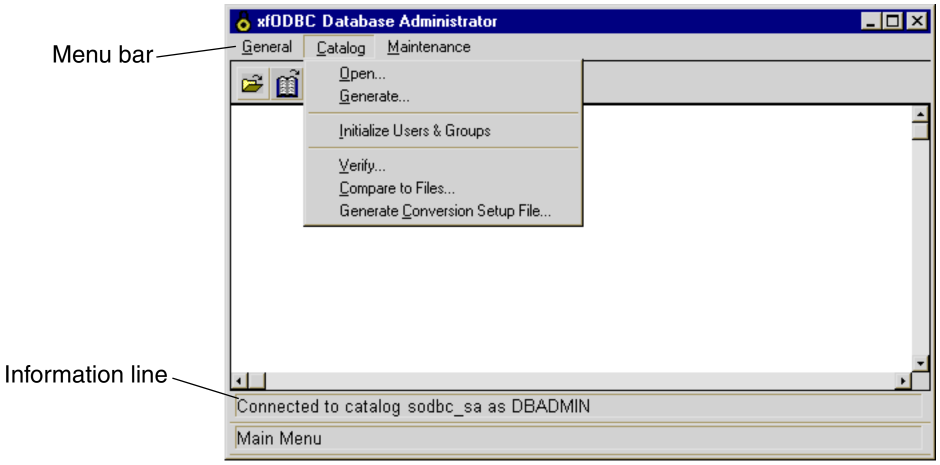
where *verification_log* specifies the path and filename of the generated log file. If you don't specify a path, the verification log is saved to the current working directory.

Note the following:

- Paths specified with DBA command-line options must conform to the platform's standard (for example, c:\my_data on Windows systems versus /usr/my_data on UNIX systems).
- The **-c** option can be used by itself, but all other DBA command-line options must be used in conjunction with the **-c** option.

DBA menus and windows

When you first enter DBA, the main DBA window is displayed.



Connected to catalog sodbc_sa as DBADMIN

Main Menu

Figure 6-1. The main DBA window.

Activating and deactivating the menu bar

You can select an entry from a menu as long as the menu bar is active. To activate or deactivate the menu bar,

- on Windows, click the menu bar or press ALT.
- on UNIX and OpenVMS, press CTRL+P.

Selecting a menu or menu entry

There are three ways to make a menu selection:

- Arrow keys
- Quick-select characters
- Shortcuts

To use arrow keys and quick-select characters, the menu bar must be active. To use shortcuts, the menu bar does not need to be active; shortcuts bypass the menu.

Use the left and right arrow keys to move across the menu headings in the menu bar. Use the up and down arrow keys to move among menu entries. Press ENTER to select a highlighted entry.

A *quick-select character* is a single character that accesses a menu entry. When a menu is pulled down, you can type the quick-select character to select the menu entry. For example, if you pull down the General menu and then type b, the About window will open.

A *shortcut* is a key or key sequence associated with a menu entry—for example, F4 for Exit. You can use a shortcut anytime the associated entry is valid; the menu does not have to be active. Shortcuts are different for each type of terminal and can be reassigned by your system manager. To view the shortcuts for your particular terminal, pull down any menu. Shortcuts are listed to the right of menu entries.

Skipping a field

If a field is optional and does not have a default value, you can leave the field blank by pressing ENTER. If an optional field has a default value, you can clear the field by pressing the spacebar or the BACKSPACE key. Then press ENTER to move to the next field.

Moving between fields

On Windows, press SHIFT+TAB to move to the previous field in the current input window, and press TAB to move to the next field in the current input window.

On UNIX and OpenVMS, use the UP ARROW key or select Previous Field from the Input menu or List menu to return to the previous field in the current input window. If the cursor is positioned on the first field in a window, the cursor moves to the last field in the window. Use the DOWN ARROW key or select Next Field from the Input menu or List menu to move the cursor to the next field in an input window.

Moving the cursor in a line of text

To move the cursor to the left or right within a line of text, use the left or right arrow key.

On UNIX and OpenVMS you can move the cursor by selecting the Move Left or Move Right entry in the Input menu or List menu.

Entering data

After you have finished typing data for a particular field, press ENTER to enter it.

Abandoning your changes

Resetting a field to its original value on UNIX or OpenVMS

To restore the data in a field to its original state on UNIX or OpenVMS (the state it was in before you typed anything), select Reset Field from the Input menu.

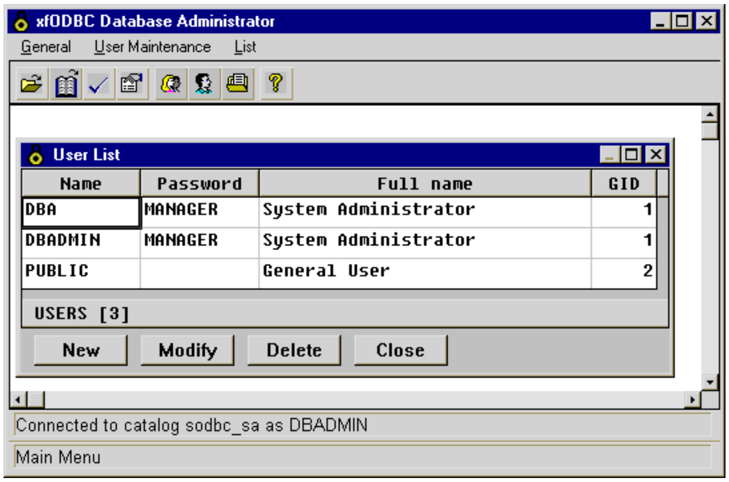
Abandoning changes to all fields

When you select Abandon from the General menu, any data you entered in the current input window is ignored. The original data for that window is restored, and you are returned to the previous window or menu.

On Windows, you can also click the close button to abandon changes for the current input window.

Using lists

To edit a list entry, highlight the entry and press ENTER. If it can be edited, an input window opens.



Name	Password	Full name	GID
DBA	MANAGER	System Administrator	1
DBADMIN	MANAGER	System Administrator	1
PUBLIC		General User	2

USERS [3]

Connected to catalog sodbc_sa as DBADMIN

Main Menu

Figure 6-2. The User List in DBA.

Moving among entries in a list

To...	From the list menu, select...
Move to the previous entry	Previous Entry
Move to the next entry	Next Entry
View the previous page of entries	Previous Page
View the next page of entries	Next Page
Move to the first entry	Top Entry
Move to the last entry	Bottom Entry

Searching the Table List

If you're in the Table List, you can search for an entry by selecting Find from the List menu. An input window is displayed. Enter the name or partial name of the entry you wish to find. DBA highlights the first matching entry in the list.

Viewing hidden areas of the Table List

When you view the Table List, columns or a portion of a column may extend beyond the edge of the Table List window. To view hidden areas of the Table List, select Toggle View from the List menu.

Exiting a list

To exit a list, press F4.

Exiting

Exiting the current function

To save your changes and exit the current input window or list, select Exit from the General menu or use the Exit shortcut (F4). You are returned to the previous window or menu.

To exit an input window without saving any changes, select Abandon from the General menu or use the Abandon shortcut (CTRL+A).

Exiting DBA

There are two ways to exit DBA:

- Exit each input window and list until the main window is displayed. Then, from the General menu, press the Exit shortcut again or select Quit.
- To exit DBA immediately, click the close box (on Windows) or select Quit from the General menu. DBA will quit regardless of the input window or list that's currently open.

DBA error messages

This section lists errors you may encounter when using the *xf*ODBC Database Administrator (DBA) program. For information on errors you may encounter when generating a system catalog, see "Dbcreate error and warning messages" on page 4-15. For information on data access errors, see "Data Access Errors" on page 11-11.

Cannot connect to catalog - *connect_filename*

The specified connect file does not exist in the GENESIS_HOME directory.

- Verify that the GENESIS_HOME environment variable is set correctly. See "Specifying the connect file location (GENESIS_HOME)" on page 3-19 for information.
- Verify that you have placed the connect file *connect_filename* in the GENESIS_HOME directory.
- Verify that you have entered the full filename for the connect file, including filename extension if applicable.

Cannot delete a system catalog table

You cannot delete system catalog tables (GENESIS_*) in DBA.

Cannot delete group: *group_name*

The group you are trying to delete has users assigned to it. Delete the associated users; then delete the group.

Cannot open GENESIS_TABLES.ISM table

The system table file cannot be opened. Verify that the system table files **GENESIS_TABLES.ISM** and **GENESIS_TABLES.IS1** (in Windows and UNIX environments) are in the dictsource directory specified in your connect file. (OpenVMS systems should have the **GENESIS_TABLES.ISM** file. Windows and UNIX systems should have both files.)

Cannot open system catalog in *path*

The path specified on the dictsource line in the connect file is invalid. Verify that the dictsource path contains the system catalog files, **GENESIS_***. If the dictsource path contains an environment variable, verify that this variable is set in the environment.

Dictionary source path not found in the connect file

The path specified by dictsource in the connect file does not exist. Verify that the connect file contains a dictsource specification and that the path specified actually exists. If the dictsource path contains an environment variable, verify that this variable is set in the environment.

Duplicate group ID specified, try again

The group ID you specified already exists. Re-enter group information.

Duplicate user name specified, try again

The combination of user name, password, and group ID you entered for a new user already exists. Re-enter the user information.

ERROR: Cannot open one of the system catalog tables

While verifying the database, DBA could not open one of the system catalog tables. Check that all system catalog files are in the dictsource directory specified in the connect file.

ERROR: Index *index_name* - segment column field *column_name* not found

While verifying the database, DBA could not find a segment in the system column table. Regenerate the system catalog using **dbcreate** or DBA (Catalog > Generate). If the problem persists, you'll need to open your repository and make corrections to your data definitions.

ERROR: Index *index_name* not found

While verifying the database, DBA could not find an index in the system column table. Regenerate the system catalog using **dbcreate** or DBA (Catalog > Generate). If the problem persists, you must open your repository and make corrections to your data definitions.

Error reading initialization file

A file error has occurred while reading the conversion setup file. Open the conversion setup file with a text editor and verify that all data files exist in the specified directories. If environment variables are used, verify that those variables are set in the environment.

Incomplete group record

To complete the group record, fill in a group name and access level.

Incomplete user record

To complete the user record, fill in a user name, password, and group ID.

Invalid group ID specified, try again

The group ID you entered is invalid. For a list of valid group IDs, select User Maintenance > Select Group.

Login failed: invalid group access: *user_ID*

The user does not have the necessary access level to open the system catalog. To use DBA, the user must belong to a group with an access level of 254 or 255. If no such user exists, you may need to re-initialize users and groups for the system catalog you are attempting to open.

Login failed: invalid group ID: *group_ID*

The user's group ID was not found in the system group file. Use a different user to open the system catalog. This indicates that the user or group catalog files have become corrupt. You may need to re-initialize users and groups.

Login failed: invalid password

The password entered does not match the password assigned to the user. Enter the correct password.

Login failed: invalid user name: *user_name*

The user name entered does not match any in the system user table.

Login failed: missing user ID

No user name was entered. Enter a user name in the User name field of the Open System Catalog window, or enter a string with the following format in the Connect file field of the Open System Catalog window: *user_ID/password/connect_filename*.

Login failed: unable to open group file

The system group files cannot be accessed. Verify that the system group files (**sodbc_groups.***) are in the dictsource directory specified in the connect file.

Login failed: unable to open user file

The system user files cannot be accessed. Verify that the system user files (**sodbc_users.***) are in the dictsource directory specified in the connect file.

Missing argument for *option_name* option

The **-c**, **-g**, and **-v** options must be followed by an argument:

`-c` *connect_string*
`-g` *conversion_setup_path&filename*
`-v` *verify_log_path&filename*

No window library file found

Set SODBC_DBA to the directory containing **xfdba.is1** (in Windows and UNIX environments), and **xfdba.ism**. These are the UI Toolkit window library files for the DBA program. The **xfdba.dbr** file must also be in this directory.

Too many arguments for *option* option

The **-c**, **-g**, and **-v** options must have only *one* argument, and the **-i** option must *not* have an argument.

WARNING: the name *name* is one of the SQL reserved words

While verifying the database, DBA has discovered that an SQL reserved word is used for a column names. The SQL parser is cannot execute the SQL statement. Use S/DE Repository to change *name* to one that is not reserved by SQL. For a list of SQL reserved words, see "ODBC Reserved Words" on page B-65.

Opening the System Catalog in DBA

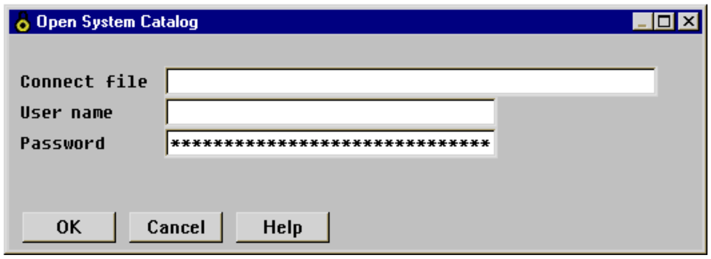
To open a system catalog, you must have a connect file, and the user you log in with must have an access level of 254 or 255. If you initialized users and groups, your system catalog already has two such users: DBADMIN and DBA.

There are two ways to open a system catalog in the DBA program:

- By opening DBA and then selecting Catalog > Open
- By specifying the connect string when you open DBA from the command line

Opening a system catalog from DBA

1. Start DBA, and then from the Catalog menu, select Open.


Open System Catalog

Connect file	
User name	
Password	******************************

Figure 6-3. The Open System Catalog window.

2. In the Open System Catalog window (shown in figure 6-3), fill in the fields individually or enter a connect string in the Connect file field using the following syntax:

   user_name / password / connect_filename.

   Connect file. The name of the connect file for the system catalog. Unless you specify a path, the connect file must be in the GENESIS_HOME directory. Optionally, you can enter the entire connect string in this field.

   User name. A case-sensitive alphanumeric entry stored in the system catalog. The default user names are DBADMIN, DBA, and PUBLIC. By default, DBADMIN and DBA belong to groups with access levels that enable them to open the system catalog in DBA. The PUBLIC user does not.

   Password. A case-sensitive alphanumeric entry stored in the system catalog. The default password for the DBADMIN user and the DBA user is MANAGER.

3. Click OK. If DBA can open the system catalog, the Open System Catalog window will close, and the information line at the bottom of the main DBA window will have the message “Connected to catalog *connect_filename* as *user_name*.”

Opening a system catalog from the command line

You can open DBA and your system catalog in one step from the command line. (Note that you can specify other command-line options after the **-c** option. See "Command line options" on page 6-2.)

- At a Windows or UNIX prompt, use the following syntax:

  ```
  dbr SODBC_DBA:xfdba.dbr -c connect_string
  ```

- At an OpenVMS prompt, use the following syntax:

  ```
  $ XFDBA -C connect_string
  ```

where *connect_string* has the *user_name*/*password*/*connect_filename* format.

The DBA program will open. If DBA can open the system catalog, the information line at the bottom of the main DBA window will have the message "Connected to catalog *connect_filename* as *user_name*." If DBA cannot open the system catalog, no message will appear in the information line.

For example, to open the system catalog for the sample database in a Windows or UNIX environment, enter

```
dbr SODBC_DBA:xfdba.dbr -c DBADMIN/MANAGER/sodbc_sa
```

To open the system catalog for the sample database in an OpenVMS environment, enter

```
$ XFDBA -C DBADMIN/MANAGER/SODBC_SA
```

These examples assume that your connect file is **sodbc_sa**, that SODBC_DBA is set to the directory where the DBA program resides, and that GENESIS_HOME is set to the directory where the connect file resides.

If the system catalog won't open...

If you can't open a system catalog, typically it's because *xf*ODBC can't locate the necessary data files and system catalog files. Make sure the files are stored in the expected directories and that the connect file and environment variables are set correctly.

Check your connect file

- Did you enter the *entire* connect filename, including the file extension (if applicable), when connecting to the system catalog?
- Note the dictsource line in the connect file. Is the specified directory valid? Is the line formatted correctly?
- Did you use environment variables in the Open filename field in Repository? If so, you can define those variables in the connect file.

Verify environment variables

Because environment variables enable *xf*ODBC to locate files, it's important to verify their settings.

- GENESIS_HOME—Is this variable set correctly? For client/server configurations, was it set before the SQL OpenNet server was started? See "Specifying the connect file location (GENESIS_HOME)" on page 3-19 for information.
- SODBC_DBA—Is this variable set correctly? See "Specifying the location of DBA and dbcreate" on page 3-18 for information.

Verify the log-in

- Do you have user and group files?
- When entering the connect string, did you type the user name and password *exactly* as stored? They are case sensitive.

Customizing Users and Groups

Users and groups enable you to control access to your database and your system catalog. If you initialized users and groups, your system catalog already has a default set of users and groups. You can use these as they are, or you can modify them. You can also create your own users and groups.

A user's access level is determined by membership in a group. To create users with varying access levels,

1. Create groups.
2. Define the access levels for each group.
3. Assign users to those groups.

For information on how group access levels and table access levels work together, see "Setting Security Levels" on page 8-2.

When you modify users or groups in DBA, the system catalog is updated automatically. You do not need to regenerate the system catalog.

Initializing users and groups

DBA and **dbcreate** have options that enable you to initialize users and groups—that is, create or return to an initial, default set of users and groups. This default set includes three default users and two default groups. (See the table below.) You can initialize users and groups as you generate or regenerate a system catalog (see "Generating the System Catalog" on page 4-2 and "Initializing users and groups with dbcreate" on page 4-10). And you can initialize users and groups for an existing system catalog without regenerating the system catalog, as described below.

Initializing user and groups removes users and groups you've added, removes modifications you've made, and restores users and groups to their default settings.

User name	Password	Assigned group	Access level
DBA	MANAGER	SYSTEM	254
DBADMIN	MANAGER	SYSTEM	254
PUBLIC	*No password*	USER	100

For information on access levels, see "Setting Security Levels" on page 8-2.

To use DBA to initialize users and groups without regenerating the system catalog,

1. Open the system catalog in DBA. Close any open lists or input windows.
2. From the Maintenance menu, select Initialize Users & Groups.

   The following prompt is displayed:

   Do you want to overwrite the existing table?
3. To initialize users and groups, select Yes.

To initialize users and groups from the command line without regenerating the system catalog, do one of the following. (For information on DBA command-line options, see "Command line options" on page 6-2.)

- At a Windows or UNIX prompt, use the following syntax:

  ```
  dbr SODBC_DBA:xfdba.dbr -c connect_string -i
  ```

- At an OpenVMS prompt, use the following:

  ```
  $ XFDBA -C connect_string -I
  ```

where *connect_string* has the *user_name/password/connect_filename* format.

For example, to initialize users and groups for the sample database in a Windows or UNIX environment, enter

```
dbr SODBC_DBA:xfdba.dbr -c DBADMIN/MANAGER/sodbc_sa -i
```

To initialize users and groups for the sample database in OpenVMS, enter

```
$ XFDBA -C DBADMIN/MANAGER/SODBC_SA -I
```

These examples assume that your connect file is **sodbc_sa**, that SODBC_DBA is set to the directory where the DBA program resides, and that the connect file is located in the GENESIS_HOME directory.

Viewing groups

To view groups, close any open lists and input windows and select Groups from the Maintenance menu of DBA. The Group List window is displayed, as shown in figure 6-4.

The Group List window displays a list of groups with the following information:

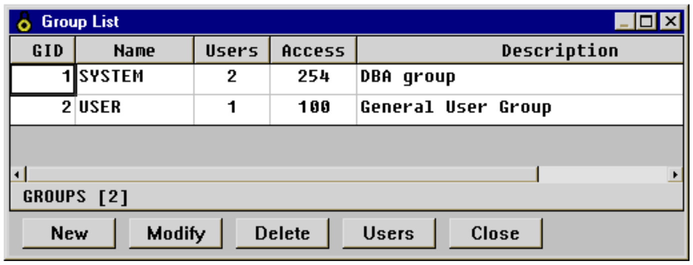
GID. An automatically assigned group ID number.

Name. The alphanumeric identifier for each group.

Users. The number of users assigned to each group.

Access. The access level of each user in the group (numeric, from 0 to 255).

Description. A brief description of each group.



GID	Name	Users	Access	Description
1	SYSTEM	2	254	DBA group
2	USER	1	100	General User Group

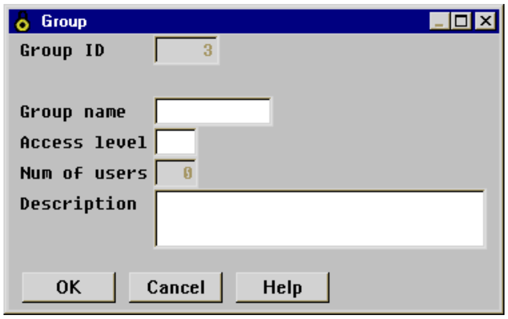
GROUPS [2]

Figure 6-4. The Group List window.

Creating a group

You can create up to 999,999 groups, and you can assign a maximum of 255 users to a group.

1. Open the Group List window. (See “Viewing groups” on page 6-14.)
2. From the Group Maintenance menu, select New Group. The Group window is displayed, as show in figure 6-5.



Group ID 3

Group name

Access level

Num of users 0

Description

Figure 6-5. The Group window.

3. Enter data in each field as described below.

   Group ID. An automatically assigned group number. This field is not modifiable.

   Group name. Enter an alphanumeric identifier of up to 10 characters.

   Access level. Enter a number between 0 and 255 that determines users’ read/write access to data. This level determines the access level of all users in the group. Note that a group must be set to at least 100 for users in that group to access the database.

   We recommend that you use levels 254 and 255 for administrative users only. For more information on setting access levels, see “Setting Security Levels” on page 8-2.

Num of users. The total number of users assigned to this group. This field is not modifiable and is set to zero when you're adding a new group.

Description. (optional) Enter an alphanumeric description of up to two lines of 30 characters.

4. Select OK or press F4.

Modifying a group

1. Open the Group List window. (See "Viewing groups" on page 6-14.)
2. In the Group List window, highlight the group you want to modify.
3. From the Group Maintenance menu, select Modify Group.
4. Make any changes. Then select OK or press F4.

Deleting a group

1. Open the Group List window. (See "Viewing groups" on page 6-14.)
2. In the Group List window, highlight the group you want to delete.
3. From the Group Maintenance menu, select Delete Group.

   A window is displayed with the selected group's name and description and the following prompt:

   Do you want to delete the current entry?

4. To delete the group, select Yes.

   Note that you can't delete a group that has users. The users must first be deleted or assigned to other groups.

Viewing users in a group

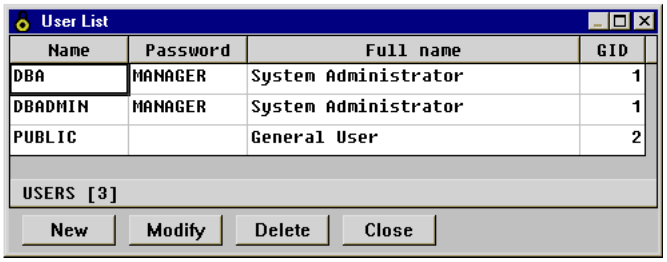
1. Open the Group List window. (See "Viewing groups" on page 6-14.)
2. Highlight the group in the Group List window.
3. From the Group Maintenance menu, select View Users.

   The User List window opens. For information on this window, see "Viewing all users" on page 6-17.

   To modify a user, see "Modifying a user" on page 6-18.

Viewing all users

To view a list of all users in the system catalog, close any open lists or input windows and select Users from the Maintenance menu. (You can also view a list of users for a specific group. See "Viewing users in a group" above.)



Name	Password	Full name	GID
DBA	MANAGER	System Administrator	1
DBADMIN	MANAGER	System Administrator	1
PUBLIC		General User	2

USERS [3]

Figure 6-6. The User List window.

The User List window displays the following information for each user:

Name. A case-sensitive alphanumeric identifier that you assign.

Password. A case-sensitive alphanumeric identifier. Users are not required to have passwords.

Full name. The user's full name.

GID. The ID of the group the user belongs to.

Adding a user

You can add up to 255 users to a group.

1. Open the User List window. (See "Viewing all users" above.)
2. From the User Maintenance menu, select New User. The User window is displayed, as shown in figure 6-7.
3. Enter data in each field as described below.

   User name. Enter an alphanumeric identifier for the user you are creating. It can be up to 10 characters long. This field corresponds to the Name column in the User List window and is case sensitive.

   Password. (optional) Enter an alphanumeric password. It can be up to 10 characters long. Passwords are case-sensitive and are visible only to users who can open DBA (users with an access level of 254 or greater). The following characters are *not* allowed: ~ @ # $ % ^ & * _ + = \ } { " , : ? / < > ! '

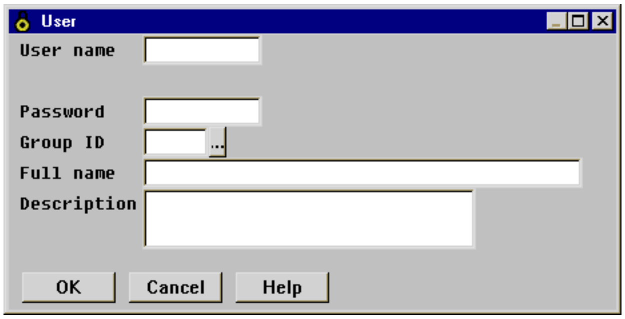


Figure 6-7. The User window.

Group ID. Enter the ID of the group you want to assign a user to. A user's access level is determined by the group it belongs to. To view a list of available groups, select Select Group from the User Maintenance menu.

Full name. (optional) Enter the user's full name. It can be up to 40 characters long.

Description. (optional) Enter an alphanumeric description of the user. The description can be up to 60 characters long.

Modifying a user

1. Open the User List window. See "Viewing all users" on page 6-17.
2. Highlight the user in the User List window.
3. From the User Maintenance menu, select Modify User.
4. Make any changes. Then select OK or press F4.

Deleting a user

1. Open the User List window. See "Viewing all users" on page 6-17.
2. Highlight the user in the User List window.
3. From the User Maintenance menu, select Delete User.

   A window is displayed with the selected user's user name, full name, and the following prompt:

   Do you want to delete the current entry?
4. To delete the user, select Yes.

Customizing Tables and Table Elements

You can use DBA to view tables, columns, indexes, and segments, and to delete tables and columns. And you can use a conversion setup file to change a table's access level, to add deleted tables back into the system catalog, and to change the path and filename for a table's data files.

Viewing and customizing tables

Viewing the tables in your system catalog

1. Open a system catalog in DBA. Close any open input windows or lists.
2. From the Maintenance menu, select Tables. The Table List window displays the following.

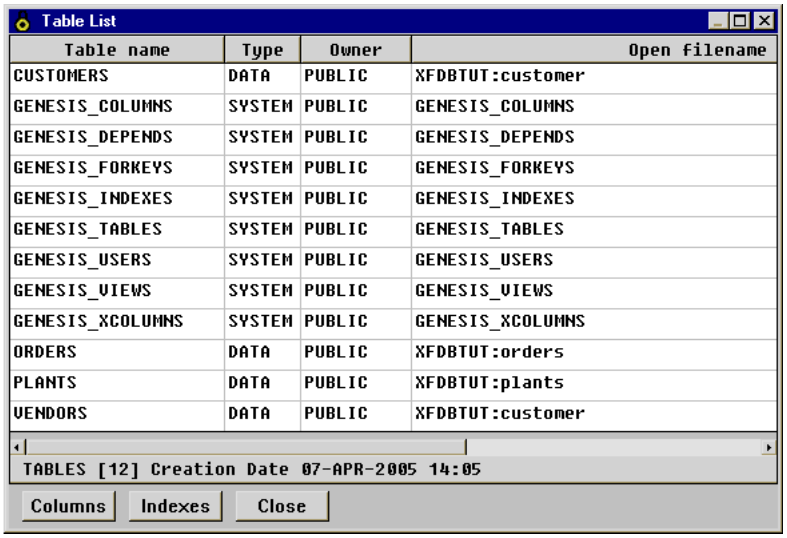


Table name	Type	Owner	Open filename
CUSTOMERS	DATA	PUBLIC	XFDBTUT:customer
GENESIS_COLUMNS	SYSTEM	PUBLIC	GENESIS_COLUMNS
GENESIS_DEPENDS	SYSTEM	PUBLIC	GENESIS_DEPENDS
GENESIS_FORKEYS	SYSTEM	PUBLIC	GENESIS_FORKEYS
GENESIS_INDEXES	SYSTEM	PUBLIC	GENESIS_INDEXES
GENESIS_TABLES	SYSTEM	PUBLIC	GENESIS_TABLES
GENESIS_USERS	SYSTEM	PUBLIC	GENESIS_USERS
GENESIS_VIEWS	SYSTEM	PUBLIC	GENESIS_VIEWS
GENESIS_XCOLUMNS	SYSTEM	PUBLIC	GENESIS_XCOLUMNS
ORDERS	DATA	PUBLIC	XFDBTUT:orders
PLANTS	DATA	PUBLIC	XFDBTUT:plants
VENDORS	DATA	PUBLIC	XFDBTUT:customer

TABLES [12] Creation Date 07-APR-2005 14:05

Figure 6-8. The Table List window.

Table name. The alphanumeric name of the table.

Type. The type of table:

SYSTEM	System table (created by **dbcreate**)
DATA	Data table (from your Synergy database)

Owner. The table owner. If the table was generated from a repository, this is "Public" (all users). If an ODBC application created it, the owner is the user who created it. Owner can't be modified.

Open filename. The name of the file that contains this table. Notice that the Open filename field may include an environment variable, which must be set in the connect file, in an environment setup file, or in the environment. This field corresponds to the Open filename field in Repository.

Access. The access level assigned to each table, which is 100 by default. You can change this level with a conversion setup file. See "Modifying table access levels" on page 6-22.

The creation date for the system catalog is listed in the information line at the bottom of the list.

To view hidden areas of the Table List, select Toggle View from the List menu.

Viewing a specific table

1. Open the Table List window. (See "Viewing the tables in your system catalog" on page 6-19.)
2. In the Table List Window, highlight the table name.
3. From the Table Maintenance menu, select View Table. The Table window displays the following information about the table you selected. (Note that you cannot modify the information in this read-only window.)

   Table owner. The table's owner. This is "Public" if generated from S/DE Repository.

   Table name. The alphanumeric name of the table.

   Table type. The type of table:

SYSTEM	System table (created by **dbcreate**)
DATA	Data table (from your Synergy database)

   File type. The type of file:

   ISAM
   RELATIVE
   ASCII (ASCII sequential)

   Access level. The access level assigned to this table. (For more information about access levels, see "Modifying table access levels" on page 6-22.)

   # of columns. The number of columns in the table.

   Record size. The number of characters (bytes) allowable in a single record in this table.

   File name. The actual name and location of the data file including an environment variable, if applicable. This corresponds to the Open filename field in the Table List.

Locating a table in a long list

1. Make sure the Table List window is active. (If the Table window is open, close it.)
2. From the List menu, select Find.
3. In the small window that's displayed, type the table name and select OK. The DBA program highlights the first match.

   The Find feature does *not* support wildcard characters, such as asterisk (*) or question mark (?).
4. To view the attributes of the highlighted table, press ENTER.

Deleting a table

Deleting a table removes user access to that table, and it removes the table, its columns, and its indexes only from the system catalog. The repository definition, relations, and the actual data file remain unaltered (only the reference to the data table is removed from the system catalog). Note that you cannot delete the system tables (see "System catalog" on page 1-5 for a list of system tables).

Rather than using DBA to delete a table, we recommend that you use DROP TABLE or make such changes using S/DE Repository and then regenerate the system catalog.

To use DBA to delete a table,

1. Open the Table List window. (See "Viewing the tables in your system catalog" on page 6-19.)
2. In the Table List window, highlight the name of the table to be deleted.
3. From the Table Maintenance menu, select Delete Table.

   A window is displayed with the selected table's name, file location, and the following prompt:

   Do you want to delete the current entry?

If the SODBC_CNVFIL environment variable is set to the location and filename of the conversion setup file, deleting a table in DBA automatically sets the conversion setup file's IN|OUT setting for the table to OUT. This prevents DBA and **dbcreate** from reinserting the table if you regenerate the system catalog using the **-c** option with **dbcreate** or the "Clear and recreate catalog" option with DBA.

To delete a table from the system catalog and keep it out, even if you regenerate the system catalog, do the following:

1. Generate a conversion setup file.
2. Set the SODBC_CNVFIL environment variable to the conversion setup file.
3. Delete the table in DBA using the above procedure.

   Alternatively, you can change the IN|OUT setting for the table to OUT by manually editing the conversion setup file. For information, see "Generating and Editing a Conversion Setup File" on page 6-27.
4. Do one of the following:
   - Use **dbcreate** from the command line to regenerate the system catalog. Be sure to use the **-c** option and to specify the conversion setup file as input.
   - Use DBA to regenerate the system catalog. Be sure to use the "Clear and recreate catalog" option and to specify the conversion setup file as input.

Adding a deleted table back into the system catalog

There are two ways to add a deleted table back into the system catalog. You can regenerate the system catalog without using a conversion setup file and then regenerate the conversion setup file, or you can do the following:

1. Generate a conversion setup file.
2. Open the conversion setup file in a text editor and manually change the IN|OUT setting for the table.
3. Using the conversion setup file as input, regenerate the system catalog.

For more information, see "Generating and Editing a Conversion Setup File" on page 6-27 and "Regenerating the System Catalog" on page 4-9.

Modifying table access levels

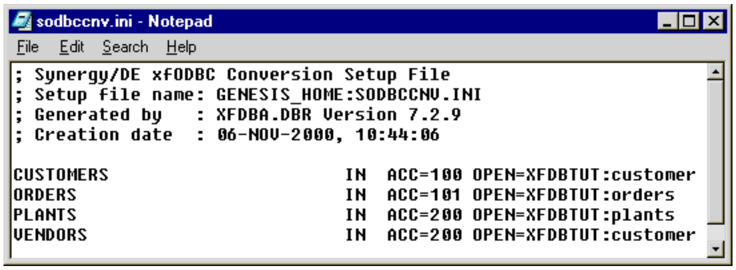
When you first generate the system catalog, all tables are set by default with an access level of 100 (except the GENESIS_* tables, which are set to 0). To change these access levels,

1. Generate a conversion setup file.
2. Open the conversion setup file in a text editor and manually change the levels.
3. Using the conversion setup file as input, regenerate the system catalog.

For more information, see "Generating and Editing a Conversion Setup File" on page 6-27 and "Regenerating the System Catalog" on page 4-9.

In the example in figure 6-9, the access levels for the Orders, Plants, and Vendors tables have been modified.

We recommend setting table access levels to even numbers. An even-numbered access level enables the *xf*ODBC driver to read from, but not write to a table. If you set a table's access level to an odd number, a user may update a record that's in use by another application.



```
; Synergy/DE xfODBC Conversion Setup File
; Setup file name: GENESIS_HOME:SODBCCNV.INI
; Generated by    : XFDBA.DBR Version 7.2.9
; Creation date   : 06-NOV-2000, 10:44:06

CUSTOMERS                        IN  ACC=100 OPEN=XFDBTUT:customer
ORDERS                           IN  ACC=101 OPEN=XFDBTUT:orders
PLANTS                           IN  ACC=200 OPEN=XFDBTUT:plants
VENDORS                          IN  ACC=200 OPEN=XFDBTUT:customer
```

Figure 6-9. Conversion setup file with modified access levels.

Changing the location of table data files

1. Generate a conversion setup file.
2. Open the conversion setup file in a text editor and manually change the path and filename of the data file.
3. Using the conversion setup file as input, regenerate the system catalog.

   For more information, see "Generating and Editing a Conversion Setup File" on page 6-27 and "Regenerating the System Catalog" on page 4-9.

Viewing and deleting columns

Viewing columns in a table

You can view a list of all the columns in a table, and you can view the attributes of each individual column.

Note that the terms *column* and *field* are often used interchangeably. When a database is presented visually as a data sheet, typically the top row lists all the *columns*. On a form in which a user enters data, these columns are represented as *fields*. For our purposes, a repository field is equivalent to a system catalog column.

To view a list of columns in a table,

1. Open the Table List window. (See "Viewing the tables in your system catalog" on page 6-19.)
2. In the Table List Window, highlight a table name.
3. From the Table Maintenance menu, select View Columns.

   The Column List window displays a list of the columns, sorted by offset, along with attributes of these columns.

   Column name. Heading name for each data column.

   Type. Synergy data type for the column (ALPHA, DATE, USER, etc.).

   Size. The size of the field (in bytes) before the system catalog was generated.

   Precision. The number of characters to the right of the decimal point in an implied-decimal field.

   Position. Column's start position (in bytes) from the leftmost character in the record. The columns in this window are sorted by their offset position.

Viewing information about a column

To view more complete information about an individual column,

1. Open the Column List window. (See “Viewing columns in a table” on page 6-23.)
2. In the Column List window, highlight the column.
3. From the Column Maintenance menu, select View Column.

The Column window displays the following information. You cannot modify the information in this read-only window.

Column name. Name of the column.

SQL type. Data type as defined by the SQL_DescribeCol ODBC API function (SQL_TIME, SQL_BIT, etc.).

The following fields are in the “Synergy details” portion of the Column window.

Type. The Synergy data type (ALPHA, DATE, etc.).

Position. The column’s start position (in bytes) from the leftmost character in the record.

Size. The size of the field as defined in S/DE Repository.

Signed. Indicates whether a numeric column in the system catalog is signed or unsigned (which is determined by whether “Negative allowed” or a positive range of values is selected for the corresponding field in the repository).

Precision. The number of characters to the right of the decimal point in an implied-decimal field. If an implied-decimal field has no characters to the right of the decimal point, this field is blank.

User data. The information entered in the S/DE Repository User data field. It can contain any string of up to 30 characters and is available only for user-defined repository fields. *xf*ODBC checks the content of this field when it generates a system catalog. If this field contains a string in the format ^CLASS^=*date_format*, *xf*ODBC interprets the column as a date with the specified format. See “Date and time fields” on page 3-15.

The following fields are in the “Support details” portion of the Column window.

Type. Internal data type used by *xf*ODBC:

Ordinal position. Ordinal position of the column in the record. A column with an ordinal position of 1 appears at the leftmost position in a data sheet when viewed using an ODBC-enabled database application.

Length. For use by Synergy/DE Developer Support.

Internal format. For use by Synergy/DE Developer Support.

Null allowed. Indicates whether a null value (or equivalent) is allowed in this column: Y for yes or N for no. See “Preventing null updates and interpreting spaces, zeros, and null values” on page 3-27.

Deleting a column

Note the following:

- Using DBA to delete tables and columns can lead to unpredictable results. We *strongly* recommend that you make such changes using S/DE Repository and then regenerate the system catalog.
- You cannot delete columns from the system tables, tables in the system catalog whose names start with GENESIS_.

1. Open the Column List window. See "Viewing columns in a table" on page 6-23.
2. In the Column List window, highlight the column to be deleted.
3. From the Column Maintenance menu, select Delete Column.

   A window is displayed with the selection's table name, column name, and the following prompt:

   Do you want to delete the current entry?

   Deleting a column removes only its reference from the system catalog. This action does *not* delete actual data. However, a deleted column is no longer accessible to an ODBC-enabled application.

Viewing indexes in a table

Indexes are a superset of keys. They're defined in your repository, and they enable quicker access to records.

1. Open the Table List window. See "Viewing the tables in your system catalog" on page 6-19.
2. In the Table List window, highlight the table.
3. From the Table Maintenance menu, select View Indexes.

   The Index List window displays a list of indexes with the following information:

   Index. The name of the index as defined in the repository.

   Type. The index type:

   UNIQUE
   NON-UNIQUE (duplicates allowed)

   Segments. The number of segments for the index.

Viewing segments for an index

1. Open the Index List window. (See "Viewing indexes in a table" on page 6-25.)
2. In the Index List window, highlight the index; then select View Segments from the Index Maintenance menu.

   The Segment List window displays the following information:

   Segment#. The sequence of the segment in the index.

   Field name. The field name of the index segment.

   Order. The sort order of the index: ASC for ascending or DESC for descending.

Generating and Editing a Conversion Setup File

Conversion setup files are text files that contain information on tables in the data files. This information includes table names, table access levels, whether a table will be part of the system catalog, data file locations, as well as information about the conversion setup file itself (name, version of DBA used to generate it, and the date it was generated). For an example, see figure 6-11 on page 6-29.

Conversion setup files are used when you regenerate a system catalog and enable you to

- change the path and filename specifications of the data files.
- change the access level of a table.
- remove a table from the system catalog.
- add a deleted table back into a system catalog.
- prevent a table from being added back into a system catalog if the system catalog is regenerated. Although you can use DBA to delete a table from the system catalog, you must use a conversion setup file to keep it out of the system catalog when regenerating.

To use a conversion setup file,

1. Generate the file. There are two ways to do this. You can open the system catalog and then generate the file from DBA, or you can generate the file from the command line. See "Generating the conversion setup file from the command line" on page 6-28.
2. Make any necessary changes to the conversion setup file. See "Editing the conversion setup file" on page 6-29.
3. Using the conversion setup file, regenerate the system catalog. See "Regenerating the System Catalog" on page 4-9.

Generating the conversion setup file from DBA

1. Open the system catalog in DBA. (See "Opening the System Catalog in DBA" on page 6-10.) Close any open input lists or windows.
2. From the Catalog menu, select Generate Conversion Setup File.

   The Generate Conversion Setup File window opens and prompts you for the name and location of the conversion setup file.

   If the SODBC_CNVFIL environment variable *is* set, this window opens with the name and location that SODBC_CNVFIL specifies. If the SODBC_CNVFIL environment variable is *not* set, this window is automatically populated with the following path and filename: GENESIS_HOME:SODBCCNV.INI.
3. Enter the full path and filename for the conversion setup file you want to generate (or accept the default), and then click OK. You can use environment variables as shown in figure 6-10.

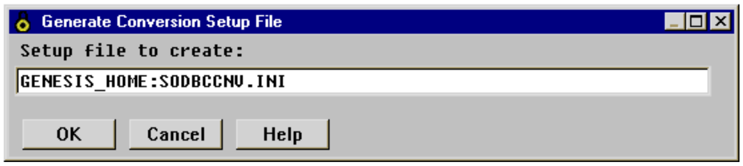


Figure 6-10. Specifying the path and filename for a conversion setup file.

DBA generates the conversion setup file, which reflects the current table attributes. Once you've created the conversion setup file, you can use this file to affect the way the system catalog is regenerated. See "Regenerating the System Catalog" on page 4-9.

Generating the conversion setup file from the command line

To generate a conversion setup file from the command line, do one of the following:

- At a Windows or UNIX prompt, use the following syntax:

  `dbr SODBC_DBA:xfdba.dbr -c` *connect_string* `-g` *filename*

- At an OpenVMS prompt, use the following syntax:

  `$ XFDBA -C` *connect_string* `-G` *filename*

where *connect_string* has the *user_name/password/connect_filename* format and *filename* is the full path and filename of the conversion setup file.

The following examples generate a conversion setup file named **cnv.ini** in a directory named data.

Windows:

```
dbr SODBC_DBA:xfdba -c DBADMIN/MANAGER/sodbc_sa -g c:\data\cnv.ini
```

UNIX:

```
dbr SODBC_DBA:xfdba -c DBADMIN/MANAGER/sodbc_sa -g /usr/data/cnv.ini
```

OpenVMS:

```
$ XFDBA -C DBADMIN/MANAGER/SODBC_SA -G DKA600:[DATA]CNV.INI
```

This procedure opens DBA, generates the conversion setup file you specify—*without prompting you for confirmation*—and then closes DBA, returning you to the command line.

For information on DBA command line options, see "Command line options" on page 6-2.

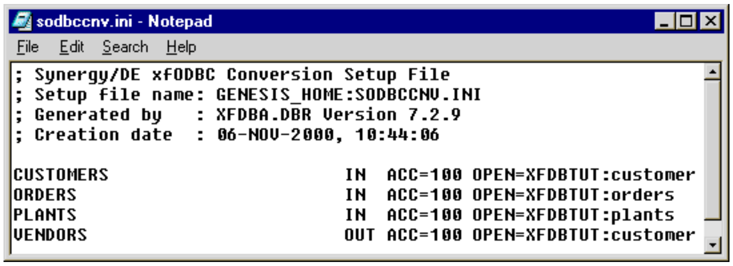
Editing the conversion setup file

Using a text editor, you can modify the conversion setup file in the following ways:

- Mark a table IN or OUT, so that it can be included in or excluded from a system catalog.
- Change access level for the data tables.
- Change the file location of a table in the system catalog.

Whenever you manually change the conversion setup file, you *must* regenerate the system catalog for the changes to go into effect. See "Regenerating the System Catalog" on page 4-9.

Figure 6-11 shows a conversion setup file. This sample was generated on Windows from the sample database and was then modified: the IN/OUT setting for the Vendors table was set to OUT.



```
; Synergy/DE xfODBC Conversion Setup File
; Setup file name: GENESIS_HOME:SODBCCNV.INI
; Generated by   : XFDBA.DBR Version 7.2.9
; Creation date  : 06-NOV-2000, 10:44:06

CUSTOMERS                          IN  ACC=100 OPEN=XFDBTUT:customer
ORDERS                             IN  ACC=100 OPEN=XFDBTUT:orders
PLANTS                             IN  ACC=100 OPEN=XFDBTUT:plants
VENDORS                            OUT ACC=100 OPEN=XFDBTUT:customer
```

Figure 6-11. A conversion setup file for the sample database.

Each table is listed with the following:

table_name [IN|OUT][ACC=*access_level*][OPEN=*path*]

table_name

The name of a table in your Synergy data files. In figure 6-11, the conversion setup file lists four tables: CUSTOMERS, ORDERS, PLANTS, and VENDORS.

IN | OUT

(optional) Determines whether the table is considered when the system catalog is regenerated. If you use **dbcreate** with the **-x** option or DBA with the "Add new tables and update existing tables" option, tables marked as OUT are *not* overwritten or removed. If you use **dbcreate** with the **-c** option or DBA with the "Clear and recreate catalog" option, tables marked as OUT are omitted from the system catalog. The default is IN. In figure 6-11, for example, the Vendors table has been marked as OUT. If the system catalog is regenerated using this file and the **-c** option (**dbcreate**) or the "Clear and recreate catalog" option (DBA), the vendors table will no longer be accessible to an ODBC-enabled application. Note that data files are *not* altered; the Vendor table will remain as part of the database.

access_level

(optional) Access level of the table. This can be any numeric value from 0 to 255. The default value is 100. See "Modifying table access levels" on page 6-22.

path

(optional) Location and name of the data file for the *table_name* table. If this option is not specified or if no path or environment variable precedes the filename specified in the Open filename field of the Repository file definition, the datasource specification in the connect file is used. In figure 6-11, the Vendors table is part of the ISAM file named **customer**.

Verifying the System Catalog

If you've made changes to a system catalog, you may want to verify that the system catalog tables still match the definitions in the repository or the database.

Comparing the system catalog to repository definitions

You can create a verification log that lists table names, file names, the number of columns read and defined for each table, and the number of indexes read for each table. You can create this file from the DBA program or from the command line.

Using DBA to create a verification log

1. Open the system catalog in DBA. Close any open lists or input windows.
2. From the Catalog menu, select Verify.

   The Verify System Catalog window opens and prompts you for a name and location.
3. Accept the default or enter a new location and name. Then select OK.

   A message displays telling you how many tables were verified and how many errors were found. However, be sure to read the log file. It contains more complete information.
4. Select OK.

Creating a verification log from the command line

To create a verification log from the command line, use the following syntax at a Windows or UNIX prompt:

```
dbr SODBC_DBA:xfdba.dbr -c connect_string -v verify_file
```

or use the following syntax at an OpenVMS prompt:

```
$ XFDBA -C connect_string -V verify_file
```

where *connect_string* has the *user_name/password/connect_filename* format and *verify_file* is the full path and filename for the verification log.

The following examples generate a verification log named **vrfy.log** in a directory named data.

Windows:

```
dbr SODBC_DBA:xfdba.dbr -c DBADMIN/MANAGER/sodbc_sa -g c:\data\vrfy.log
```

UNIX:

```
dbr SODBC_DBA:xfdba.dbr -c DBADMIN/MANAGER/sodbc_sa -g /usr/data/vrfy.log
```

OpenVMS:

```
$ XFDBA -C DBADMIN/MANAGER/SODBC_SA -G DKA600:[DATA]VRFY.LOG
```

For information on DBA command-line options, see "Command line options" on page 6-2.

Comparing the system catalog to a database

You can compare the tables in the system catalog to the data definitions in a database. You can do this from the command line or from the DBA program. The results of the comparison are saved to a log file.

For information on comparing from the command line, see fcompare in the "Synergy DBMS" chapter of *Synergy Tools*.

Using DBA to compare the system catalog to a database

1. Open the system catalog in DBA. Close any open lists or input windows.
2. From the Catalog menu, select Compare to Files.

   The Compare System Catalog to Files window opens.
3. Fill in or select from the following:

   Connect file. Enter a connect file name or accept the default. The connect file determines which database and system catalog will be compared.

   Option. Select All to compare all tables in the system catalog to the database. Select Specific to compare one system catalog table to the database.

   Table name. If you selected the Specific option for the Option field, enter the name of the table you want compared to the database. This is available only if Specific is selected for the Option field.

   Verify data. Select this option to verify data in the ISAM files. (Data definitions are compared regardless of this setting, but data is verified only if this option is selected.) This option is available only if Specific is selected for the Option field.

   Log file. Enter the path and name for the log file. The log file will contain the results of the comparison.

   Verbose logging. Select this option if you want to generate additional (verbose) output information.
4. Click OK.

7

Creating Routines for User-Defined Data Types

Introduction

If your repository contains user-defined fields—fields created as user type fields in S/DE Repository—you can write C language routines to manipulate user-defined field data as it's read from and written to the database. (This is similar to creating user-overloadable ReportWriter routines.) For example, if a user-defined field stores street addresses in mixed-case characters, and you want the *xf*ODBC driver's output to be in all uppercase characters, you can create a routine to convert the addresses as they're read from the database. And if you want the addresses converted to mixed-case characters as they're input, you can also create a routine to do that.

To create a user-defined data routine,

- on Windows, you'll create a DLL named **xfodbcusr.dll** in the connect subdirectory of the main Synergy/DE installation directory.
- on UNIX, you'll create a shared library, **XFODBCUSR.so**, in the connect subdirectory of the main Synergy/DE installation directory.
- on OpenVMS, you'll create a shared image, **xfodbcusr_so.exe**. The logical XFODBCUSR_SO (set in **CONNECT_STARTUP.COM**) will point to the location of this shared image.

These are the files that the *xf*ODBC driver calls for user-defined data routines. Create 32-bit versions for 32-bit Connectivity Series. Create 64-bit versions for 64-bit Connectivity Series. These files must reside on the same machine as the data. For client/server configurations, this means the file must reside on the server.

To help you learn to create your own user-defined data routines, we've created a tutorial that steps you through the process; see "Using xfodbcusr.c As an Example" on page 7-6. And see "Using xfodbcusr.c As a Template" on page 7-3 for information on **xfodbcusr.c**, a C language source file that you can use as a template for your user-defined data routines. Note the following:

- User-defined data routines are called for all user type fields except those that have ^CLASS^ embedded within the user data string. ^CLASS^ is reserved for a subset of date storage formats supported by *xf*ODBC. For more information, see "Date and time fields" on page 3-15.
- For information on using user-defined fields as tags or keys, see "Setting Up a Repository" on page 3-2 and "Optimizing with Keys" on page 10-2.
- The Connectivity Series distribution includes base versions and working versions of the files used for user-defined data type routines (the files discussed in this chapter). The base versions are in the connect\synodbc\usr\base directory. *Do not make changes to these files*—they are included so you always have original versions. The working versions are in the connect\synodbc\user directory (except for **xfodbcusr.dll** on Windows, **XFODBCUSR.so** on UNIX, or **xfodbcusr_so.exe** on OpenVMS—these are in the connect directory). When you install Connectivity Series, the installation replaces base versions of these files to ensure you always have the latest versions. To preserve your changes, however, working versions are replaced only if they are unchanged.

Using xfodbcusr.c As a Template

To make creating user-defined data routines convenient, we've included a template file, **xfodbcusr.c**, in the *xf*ODBC installation. This file has the source code, written in C, for the file that the *xf*ODBC driver calls for user-defined data routines.

- On Windows, this file is a DLL, **xfodbcusr.dll**.
- On UNIX, this file is a shared library, **XFODBCUSR.so**.
- On OpenVMS, this file is a shared image, **xfodbcusr_so.exe**.

Until you overwrite it, this DLL, shared library, or shared image simply returns a value of 1, which tells the *xf*ODBC driver that there's no data manipulation to be done. (If, in Repository, you've changed the user-defined fields from alpha to numeric or date, the return value of 1 will indicate an error.) However, if you want to add user-defined data routines, you can do so by adding the code for these routines to **xfodbcusr.c** and then creating a new version of the DLL, shared library, or shared image from this source file. The following section describes the routines included in **xfodbcusr.c**. To see an example of how **xfodbcusr.c** can be used to create user-defined data routines, see "Using xfodbcusr.c As an Example" on page 7-6.

Functions in xfodbcusr.c

xfodbcusr.c has four functions:

- user_to_alpha()—code added to this function manipulates alpha user type field data as it's read from a database.
- user_to_number()—code added to this function manipulates numeric and date user type field data as it's read from a database.
- alpha_to_user()—code added to this function manipulates alpha user type field data as it's written to a database.
- number_to_user()—code added to this function manipulates numeric and date user type field data as it's written to a database.

Unless you've modified **xfodbcusr.c**, these functions already have code that's been commented out by #ifdef statements. (You'll need this code to complete the example in "Using xfodbcusr.c As an Example" on page 7-6.) To add your own data-manipulation routines, replace this code with the routines you've written.

user_to_alpha

```
int user_to_alpha(char *tablename, char *columnname,
                  char *userstr, char *recdata, char *indata,
                  int inlen, int maxoutlen, char *outdata,
                  int *outlen);
```

user_to_number

```
int user_to_number(char *tablename, char *columnname,
                   char *userstr, char *recdata, char *indata,
                   int inlen, char *outdata);
```

alpha_to_user

```
int alpha_to_user(char *tablename, char *columnname,
                  char *userstr, char *recdata, char *indata,
                  int inlen, int maxoutlen, char *outdata,
                  int *outlen);
```

number_to_user

```
int number_to_user(char *tablename, char *columnname,
                   char *userstr, char *recdata, char *indata,
                   char *outdata);
```

Arguments

tablename

The name of the table. (null-terminated string)

columnname

The name of the column. (null-terminated string)

userstr

The string from the User data field in a Repository field definition. (null-terminated string)

recdata

The full record. This is passed so the routine can use other values in the record.

indata

Input data string. This is the string that the routine will convert.

- For the user_to_alpha() function, *indata* is an alpha string.
- For the user_to_number() function, the format of *indata* is determined by the format of the user-defined field in your system catalog.
- For the alpha_to_user() function, *indata* is an alpha string.
- For number_to_user() function, *indata* is formatted as a 28.10 Synergy DBL zoned decimal for user-defined numeric fields and is formatted as a *YYYYMMDD* date for user-defined date fields.

inlen

The length of the input data string.

maxoutlen

Maximum output data length. Not currently used.

outdata

Output data string. This is the string produced by the routine.

- For the alpha_to_user() and number_to_user() functions, your routine determines the format of *outdata*.
- For the user_to_alpha() function, *outdata* must be formatted as an alpha string.
- For the user_to_number() function, if the string is from a user-defined numeric field, *outdata* must be formatted as a 28.10 Synergy DBL zoned decimal. If the string is from a user-defined date field, *outdata* must be formatted as a *YYYYMMDD* date.

outlen

Output data string length. Not currently used.

Return values

The functions in **xfodbcusr.c** return these values:

> 0	Instructs the driver not to use *outdata*. If the data is from a numeric or date field, a number greater than zero indicates an error (as does any number other than zero).
0	Instructs the driver to use *outdata*.
< 0	Indicates that there has been a data conversion error. Note that the application that's accessing the data may not display an error and may not continue to process data.

Note the following:

- If your system catalog was generated from a repository with numeric or date user-defined fields, your routines must include conditional statements that set the *outdata* value. The tutorial uses column and table names, but you can also use *userstr* or *recdata*. In addition, *outdata* must have a value, and the routine must return a value of 0.
- A routine for user-defined data can return a null date for a SELECT operation. To do this, fill the user_to_number() *outdata* argument with either eight zeros or eight spaces. Both designate a null date value.

Using xfodbcusr.c As an Example

This tutorial guides you through the steps needed to create user-defined data routines. As you follow the tutorial, you'll modify **xfodbcusr.c** to include routines that

- convert addresses to mixed-case characters as they're input and all uppercase characters as they're retrieved by an application.
- change dates to the fifteenth of the month as they're output, and the first of the month as they're written by an application.
- store customer limits for Oregon customers to $2000.75, no matter what number you enter, and add $200.00 to customer limits for Oregon customers as the limit is retrieved from the database.

Once you've modified the code in **xfodbcusr.c**, you'll compile and link the modified file as **xfodbcusr.dll** (Windows), **XFODBCUSR.so** (UNIX), or **xfodbcusr_so.exe** (OpenVMS). You can then watch it work as you use *xf*ODBC to read and write data to the sample database supplied with the *xf*ODBC installation.

Note that these routines are designed to work only with the sample database that's included with the *xf*ODBC installation, and that these routines, like all user-defined data routines, can't change the size of a field, only the contents.

1. Open Repository and do the following:
   - Set the *xf*ODBC sample database as the current repository. The repository files for the sample database are located in the connect\synodbc\dict subdirectory of the main Synergy/DE installation directory.
   - Change the CUST_STREET field in the CUSTOMERS structure from type alpha to type user, and set the class to alpha.
   - Change the CUST_LIMIT field in the CUSTOMERS structure from type decimal to type user, and set the class to numeric.
   - Change the OR_ODATE field in the ORDERS structure from type date to type user, and set the class to date.
   - Save your changes as you exit Repository.

   For help with any of the tasks in this step, see the *Repository User's Guide*.
2. Run **dbcreate** with the **-x** or **-c** option (overwrite or create). Either option will work. For information on **dbcreate** options, see "Generating a system catalog from the command line" on page 4-3.
3. Open **xfodbcusr.c** in a text editor. (You'll find this file in the connect\synodbc\user subdirectory of the main Synergy/DE installation directory.) Remove or comment out the #ifdef and #endif lines from all four of the functions: user_to_alpha(), user_to_number(), alpha_to_user(), and number_to_user(). For client/server configurations, uncomment the section for your host machine's operating system. Save your changes and close the file.

4. Run one of the following:
   - On Windows, run **makeusr.bat**.
   - On UNIX, run **makeusr**.
   - On OpenVMS, run **makeusr.com**.

   You'll find these files in the connect\synodbc\user subdirectory of the main Synergy/DE installation directory. These files compile **xfodbcusr.c**, link it, and save it as one of the following:
   - **xfodbcusr.dll** on Windows
   - **XFODBCUSR.so** on UNIX
   - **xfodbcusr_so.exe** on OpenVMS

   To compile and link using Microsoft Developer Studio (Visual C/C++ version 6.0 or higher), create a new Windows-32 Dynamic-Linked Library project, add **xfodbcusr.c**, and then build.

5. Copy the DLL, shared library, or shared image you just created to the connect subdirectory of the main Synergy/DE installation directory. This will overwrite the **xfodbcusr.dll**, **XFODBCUSR.so**, or **xfodbcusr_so.exe** file that's already there. For client/server configurations, copy this to the connect subdirectory of the main Synergy/DE installation directory on the server.

   VMS

   Note that for OpenVMS, the XFODBCUSR_SO logical must be set to the path and filename of the **xfodbcusr_so.exe** shared image. See "XFODBCUSR_SO (OpenVMS)" on page A-8 for more information.

6. Start an ODBC-enabled application, and access the data in the sample database. (For help with this step, see chapter 9, "Accessing a Synergy Database.") Note the following:
   - All text in the CUST_STREET column is capitalized.
   - All dates in the OR_ODATE column are set to the first of the month.
   - CUST_LIMIT values for customers who live in Oregon (OR) are $200.00 greater than other CUST_LIMIT values.

7. To verify that addresses are converted to mixed-case characters as they're input and all uppercase characters as they're retrieved by an application, do the following:
   - From the ODBC-enabled application, INSERT or UPDATE a row, changing the CUST_STREET value to all uppercase characters.
   - Return to Repository, and change CUST_STREET back to type alpha.
   - Generate a system catalog for the sample database by running **dbcreate**.
   - Finally, return to the ODBC-enabled application and access the sample database. The text in the CUST_STREET column should be in mixed case.

8. To verify that dates are stored as the fifteenth of the month and display as the first of the month when retrieved, do the following:
   - From the ODBC-enabled application, INSERT or UPDATE a row, changing the OR_ODATE value to any date other than the fifteenth. This date will be displayed as the first, but stored as the fifteenth.
   - In Repository, change for OR_ODATE back to type date.
   - Generate a system catalog for the sample database by running **dbcreate**.
   - Finally, return to the ODBC-enabled application, access the sample database, and notice that the date in the OR_ODATE column is the fifteenth.
9. To verify that customer limits for Oregon customers are stored as $2000.75 and that $200.00 is added when they're retrieved, do the following:
   - From the ODBC-enabled application, INSERT or UPDATE a row, changing CUST_LIMIT to any value.
   - In Repository, change CUSTOMER_LIMIT back to decimal.
   - Generate a system catalog for the sample database by running **dbcreate**.
   - Return to the ODBC-enabled application, access the sample database, and notice that if you changed a CUST_LIMIT for a customer from Oregon, the numeric_to_user() function changed the limit to $2000.75, no matter what you entered. If the customer isn't from Oregon, the limit you entered is stored.

You can also use the following methods to debug user-defined data routines:

- Create an alpha overlay column on the user-defined column and compare the values in an ODBC-enabled application.
- Create a separate system catalog and connect file that use the same data but without user-defined fields. Then run the ODBC-enabled application with both system catalogs, and compare the output.

8

Configuring Data Access

Setting Security Levels

Access to your database is controlled by the access levels assigned to tables and groups. When you generate a system catalog, tables are assigned a default access level. When you initialize users and groups, **dbcreate** or the *xf*ODBC Database Administrator (DBA) program creates default groups (which contain default users) with predefined access levels. After initialization, you can manage access to your database by creating your own groups with appropriate access levels and creating a conversion setup file to customize the access level of your tables.

Without the user and group files (**sodbc_user.***, **sodbc_group.***), there is no user or password validation when connecting to the database, and all connected users have read-only access to all database tables. Be sure to generate these files (see "Generating the System Catalog" on page 4-2) and keep them with the other system catalog files.

Understanding access levels for tables and groups

Both tables and groups have access levels, which range from 0 to 255. The access level of a group applies to all users in the group. For users in a group to access a table, that group must have an access level equal to or greater than the table's access level. Access levels are further defined by odd and even numbers. Even numbers allow read-only access; odd numbers allow read/write access. This applies to both tables and groups.

Refer to the table below to see how the levels and read/write access of tables and groups interact. Note the following:

- Users in group 1 cannot access table B because their access level (100) is less than that of the table (101). Note that a group must be set to at least 100 for users in that group to access the database.
- Even with an access level of 254, group 3 cannot write to table B because 254 is an even number, meaning group 3 has read-only access.
- In order for a group to have read/write access to a table, both the group and the table must have an odd-numbered access level. Otherwise, the group will have only read access.
- Users in a group with an access level of 255 are "super users": they have read/write access to *all* tables—even tables with read-only access.

	Group 1 access=100 (R)	**Group 2** access=101 (R/W)	**Group 3** access=254 (R)	**Group 4** access=255 (R/W)
Table A access=100 (R)	Read	Read	Read	Read/write
Table B access=101 (R/W)	No access	Read/write	Read	Read/write

When you generate your system catalog, all tables are assigned an access level of 100 (read-only) by default. You can use a conversion setup file to change table access levels. For information, see "Modifying table access levels" on page 6-22.

When you initialize users and groups, DBA creates three default users (DBA, DBADMIN, and PUBLIC) and two default groups (SYSTEM and USER). You can use DBADMIN to assign and modify group access levels. We recommend that only the system administrator be assigned to a group with an access level of 254 or higher. For information on how to assign and modify group access levels, see "Creating a group" on page 6-15.

⚠ To update data in a Synergy database, we strongly recommend using a Synergy application that's designed to efficiently maintain database integrity. If you use an ODBC-enabled application to write to a Synergy database, you may run into record-locking issues. For information, see "Statements that Modify Data" on page B-46.

Setting Up Access with DSNs

Using the ODBC Data Source Administrator, you can create and manage data source names (DSNs). Each DSN has a name and contains information needed to access a database—for example, the name of the connect file and user and password information. Once you've created a DSN for a database, users can access the database from an ODBC-enabled application by selecting the DSN. DSNs free end-users from having to enter the location of the data files and other connection information.

When you install *xf*ODBC, the installation adds a sample DSN named **xfODBC**. Except for some default settings, this DSN specifies one thing: it specifies that the Vortex driver is "Genesis," which means that the DSN is for a local database. Users can access any local ODBC-enabled Synergy database with this DSN, but they will be prompted for a connect file, user name, and password because these aren't specified in the DSN. You can modify this DSN and you can create your own.

On Windows, you must have write privileges to the Windows registry to create or modify a DSN. If you don't, you'll get a "Writing config data failed" error.

User DSNs, system DSNs, and file DSNs

There are three types of DSN: user DSNs, system DSNs, and file DSNs. Typically, ODBC-enabled applications can use all of these. Some older versions of Microsoft Query, however, use only file DSNs.

- *User DSNs* are data sources that are available to one user on one machine. User DSNs contain all or part of the information needed to access a Synergy database: connect file, user name, password, and port.
- *System DSNs* are available to all users of one machine, and (like user DSNs) system DSNs contain all or part of the information needed to access a Synergy database.
- *File DSNs* can be made available to all users who have the same driver and are on the same network. *Non-sharable* file DSNs are set to directly access an existing user DSN. They contain a reference to a user DSN, but nothing else. Typically non-sharable file DSNs are used in place of user DSNs for versions of Microsoft Query that don't support user DSNs. Non-sharable file DSNs can also be placed on a client, network machine, or stand-alone system. But if a non-sharable file DSN is placed on a network machine, it must reference user DSNs on the clients. Changes you make to a non-sharable DSN are automatically applied to the corresponding user DSN.

  We don't recommend *sharable* file DSNs because you have to manually edit the Windows registry to create them.

Adding a user or system DSN

1. Open the ODBC Data Source Administrator, which is available from Windows Control Panel. Be sure to use the correct version of this utility:

- 32-bit applications require 32-bit DSNs for ODBC access. To create 32-bit DSNs, 32-bit Connectivity Series must be installed. 32-bit DSNs are created by the 32-bit ODBC administrator, odbcad32.exe (which is located in %windir%\SysWOW64 on 64-bit machines).
- 64-bit applications require 64-bit DSNs for ODBC access. To create 64-bit DSNs, 64-bit Connectivity Series must be installed on a 64-bit Windows machine. 64-bit DSNs are created by the native 64-bit ODBC Administrator.

Note that on 64-bit Windows machines, there is usually a mixture of 32-bit and 64-bit applications, especially when using Visual Studio to develop 64-bit applications. For an ODBC connection that could involve both 32-bit and 64-bit applications, we recommend that you create both a 32-bit DSN and a 64-bit DSN that are identical in all aspects (including name).

Once you've opened the ODBC Data Source Administrator on 64-bit Windows, you can determine which version is running by doing one of the following:

- On Windows 8, look at the title bar, which indicates wether this program is 32-bit or 64-bit—e.g., "ODBC Data Source Administrator (32-bit)".
- For versions of Windows prior to Windows 8, open Windows Task Manager and find the **odbcad32.exe** entry. If the image name ends in *32 (see figure 8-1), the 32-bit version of the ODBC Data Source Administrator is running. Otherwise, the 64-bit version is running.

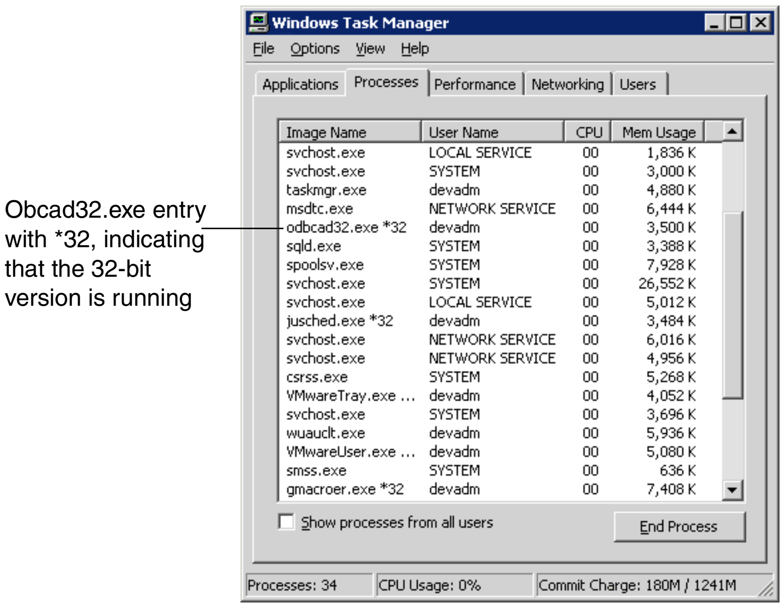
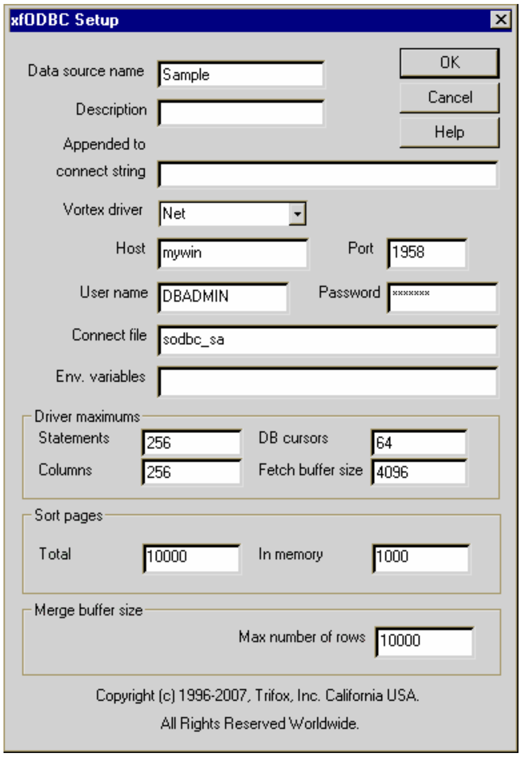

Obcad32.exe entry with *32, indicating that the 32-bit version is running
Windows Task Manager
File
Options
View
Help
Applications
Processes
Performance
Networking
Users
Show processes from all users
End Process

Image Name	User Name	CPU	Mem Usage
svchost.exe	LOCAL SERVICE	00	1,836 K
svchost.exe	SYSTEM	00	3,000 K
taskmgr.exe	devadm	00	4,880 K
msdtc.exe	NETWORK SERVICE	00	6,444 K
odbcad32.exe *32	devadm	00	3,500 K
sqld.exe	SYSTEM	00	3,388 K
spoolsv.exe	SYSTEM	00	7,928 K
svchost.exe	SYSTEM	00	26,552 K
svchost.exe	LOCAL SERVICE	00	5,012 K
jusched.exe *32	devadm	00	3,484 K
svchost.exe	NETWORK SERVICE	00	6,016 K
svchost.exe	NETWORK SERVICE	00	4,956 K
csrss.exe	SYSTEM	00	5,268 K
VMwareTray.exe ...	devadm	00	4,052 K
svchost.exe	SYSTEM	00	3,696 K
wuauclt.exe	devadm	00	5,936 K
VMwareUser.exe ...	devadm	00	5,080 K
smss.exe	SYSTEM	00	636 K
gmacroer.exe *32	devadm	00	7,408 K

Processes: 34 | CPU Usage: 0% | Commit Charge: 180M / 1241M

Figure 8-1. Determining which ODBC Data Source Administrator is running.

2. On the User DSN or System DSN tab of the ODBC Data Source Administrator window, click the Add button. The Create New Data Source window opens and lists the currently available ODBC drivers.
3. In the Create New Data Source window, select xfODBC.
4. Click Finish.
5. In the xfODBC Setup window, enter the data source information in the following fields.



Data source name	Sample		
Description			
Appended to connect string			
Vortex driver	Net		
Host	mywin	Port	1958
User name	DBADMIN	Password	*******
Connect file	sodbc_sa		
Env. variables			

Driver maximums

Statements	256	DB cursors	64
Columns	256	Fetch buffer size	4096

Sort pages

Total	10000	In memory	1000

Merge buffer size

Max number of rows	10000

Copyright (c) 1996-2007, Trifox, Inc. California USA.
All Rights Reserved Worldwide.

Figure 8-2. Adding a DSN.

Data source name. Enter a descriptive name to identify the DSN. You must use valid MS-DOS characters for this. For example, do not use the backslash or parentheses characters.

Description. Enter a description of the DSN. This is optional.

Appended to connect string. Enter a string you want sent via SQL OpenNet to the server. This is optional and is typically used to define environment variables on the server under the direction of Synergy/DE Developer Support. To do this, use the following syntax (note the initial comma):

, *ENV_VAR=env_spec[, …]*

where *ENV_VAR* is an environment variable and *env_spec* is the definition of the environment variable. You can specify multiple environment variables by separating them with commas. Note the following:

- To use this field, you must use SQL OpenNet—i.e., select Net in the Vortex driver field.
- With a Windows server, use this field only when using **vtxnet2**.
- The SQL OpenNet server uses special characters as string delimiters: the at sign (@), colon (:), and exclamation point (!). In connect strings, each of these delimiters conveys a specific instruction to the SQL OpenNet processor and generally is not passed by the processor unless an identical character follows the first. If you are using an at sign, colon, or exclamation point in an environment variable definition, or at any other place in the string, you must use a duplicate at sign, colon, or exclamation point to ensure that the parser will interpret the statement correctly.

Vortex driver. Select Net for a client/server configuration. Select Genesis for a stand-along configuration. (The Net option instructs *xf*ODBC to use SQL OpenNet. The Genesis option instructs *xf*ODBC to connect to the database directly—i.e., without using SQL OpenNet.)

Note that to use Net, the SQL OpenNet server must be running on the host specified in the Host field.

Host. If you select Net for the “Vortex driver” field, enter the host name for the machine on which the database resides. Host names can be up to 64 characters long. Note the following:

- When using **vtxnet2** on Windows, the account for this user must have the “log on as a batch job” privilege. See “The vtxnetd and vtxnet2 Programs” in the “Configuring Connectivity Series” chapter of the *Installation Configuration Guide* for information on **vtxnet2** and **vtxnetd**.
- If the **-a** option is set for **vtxnetd** or **vtxnet2**, you must specify a user name and password after the host name:

  host_name([domain\]uid/pwd)

  where *uid* and *pwd* are the user name and password for an account on the host machine or, if *domain* is specified, an account on a domain (Windows only).
- By default the user and password are encrypted as they are sent across the network via SQL OpenNet. For information on changing encryption settings, see “SQL OpenNet Client Options in net.ini” on page 8-26.

Port. If the data files are on a remote machine, enter the port address for the computer on which the database resides. See “Port settings” on page 8-11 for more information.

User name. Enter the user name exactly as it is stored in the system catalog; this field is case sensitive. You can leave this field blank if you would like the end-user to be prompted for a user name when accessing a Synergy database.

By default the user name and password are encrypted as they are sent across the network via SQL OpenNet. For information on changing encryption settings, see "SQL OpenNet Client Options in net.ini" on page 8-26.

Password. Enter the password for the user exactly as it is stored in the system catalog; this field is case sensitive. Leave this field blank if you would like the end-user to be prompted for a password when accessing a Synergy database.

Connect file. Enter the connect file name. Connect files must be located in the directory specified by the GENESIS_HOME environment variable. In a client/server configuration, the connect file must be in the GENESIS_HOME directory on the server. You can leave this field blank if you would like users to be prompted to supply the connect file (which might be useful for testing). Note, however, that you must specify a connect file here in order to use the DSN with the Synergy/DE Data Provider for .NET.

Env. variables. Use this to define environment variables used by the *xf*ODBC driver on the *client*. For services that use the *xf*ODBC driver, such as web servers, this is the only place these environment variables can be set (unless you set them in the system environment and reboot the server). For example, for a web server in a client/server configuration, you can use this field to set any of the VORTEX_API_ environment variables, VORTEX_ODBC_CHAR, or VORTEX_ODBC_TIME. For a web server with a local database, you can use this field to set GENESIS_HOME as well.

To set environment variables in this field, use the following syntax:

ENV_VAR=env_spec[, ...]

where *ENV_VAR* is an environment variable to be set on the client, and *env_spec* is the definition of the environment variable. You can specify multiple environment variables by separating them with commas.

Statements. The number of logical cursors to allocate for the connection. (*xf*ODBC assigns a logical cursor to each SQL statement.) If this option is set to a value that is greater than the DB cursors setting, *xf*ODBC is able to cache cursors by mapping multiple logical cursors to a single database cursor. Because logical cursors require less memory than database cursors, this improves performance. For optimal performance, set this option to the maximum number of SQL statements that will be open concurrently. Valid values are 4 through 1024. The default is 256.

DB cursors. The number of database cursors to allocate for the connection. Valid values are 4 through 256. The default is 64. This should be set to a value that's less than Statements.

Columns. The maximum number of columns that can be returned for a query. Enter the number of columns of the largest table in the database. Valid values are 4 through 1024. The default is 256.

Fetch buffer size. The number of bytes to allocate for the prefetch buffer. The fetch buffer size is the size of a data transfer block. Valid values are 0 (which disables the prefetch buffer) and 1024 through 99999999. The default setting is 4096.

Total. The amount of disk space in pages (4,096-byte blocks) to allocate for temporary sort tables (work files) for each open cursor. The default is 10000. Valid values are 1000 through 99999999. On 32-bit systems, however, the limit may be 2 GB due to file system limitations.

⚠ The setting for Total must be greater than or equal to the setting for "In memory." The Total and "In memory" settings are used to generate a "SET OPTION SORTPAGES *totalpages memorypages*" command when the DSN is used to connect to a database. An application uses the sum of the memory specified for all concurrently opened cursors, and on Windows, **vtxnetd** uses the sum of memory allocated for all open cursors for every connected application. See "Notes on SORTPAGES" on page B-63 for more information.

In memory. The amount of internal memory in pages (4,096-byte blocks) to allocate for temporary sort tables (work files) for each open cursor. Valid values are 1000 through 999999. The default is 1000. This setting affects the performance of join queries and queries with ORDER BY clauses.

Max number of rows. The maximum number of rows that can be returned when a statement is optimized with multimerge. Multimerge optimizes SELECT statements that have one or more OR clauses by evaluating each side of each OR clause as a separate SELECT statement and then combining the results (in a multimerge optimization). Note that this works only when keys are available to optimize each side of each OR clause. This setting must be a positive numeric value from 100 to 65536.

- If the statement is *not* optimized with multimerge, this setting does not apply.
- To turn this setting off, use SET OPTION MERGESIZE. (See SET OPTION on page B-57.) You cannot use this dialog box to turn multimerge off.
- If the result set of an optimized statement is larger than this setting, an error is generated.

By default, multimerge is enabled (the default setting is 10000) because some applications automatically generate the kind of statements that a multimerge optimizes. For example, if you use Microsoft Access to issue a query that selects all of the columns in a table, Access generates a SELECT statement with a series of OR clauses that repeatedly specify key segments.

Note the following:

- In some cases, multimerge could impair performance.
- For each row allowed by this setting, *xf*ODBC uses six bytes of memory, so setting this to a large value doesn't generally affect performance.
- You can also set this option in an SQL statement by using SET OPTION MERGESIZE.

6. Click OK.

You can now use the DSN in an ODBC-enabled application.

Adding a non-sharable file DSN

1. Add a user DSN by following the steps in "Adding a user or system DSN" on page 8-5.
2. Use a text editor to create a file with the following text:

   ```
   [ODBC]
   DSN=dsn_name
   ```

   where *dsn_name* is the name of the user DSN.
3. Save this file to the directory where DSNs are stored for the ODBC Administrator. Give it a **.dsn** extension.

   You can now use the file DSN in an ODBC-enabled application.

Modifying a DSN

To modify an existing user, system, or non-sharable file DSN, do the following:

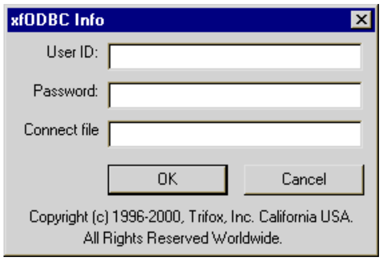
1. Open the ODBC Data Source Administrator utility. See step 1 in "Adding a user or system DSN" on page 8-5.
2. Select the DSN on the User DSN, System DSN, or File DSN tab. Then click Configure.

   When you click the Configure button, the xfODBC Setup window displays the current configuration.
3. Make any necessary changes to the fields in the xfODBC Setup window, and then click OK. See "Adding a user or system DSN" on page 8-5 for information on these fields.

If you're using the ODBC Data Source Administrator, changes made to non-sharable file DSNs are automatically applied to the corresponding user DSN. If you make a change to a non-sharable file DSN that resides on a network machine, the change is applied to the corresponding user DSN on the machine you use to make the change. The corresponding user DSNs on other clients must be modified separately.

Prompting the user for information

If you would prefer that users enter the user name, password, or connect file for the database, leave that information blank in the xfODBC Setup window. When the user attempts to access the Synergy data through a third-party application, the *xf*ODBC driver checks the DSN information and prompts the user for missing information. (Note, however, that to use the DSN with the Synergy/DE Data Provider for .NET, the DSN must specify a connect file.) For example, if you want users to enter their user name and password and the connect file name, leave the User ID, Password, and Connect file fields blank in the xfODBC Setup window. The users will be prompted for this information, as in figure 8-3.



Copyright (c) 1996-2000, Trifox, Inc. California USA.
All Rights Reserved Worldwide.

Figure 8-3. The xfODBC Info window.

Deleting a DSN

1. On the User DSN, File DSN, or System DSN tab of the ODBC Data Source Administrator, select the DSN you want to delete.
2. Click the Remove button.

Port settings

To make a connection with SQL OpenNet, the port setting for the client must match the port number for the SQL OpenNet server. For example, if **vtxnetd** is started on port 1990, the client must use port 1990 to connect to the SQL OpenNet server.

On Windows and UNIX *servers*, you can specify the port number in

- the vtxnet setting in the TCP/IP **services** file, which is in %windir%\system32\drivers\etc on Windows and /etc on UNIX. This is the default port setting, but it is used only when the port is not specified in the **vtxnetd** or **vtxnet2** start-up command.
- the **vtxnetd** or **vtxnet2** command line in **opennet.srv** (on Windows) or the **vtxnetd** command line in the **startnet** script (on UNIX). This overrides the **services** file setting.

On an OpenVMS *server*, the default (1958) is hard-coded, but you can override the default by setting a port number in the **vtxnetd** command line in **NET.COM**.

For more information on server-side port settings, see the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.

On *clients*, specify the port number in the DSN or, for a DSN-less connection, specify it with

- the vtxnet setting in the TCP/IP **services** file, on Windows (%windir%\system32\drivers\etc) and UNIX (/etc). For more information, see the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.
- a port setting in **net.ini**. If set here, this overrides the services setting. For more information, see "SQL OpenNet Client Options in net.ini" on page 8-26.

A quick way to ensure your port settings match is to use either the **synxfpng** utility (with the **-x** option) or the **vtxping** utility without specifying a port in the command line. If the connection is successful, the port settings match. For more information, see "The vtxping Utility" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide* and "The synxfpng Utility" in the "Configuring xfServer" chapter of *Installation Configuration Guide*.

Setting Runtime Data Access Options

*xf*ODBC has many options that enable you to control how *xf*ODBC behaves *as* it accesses data. To be effective, these options must be set before you connect to a database.

If you're using ADO.NET, also see "Time columns and ADO.NET" on page 9-20.

Note that third-party applications used to access Synergy data usually have options, such as query time-out, that use underlying ODBC calls. Additionally, note the following:

- For information on other options that affect how data is accessed by changing the way the system catalog is generated, see "Setting catalog generation options" on page 3-23.
- For information on how to set environment variables, see "Setting environment variables" on page 3-30.
- For information on SQL OpenNet options that affect *xf*ODBC, see "SQL OpenNet Client Options in net.ini" on page 8-26.

Formats for returned dates and times

Date/time (timestamp), date, and time columns are returned with the following formats and default data types:

Database column	Format of returned data	Returned data type
timestamp	YYYY-MM-DD HH:MI:SS	SQL_TTYPE_IMESTAMP
date	YYYY-MM-DD	SQL_TYPE_DATE
time	HH:MI:SS	SQL_TYPE_TIME (System.TimeSpan for ADO.NET; see "Time columns and ADO.NET" on page 9-20)

You can use the SQL command TO_CHAR to change the display format. For information, see TO_CHAR on page B-39.

Changing the data type returned for timestamp columns

You can change the data type returned for timestamp columns by setting the VORTEX_ODBC_DATETIME environment variable to the integer value for the ODBC data type you want returned. The default is SQL_TIMESTAMP (which is 11). For example, the following instructs *xf*ODBC to return SQL_CHAR data for timestamp columns:

```
VORTEX_ODBC_DATETIME=1
```

If you set VORTEX_ODBC_DATETIME, set it in the system environment. For client/server configurations, set it on the client.

Changing the data type returned for time columns

You can change the data type returned for time columns by setting VORTEX_ODBC_TIME to one of the following integer values for the ODBC data type you want returned:

10	Describe time columns as SQL_TIME.
11	Describe time columns as SQL_TIMESTAMP.

This is useful when using ADO.NET, which retrieves SQL_TIME columns as System.TimeSpan, a .NET data type that represents a time interval, which is generally more difficult to use than a specific time. (See "Time columns and ADO.NET" on page 9-20.) Setting VORTEX_ODBC_TIME to 11, however, enables you to get time columns as timestamp values. Note the following:

- If VORTEX_ODBC_TIME is not set (which is the default when the Synergy/DE Data Provider for .NET is not in use), the *xf*ODBC driver describes time columns as SQL_TIME.
- When you use the Synergy/DE Data Provider for .NET, VORTEX_ODBC_TIME is automatically set to 11, and you cannot override this setting.
- SQL_TIMESTAMP values have both a date and a time, so to create a SQL_TIMESTAMP value, *xf*ODBC includes the date 1-1-1.

If you set VORTEX_ODBC_TIME, set it in the system environment. For client/server configurations, set it on the client. Note that for a service, such as IIS or SQL Server, you must reboot after setting VORTEX_ODBC_TIME, unless you set it in the DSN (see "Env. variables" on page 8-8).

Converting dates returned without centuries

When a system catalog is generated, each date field that doesn't include a century (in other words, each date with a YY year, rather than YYYY) is formatted as a date with a rolling (RR) century. (See "Date and time fields" on page 3-15.) Then, when the *xf*ODBC driver retrieves a date with a rolling century, it converts it to a date with a century (a YYYY date). The century part of the date is determined by the SYNCENTURY environment variable:

- If the year for a retrieved date is between 0 and the value of SYNCENTURY, *xf*ODBC uses the current century (20*xx*).
- If the year is between SYNCENTURY (inclusive) and 99, *xf*ODBC uses the previous century (19*xx*).

The default for SYNCENTURY is 50. If SYNCENTURY is not set or is set to a negative value, 50 is the cutoff year.

For standalone configurations, set SYNCENTURY in the connect file or in the environment. For client/server configurations, set it in the connect file on the server.

If the century for dates whose years fall between SYNCENTURY and 99 is set to the current century, most likely the SODBC_NO`ROLL environment variable was set when the system catalog was generated. This environment variable was used for Y2K conversions and is no longer necessary. However, if it was set when the system catalog was generated, *xf*ODBC ignores the SYNCENTURY setting, which results in two-digit years being stored as YY years rather than RR (rolling) years.

Treating invalid dates as null data

If a database has invalid date data, SELECT statements fail. This occurs even if the columns with invalid dates are not referenced in the SELECT statement. You can, however, instruct the *xf*ODBC driver to treat invalid dates as null by setting the convert_error option to yes. See "Setting the convert_error Option" on page 5-4 for information.

Masks for dates and times in SQL statements

When you write a date/time, date, or time column to a database, *xf*ODBC must convert the data to the *xf*ODBC driver's internal date/time format. This is true if you create the SQL statement and if an ODBC-enabled application creates the SQL statement. The *xf*ODBC driver uses four masks to interpret date/time, date, and time columns. By default, these are YYYY-MM-DD HH:MI:SS, YYYY-MM-DD, HH:MI:SS, and YYYY_MM_DD HH:MI:SS.UUUUUU. *xf*ODBC first attempts to use the first mask (YYYY-MM-DD HH:MI:SS). If it's unable to use this, it attempts to use the second mask (YYYY-MM-DD), and so on. Dates and times specified in SQL statements must have dates and times that match one of the masks. You can, however, modify the masks. If you want the *xf*ODBC driver to accept other date and time formats, use SET OPTION DATETIME (see SET OPTION on page B-57).

Setting the base date for Julian day conversions

When you enter a date into a field with the JJJJJ format, *xf*ODBC stores the date as the difference between the date you entered and the value of SYNBASEDATE. By default, SYNBASEDATE is set to 1752-09-14 (14 September 1752), but you can change this value. Note the following:

- When setting SYNBASEDATE, use the YYYY-MM-DD format.
- For stand-alone configurations, set SYNBASEDATE in the connect file, or in the environment. For client/server configurations, set it in the connect file on the server.
- SYNBASEDATE does not affect the way system catalogs are generated. (It is not used by DBA or **dbcreate**.) This variable is used by the *xf*ODBC driver when it accesses data.

Note the following:

- If you’ve used *xf*ODBC to modify date data in the database, do *not* change SYNBASEDATE. If you do, the dates in your database will be corrupt. Changing SYNBASEDATE changes part of the equation used to store and retrieve dates.
- If you use the Julian functions %JPERIOD or %NDATE, do *not* set SYNBASEDATE. It must be set to its default value (1752-09-14). If it’s set to any other value, %JPERIOD and %NDATE will retrieve and store dates using a different equation than *xf*ODBC, which will corrupt the dates in your database. (For information, see %JPERIOD and %NDATE in the “System-Supplied Subroutines and Functions” chapter of the *Synergy DBL Language Reference Manual*.)

Recognizing the MCBA deleted-record characters

The SODBC_MCBA environment variable enables you to instruct *xf*ODBC to skip records that contain the MCBA deleted-record characters—four right brackets (]]]]) at the beginning or end of a record. Note the following:

- By default, SODBC_MCBA is not set; *xf*ODBC does *not* skip records that contain the MCBA deleted-record characters.
- To instruct *xf*ODBC to skip records that contain the MCBA deleted-record characters, set the SODBC_MCBA environment variable to any value.
- For stand-alone configurations, set SODBC_MCBA in the connect file or in the environment. For client/server configurations, set it in the connect file on the server.
- The SODBC_MCBA setting does not affect the way system catalogs are generated. This is used by the *xf*ODBC driver when it accesses data. It is not used by **dbcreate** or DBA.

Changing the way *xf*ODBC describes strings

*xf*ODBC passes and, by default, describes strings as SQL_VARCHAR (that is, with trailing spaces removed). You can, however, instruct the *xf*ODBC driver to describe strings as SQL_CHAR (though they are always passed as SQL VARCHAR) which was the default behavior in *xf*ODBC versions prior to 8.3. Note the following:

- If VORTEX_ODBC_CHAR is set to 12 or is not set (the default), the *xf*ODBC driver passes *and* describes strings as SQL_VARCHAR.
- If VORTEX_ODBC_CHAR is set to 1, the *xf*ODBC driver passes strings as SQL_VARCHAR, but describes them as SQL_CHAR.

If you set VORTEX_ODBC_CHAR, set it in the system environment. (For a service, such as IIS or SQL Server, you must either reboot after setting VORTEX_ODBC_CHAR or set it in the DSN. See “Env. variables” on page 8-8.) For client/server configurations, set it on the client.

Note that VORTEX_ODBC_CHAR is used by the *xf*ODBC driver when it sends data to the application. It does not affect the way system catalogs are generated.

Creating a file for query processing options

The GENESIS_INITSQL environment variable enables you to specify a file that contains predefined SET OPTION commands. (This includes all SET OPTION commands except DATETIME, SORTPAGES, and TMPINDEX.) For information on SET OPTION commands, see SET OPTION on page B-57. The SQL statements in this file are executed each time a connection is made to the driver.

Note the following:

- Each option must be on a separate line in the file, and each line must have the following format: set option *option*. For example:

  ```
  set option logfile 'vtx4.log'
  set option tree on
  set option error on
  ```

- The GENESIS_INITSQL environment variable must be set to the path and filename of the options file and must be set in the environment. For client/server configurations, it must be set in the environment on the server or in the **opennet.srv** file (Windows only).

System Catalog Caching

Note the following:

- It is important that you carefully follow the instructions in this section. If you don't, system catalogs will not be loaded and unloaded correctly, and shared memory will not be allocated and freed correctly.
- On UNIX, only experienced system administrators should attempt to use system catalog caching. We've tested the procedures documented here on our systems, but don't guarantee that they'll work on all systems. Shared memory mechanisms vary from one operating system to another.

You can improve performance by instructing *xf*ODBC to cache system catalogs. When a system catalog is cached, the *xf*ODBC driver consults the cached catalog in memory rather than rereading the catalog from disk for each new command.

To cache system catalogs,

1. Add a **syngenload** command to one of the following files (which are in the synergyde\connect directory):

   - **opennet.srv** on Windows
   - **startnet** on UNIX
   - **STARTNET.COM** on OpenVMS

   As distributed, these files include a sample **syngenload** command that's been commented out. To use the command, uncomment the line and change the user name, password, and connect file. See "Using syngenload" on page 8-19 for information on the syntax.

   If necessary, you can run **syngenload** from the command line (see "Running syngenload from the command line (Windows and OpenVMS)" on page 8-20) . However, we recommend specifying the **syngenload** command in one of the above files, which are read each time the Synergy/DE OpenNet Server service (**SynSQL**) is started.

2. On UNIX or OpenVMS, do one of the following:

   - On UNIX, make sure the following line is uncommented:

     ```
     rm $GENESIS_HOME/synodbccache.dat
     ```

   - On OpenVMS, make sure all three of the vortexipc lines in **STARTNET.COM** are uncommented:

     ```
     $vortexipc:==$CONNECTDIR:vortexipc.exe
     $vortexipc /d
     $vortexipc /c 1000 0 0
     ```

Note the following:

- The **SynSQL** service must be running *before* you load a system catalog. For information on starting the **SynSQL** service, see the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.
- You can load more than one system catalog at a time by issuing multiple **syngenload** commands.
- On Windows, when the **SynSQL** service is stopped, all loaded system catalogs are unloaded.
- To free shared memory on UNIX and OpenVMS, you must use the **syngenload -u** command to unload cached system catalogs. On UNIX and OpenVMS, stopping the SQL OpenNet server does *not* unload cached system catalogs.
- Do not delete the **synodbccache.dat** file unless instructed (as in "Correcting other caching problems" on page 8-25, for example). This file is generated by *xf*ODBC when a system catalog is cached. Deleting this file will corrupt the cache subsystem. Use the **-u** option for **syngenload** to unload a system catalog.

Using syngenload

To load (cache) a system catalog into memory or unload a system catalog from memory, use **syngenload** with the **-l** or **-u** option. The **syngenload** program is in the connect directory and has the following syntax:

```
syngenload option
```

where *option* is one of the following:

/?	Display command line help for **syngenload**.
-l "*user/pswd*" *connect_file*	Cache the system catalog stored in the directory specified in the dictsource line of *connect_file*. *User* must be one of the users defined in the system catalog, and *pswd* must be the password for the user. The quotes around *user*/*pswd* are required.
-m *user/pswd connect_file base_size*	Display shared memory segment addresses. See "Adjusting the shared memory subsystem settings" on page 8-23 for more information.
-u *connect_file*	Unload the system catalog stored in the directory specified in the dictsource line of *connect_file*.

For example, the following command loads the system catalog specified by the **sodbc_sa** connect file using the default DBADMIN administrative user:

```
syngenload -l "DBADMIN/MANAGER" sodbc_sa
```

The next example unloads the system catalog specified by the **sodbc_sa** connect file:

```
syngenload -u sodbc_sa
```

When loading a system catalog, **syngenload** opens the system catalog files—just like an ODBC-enabled application. So if an ODBC-enabled application can’t open the system catalog, **syngenload** won’t be able to either.

Note the following:

- If you modify a system catalog while it’s cached, the copy on disk will be updated, but the cached version won’t be updated until it’s reloaded.
- In general, to unload a system catalog, you must be the user that loaded it. On UNIX, however, the root account can unload any system catalog.
- If **syngenload** is started by the Synergy/DE OpenNet Server service (**SynSQL**) on Windows, errors are written to the Windows event log.
- If you’re running **syngenload** from the command line, errors are written to the screen.
- For security reasons, the system catalog files that store user and group information (**SODBC_USERS.*** and **SODBC_GROUPS.***) aren’t cached. *xf*ODBC reads these from disk.

Running syngenload from the command line (Windows and OpenVMS)

If necessary, you can run **syngenload** from the command line for short-term testing purposes. However, we recommend putting the **syngenload** command in one of the files listed in step 1 on page 8-18. These are read each time the **SynSQL** service is started.

Before running **syngenload** from the command line on Windows or OpenVMS, you must set the VORTEX_SHM_FILE environment variable to **synodbccache.dat** in the connect\synodbc directory. (On UNIX, this is set for you.) For example:

```
set VORTEX_SHM_FILE=%GENESIS_HOME%\synodbccache.dat
```

WIN

Note that on Windows, you must stop and then restart **vtxshm** before running **syngenload**:

1. If **vtxshm** is running (check this in Windows Task Manager), stop it by doing one of the following:
   - In Windows Task Manager, select the **vtxshm.exe** entry, and then click End Process.
   - Enter the following at the command line:

     ```
     vtxshm -t
     ```

Vtxshm is a part of the *xf*ODBC caching mechanism and is automatically started by the **SynSQL** service. However, when **vtxshm** has been started by **SynSQL**, it isn’t available to **syngenload** if you start **syngenload** from the command line.

2. Start **vtxshm** from the command line:

```
start /B vtxshm -s
```

You can now run **syngenload** from the command line.

Using logging to determine if a system catalog is cached

To determine if a system catalog is cached or has been unloaded from shared memory, use Synergy driver logging or Synergy DBMS logging.

Synergy driver logging

Synergy driver logging lists the path and name of the shared memory file and lists errors encountered while attempting to use shared memory. To use Synergy driver logging, add the following lines to your connect file:

```
logfile file_spec
loglevel 1
```

where *file_spec* is the path and filename of the log file you want to create. For example, here's the sample connect file with these lines:

```
dictsource "C:\Program Files\Synergex\SynergyDE\connect\synodbc\dict\"
datasource ";C:\\Program
Files\\Synergex\\SynergyDE\\connect\\synodbc\\dat;"
XFDBTUT=C:\Program Files\Synergex\SynergyDE\connect\synodbc\dat
logfile c:\temp\connect.log
loglevel 1
```

Note the following:

- Add the logfile and loglevel lines to the connect file only *after* the **sodbccache.dat** file has been created. This file is created with the first **syngenload -l** command of the session. If you add **syngenload** commands to **opennet.srv**, **startnet**, or **STARTNET.COM**, add the logfile and loglevel lines to your connect file *after* the SQL OpenNet server has been started.
- The error "SHARED MEMORY error: Cannot attach to shared memory: Attempt to access invalid address" indicates that you need to set the VORTEX_SHM_BASE environment variable. Set it to address 80000000 (which on 32-bit Windows systems is 00000008, on 64-bit Windows systems is 0000000800000000, and so forth).
- The error "SHARED MEMORY error: Cannot open c:\...\syodbc\synodbccache.dat, No such file or directory" indicates that you need to set the VORTEX_SHM_FILE environment variable to the **synodbccache.dat** file. (See "Running syngenload from the command line (Windows and OpenVMS)" on page 8-20.)

Synergy DBMS logging

Synergy DBMS logging tells you if the system catalog files are open and logs reads to the files. (These files, whose names begin with "GENESIS_", are listed in "System catalog" on page 1-5.) If the system catalog files are read *as* you access your database with *xf*ODBC, the system catalog is *not* cached. (If the system catalogs are cached, the files will be read to create the cache, but not to access data.) For information on Synergy DBMS logging, see "Synergy DBMS logging" on page 11-8.

Troubleshooting system catalog caching

Cannot allocate shared memory

On Windows, if you get a "cannot allocate shared memory" error, the Synergy/DE OpenNet Server service (**SynSQL**) probably isn't running. If you're having trouble starting **SynSQL**, check the Windows event log for information. The **sqld** program has an **-l** option that writes more detailed information to the Windows event log. See "The sqld program" in the Windows section of the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide* for information.

On OpenVMS, this error may indicate that one or more of the vortexipc lines in **STARTNET.COM** is commented out. See step 2 on page 8-18 for information.

Cannot attach to shared memory (Windows)

If you attempt to run **syngenload** from the command line, you may get the following error: "ERROR: SHARED MEMORY error: Cannot attach to shared memory: The operation completed successfully." This may indicate that **syngenload** cannot access the **vtxshm** program. If you see this, follow the instructions in "Running syngenload from the command line (Windows and OpenVMS)" on page 8-20.

Invalid parameter or argument (UNIX)

If you encounter an "Invalid Parameter" or "Invalid Argument" error when caching a system catalog on a UNIX system, you will need to adjust shared memory subsystem settings. The **syngenload** program uses this subsystem when you instruct it to load a system catalog. There are two shared memory subsystem settings for caching:

- To determine where to start the cache, **syngenload** uses the VORTEX_SHM_BASE environment variable, if set, for the cache's base address. If this environment variable is not set (which is the default), the operating system determines which address the cache uses.
- To determine the amount of space for each shared memory segment used by the cache, **syngenload** reads the **trim.ini** file. As distributed (in the $TRIM_HOME/lib directory), the **trim.ini** file specifies a 500K shared memory segment size.

Before you make any adjustments, however, try caching your system catalog by using **syngenload** from the command line. If you get a “Invalid Parameter” or “Invalid Argument” error, follow the steps in “Adjusting the shared memory subsystem settings” below. If you don’t get an error, do the following:

1. Set Synergy driver logging. (See “Synergy Driver Logging” on page 5-5.)
2. Use an ODBC-enabled application to connect to the database. Then check the file created by Synergy driver logging to see if there is an “Invalid Parameter” or “Invalid Argument” error after the **synodbccache.dat** entry. If there is, follow the steps in “Adjusting the shared memory subsystem settings” below. If you don’t see an error after this entry, caching is working correctly.
3. Turn off Synergy driver logging.

Adjusting the shared memory subsystem settings

For most UNIX systems, you’ll need to adjust the shared memory segment size, but you may not need to set the base address. On Linux systems, however, it’s almost certain that you’ll need to set the base address as well. In either case, follow the instructions below.

1. Reboot your machine.
2. Make sure there is no **synodbccache.dat** file in the $CONNECTDIR/synodbc directory. If it’s there, delete it.
3. Estimate the size of your cache by totaling the size (in bytes) of all the **GENESIS*.is1** files in your system catalog. Then add 100,000 to the sum. You’ll use this figure as the base size (*base_size* in step 4 and step 6) for shared memory segment settings.
4. Use the **tman** utility, a memory analyzer, to determine if there is enough shared memory space. Use the following syntax:

   `tman -n` *base_size base_size*

   If this causes an error, ask your system administrator to re-configure the shared memory subsystem to allow a maximum size that’s at least twice as large as *base_size*. (This usually requires a system reboot.)
5. Divide *base_size* by 1,024. Then open **trim.ini** in a text editor and set shmem_seg_size to the result of the calculation. (The shmem_seg_size setting specifies a value in kilobytes.)

   Note that the goal in setting shmem_seg_size is to create a shared memory segment that’s large enough for the entire system catalog, which is very important if you also set VORTEX_SHM_BASE. When this environment variable is set, the system catalog must be in a single shared memory segment. (If it’s split between two or more segments, they must be contiguous—something we can’t control.)

6. Use **syngenload** to cache your system catalog. If you get an “Invalid Parameter” or “Invalid Argument” error, try increasing the shmem_seg_size setting in **trim.ini** by 500. Then go back to step 4, increasing *base_size* by 512,000. If you still get the error, use **syngenload** with the following syntax:

   ```
   syngenload -m user/pwd connect_file base_size
   ```

   This will display three shared memory segment addresses.

   ```
   VORTEX_SHM_BASE values for size 512000
   SID: 1376262, VORTEX_SHM_BASE: 00303040 Native: 40303000
   SID: 1409031, VORTEX_SHM_BASE: 00003840 Native: 40380000
   SID: 1441800, VORTEX_SHM_BASE: 00D03F40 Native: 403FD000
   ```

   Now set the VORTEX_SHM_BASE environment variable to the first address (in the example above, this is 00303040), and start again at step 4. (See VORTEX_SHM_BASE in the “Environment Variables” chapter of *Environment Variables & System Options* for information on setting this environment variable.)

   Note that the goal in setting VORTEX_SHM_BASE is to set the base address to one that’s high enough to make the cache available to all programs that use it. (For example, if you don’t set this environment variable and the Synergy runtime is loaded into memory, the address may not be available; the runtime may have allocated it for something else before the cache is attached.) If you find in the final step that the first shared segment address listed isn’t high enough, set VORTEX_SHM_BASE to the second address listed—then the third if the second doesn’t work.

7. Check the **synodbccache.dat** file to ensure that the shared memory ID is the same for all entries. (This file is generated in the $CONNECTDIR/synodbc directory when a system catalog is cached.) Check this by comparing values in the second column (the shared memory ID column) of the **synodbccache.dat** file. All entries in the second column must have the same value. For example:

   ```
   SODBC_SA_GENESIS_TABLES 1507332 00000008 65536 0 5048
   SODBC_SA_GENESIS_COLUMNS 1507332 00000008 65536 5048 42108
   SODBC_SA_GENESIS_INDEXES 1507332 00000008 65536 47156 7768
   SODBC_SA_GENESIS_XCOLUMNS 1507332 00000008 65536 0 12004
   ```

   (The third column lists base addresses. The sixth column lists the size of cached entries.)

8. If the shared memory ID is not the same for all entries, the cache has been loaded into more than one segment, which may cause errors. In this case, add the total size of the generated cache (the sum of the values in the sixth column of the **synodbccache.dat** file), unload the cache using **syngenload**, and start again at step 4.

9. Once you've made it to this step, **syngenload** should be able to load the system catalog correctly. But we need to find out if the Synergy driver can access the cache. To do this, set Synergy driver logging, use an ODBC-enabled application to access your Synergy database, and then check the log. If you see an "Invalid Parameter" error after the **synodbccache.dat** entry, you need to use a higher address for shared memory; go back to step 6. If you don't get this error after the **synodbccache.dat** entry, caching is working correctly.

10. Turn off Synergy driver logging.

Viewing and removing shared segments

You can use the UNIX **ipcs** utility to view and remove shared memory segments that are in use. For example, if you've deleted the **synodbccache.dat** file, you can use **ipcs** to remove shared memory segments that are still allocated for caching. The second column lists shared memory IDs (which correspond to the shared memory IDs listed in **synodbccache.dat**).

```
------ Shared Memory Segments --------
key        shmid     owner    perms    bytes     nattch     status
0x00000000 196608    maryw    644      65536     0
0x00000000 229377    maryw    644      65536     0
0x00000000 491522    maryw    644      65536     0
0x00000000 524291    maryw    644      65536     0
0x00000000 1507332   root     644      10240000  0
```

Correcting other caching problems

If you find that system catalogs are not loading or unloading correctly,

- on Windows, do the following:
  1. Stop the Synergy/DE OpenNet Server service (**SynSQL**). (See "Stopping and removing SQL OpenNet" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.)
  2. Open the Windows Task Manager and end the task named **vtxshm** if it exists.
  3. Delete the **synodbccache.dat** file in your GENESIS_HOME directory.
  4. Follow the caching instructions from the beginning.
- on UNIX and OpenVMS, do the following:
  1. Delete the **synodbccache.dat** file (in your GENESIS_HOME directory).
  2. Reboot the system.
  3. Follow the caching instructions from the beginning.

SQL OpenNet Client Options in net.ini

The **net.ini** file enables you to specify an encryption key for the SQL OpenNet client, specify time-outs, instruct SQL OpenNet to return error codes for communication errors, and set environment variables on the server, as well as other settings for the SQL OpenNet client. When you install Connectivity Series, the installation creates a default **net.ini** file with default settings (including a default encryption setting). To change settings in **net.ini**, use a text editor. Note the following:

- We recommend that you don't change any **net.ini** setting except key_connect.
- The **net.ini** file must be on the client in the lib subdirectory of the directory specified by the VORTEX_HOME environment variable. The Connectivity Series installation (Windows), **setsde** (UNIX), or SYS$MANAGER:CONNECT_STARTUP.COM (OpenVMS) sets this environment variable to the connect\synodbc directory. Do not set this environment variable to another directory, and note that if you install both 32-bit and 64-bit versions of Connectivity Series on the same 64-bit Windows machine, the last version installed sets VORTEX_HOME. See "VORTEX_HOME" on page A-7 for more information.
- Make sure the **net.ini** file has no control characters. These cause "invalid integer" errors when connecting to the database.
- The **net.ini** file is not overwritten when you upgrade Connectivity Series, nor is it removed when you uninstall. We distribute a file named **net_base.ini** (also located in the connect\synodbc\lib directory), which contains default settings and can be used as a reference.

SQL OpenNet Client Options

Option	Description/syntax
hostenv0	Specifies a comma-delimited list of environment variables to be passed to and set on the server. Use the following syntax: `hostenv0` *`var_name=var_spec[,var_name2=var_spec2,...]`*
key_connect	Specifies a key for the algorithm used to encrypt user names and passwords for the database and for the host (if **vtxnetd** or **vtxnet2** is also started with the **-k** option). This encrypts user names and passwords being sent across the wire. Use the following syntax: `key_connect` *`n`* where *n* is any number between 1 and 2147483647. Note that *n* must be set to the same value on both the client, where it is set with this **net.ini** option, and the server, where it is set with the **-k** option on the **vtxnetd** or **vtxnet2** command line. See "The vtxnetd and vtxnet2 Programs" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.

SQL OpenNet Client Options (Continued)

Option	Description/syntax
packetsize	Sets the minimum network packet size used by SQL OpenNet. The default is 8192 bytes. This option defines a minimum size for an aggregate buffer, which is a buffer created when data for multiple network packets needs to be sent to the client. This reduces network traffic by combining packets and sending them as a unit with the specified minimum size. To set this, use the following syntax, where *size* is the size in bytes: `packetsize` *size* Note that changing the default packet size *may cause performance problems*. If you are using a WAN (wide area network), you may want to change this value to reduce load on the network. The packet size used by the SQL OpenNet server is set by the packetsize setting in the **net.ini** file on the client.
port	Sets the communication port number. This defaults to the **vtxnet** setting in the **services** file. For information, see "Specifying the port number" in the Windows section of the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*. `port` *port_number*
read_timeout	Specifies how long (in seconds) SQL OpenNet should wait for a read operation to complete. By default this is set to 0, which prevents a time-out. `read_timeout` *time*
return_errno	Instructs SQL OpenNet to return an operating system error code (rather than -1) if there's a communication error. By default return_errno is set to no. `return_errno yes|no`
write_timeout	Specifies how long (in seconds) SQL OpenNet should wait for a write operation to complete. By default this is set to 0, which prevents a time-out. `write_timeout` *time*

The following example **net.ini** file sets the encryption key to 6541, sets the packet size to 1300, sets the port to 1990, instructs SQL OpenNet to return error codes for communication errors, sets the read time-out to 60 seconds, sets the write time-out to 60 seconds, and sets the ENV1 and ENV2 environment variables on the server.

```
rem           SQL OpenNet init file
key_connect 6541
packetsize 1300
port 1990
return_errno yes
read_timeout 60
write_timeout 60
hostenv0 ENV1=c:\data,ENV2=c:\data2
```

Part 3: Accessing Data

This section contains information for developers and end-users. It explains how to access your database with third-party, ODBC-enabled applications.

9

Accessing a Synergy Database

The Basic Steps

Once you've generated a system catalog, created a connect file, and created a DSN, you can access your Synergy data from an ODBC-enabled application such as Microsoft Word or Crystal Reports. (There's more involved to accessing Synergy data in a .NET environment. See "Accessing Synergy Data in a .NET Environment" on page 9-10 for information.)

Follow these basic steps:

1. Open the third-party, ODBC-enabled application.
2. In the third-party application, choose the DSN for the Synergy database.

   The third-party application calls the ODBC Driver Manager, which in turn uses the DSN information to call the *xf*ODBC driver. The *xf*ODBC driver uses the DSN to locate the connect file. (See "How Third-Party Applications Use xfODBC" on page 1-11.)
3. If necessary, enter information in the log-in window (xfODBC Info).

You can enter an entire connect string in the User ID field of the xfODBC Info window. If you do, use the following syntax:

user_name / *password* / *connect_file*

Every ODBC-enabled application has its own procedure. See your application's documentation for details. See "Examples" on page 9-33 for step-by-step examples that use third-party applications to access Synergy data.

See "Appendix B: SQL Support" for information on SQL commands that *xf*ODBC supports and *xf*ODBC limitations when updating a database.

To update data in a Synergy database, we strongly recommend using a Synergy application that is designed to efficiently maintain database integrity. If you use an ODBC-enabled application to write to a Synergy database, you may run into record-locking issues. For information, see the "Statements that Modify Data" on page B-46.

Third-Party Software Requirements

The following are the general requirements for access with *xf*ODBC. For information on requirements for stand-alone and client/server configurations, see "xfODBC requirements and installation" on page 1-8.

Software	**Requirements**
ADO.NET with .NET Framework Data Provider for ODBC	.NET 4.0 Visual Studio 2010 or higher
ADO.NET with Synergy/DE Data Provider for .NET	See "System requirements for the Synergy/DE Data Provider for .NET" on page 9-11.
Crystal Reports	Version 9 or higher (version 10 is recommended)
Lotus Approach	Lotus SmartSuite Millennium 9.x
Microsoft Office	Office XP or higher with all Office updates Jet 4 with Service Pack 8 or higher
Visual Studio[a b]	Visual Studio 2010 SP1 Visual Studio 2012 Update 2

a. If you're using ADO.NET, see the ADO.NET rows above for Visual Studio requirements.

b. We do not recommend installing *xf*ODBC to run in a shared configuration when you are running Visual Studio on the client machine for *xf*ODBC development. However, if you do have this configuration, you must change your .NET security permissions on the client machine to permit access to the assembly **xfODBCSpecialization.dll**, located in the SynergyDE\connect directory on the shared machine. This assembly enables *xf*ODBC to work with Visual Studio wizards.

Troubleshooting Data Access

If you have problems accessing data from an ODBC-enabled application, follow the instructions in this section. If you're caching the system catalog, also see "Troubleshooting system catalog caching" on page 8-22.

For information on troubleshooting problems generating a system catalog, see "Errors and Troubleshooting" on page 4-14.

Review the repository definitions

If you are having problems accessing data, or if the data in the applications does not display as you expect it to, the problem might be with the way the data was defined in your repository. For information on setting up a repository for *xf*ODBC, see "Setting Up a Repository" on page 3-2.

- Have you opened up the repository files in S/DE Repository to verify that the structures are assigned to files and that the structures are set up correctly?
- Have you checked the repository definitions for all fields to verify that they are defined as expected?
- Are you able to open the data files in a Synergy application?

Verify environment and environment variables

Because environment variables are crucial in enabling *xf*ODBC to locate files, it's important to verify them.

- If you used environment variables in the repository Open filename field, have you also defined them in the connect file, in an environment setup file, or in the environment?
- Is GENESIS_HOME set correctly? For information, see "Specifying the connect file location (GENESIS_HOME)" on page 3-19.
- Have you set variables in an environment setup file? If so, does SODBC_INIFIL point to the directory where the environment setup file is located? For stand-alone configurations, is SODBC_INIFIL set in the environment or in the connect file? For client/server configurations, is it set in the environment on the server?
- Have you set environment variables in the connect file? If so, make sure SODBC_INIFIL is not set in the environment. *xf*ODBC uses environment variables set in the connect file only if SODBC_INIFIL is not set or is set in the connect file.

Verify file locations

- Is the connect file located in the directory that the GENESIS_HOME environment variable is set to? If not, did you specify the filename *and* path for the connect file when you configured the DSN?
- If you're using the datasource line in the connect file to specify the location of your Synergy data files, are all of the data files stored in the directory specified on the datasource line?
- Are the system catalog files stored in the directory specified on the dictsource line in the connect file? See "System catalog" on page 1-5 for information on these files.

Verify the system catalog

- Have you used the **fcompare** utility to compare the system catalog to the data files? For information, see "Comparing the system catalog to a database" on page 6-32.
- Have you generated the system catalog using the conversion setup file as input?
- Have you checked the conversion setup file to verify table locations, inclusion, and access?
- Have you made changes to the conversion setup file since generating the system catalog? If so, have you regenerated the system catalog?

Use DBA to verify the system catalog

You can use the *xf*ODBC Database Administrator (DBA) program to verify that the data definitions were converted correctly.

- Have you used the Verify option in DBA to ensure that DBA is able to find and read all of the tables, columns, tags, and indexes, and that DBA does not report errors reading them?
- Have you carefully compared the Repository data with each of the table attributes and column definitions as they appear in DBA?
- Have you ensured that the table access levels are set appropriately?
- Have you verified that your users are assigned to groups with appropriate access levels?

Verify encryption

If you have verified environment variables and file locations and you are still unable to connect, check the encryption settings on the client (in **net.ini**) and on the server (in the **vtxnetd** or **vtxnet2** command line). Make sure these settings match, or for testing, remove the encryption settings in both locations and see if you can connect. Additionally, make sure the **net.ini** file is in the directory specified by VORTEX_HOME. Mismatched encryption settings, along with the inability to access encryptions settings in **net.ini**, can cause a variety of errors when you try to connect, including "invalid connect syntax," "invalid user ID and/or password," and "invalid DSN" errors. For more information, see "SQL OpenNet Client Options in net.ini" on page 8-26.

Verify the log-in

- Using DBA, verify user names and passwords. When entering the connect string, did you type the user name and password *exactly* as stored? They are case sensitive.
- To debug your initial connection, use Synergy DBMS logging. See "Error Logging" on page 11-2 for more information.

TIP On Windows you can test a network connection with **vtxping** (or **synxfpng** with the **-x** option). See "xfODBC: testing the network connection for client access" in the Windows section of the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.

Open your database with a third-party application

Did you generate a system catalog from the Synergy sample database, and were you able to access this data successfully with an ODBC-enabled application?

Note that Synergex does not provide support for other ODBC-enabled applications, and resolving problems particular to a specific third-party application is beyond the scope of this user's guide.

If you are still encountering problems…

If you have followed all of the above troubleshooting steps and are still unable to access your database, turn on the ODBC logging options and use the tracing feature. See "Error Logging" on page 11-2 for instructions. You may need to call Synergy/DE Developer Support for an interpretation of the log files you generate. For more information, see "Product support information" on page x.

Other sources of information

- KnowledgeBase

  The Synergy/DE KnowledgeBase is available on the Synergex website for customers who have purchased a support agreement. The KnowledgeBase includes a wealth of troubleshooting information and answers to frequently asked questions that may help you solve a problem you have encountered. To purchase a support agreement, call your Synergy/DE account manager.

- Release notes

  The **REL_CONN.TXT** release notes distributed with Synergy products include the latest information about new features and restrictions in *xf*ODBC.

Record Locking and Transactions with *xf*ODBC

For updating Synergy databases, we strongly recommend using a Synergy application that's designed to efficiently maintain database integrity. See "Statements that Modify Data" on page B-46 for more information.

*xf*ODBC supports the record locking aspect of database transactions, but it does not treat operations in a transaction as a single atomic event. In other words, each operation takes place as soon as its statement is processed, and no operation can be rolled back. With *xf*ODBC, committing or rolling back a transaction merely releases records locked for the transaction. For example, if a transaction includes an insert, the insert will take place as soon as the INSERT statement is processed.

If you don't use a transaction, *xf*ODBC locks rows only for the time it takes to process the DELETE, INSERT, or UPDATE statement (autocommit). If you do use a transaction, *xf*ODBC locks all rows that are read after the ODBC application starts the transaction and holds the locks for the duration of the transaction. Locks are released only when the transaction is committed or rolled back or when the connection to the database is terminated.

Note that because *xf*ODBC locks all rows read after the start of a transaction, every row in the selected table will be locked until the transaction is committed or rolled back (or until the connection is terminated) unless the SQL statement that locked the rows includes a restriction that uses a unique index. For example, if you run the following query against the sample database that's distributed with Connectivity Series, it locks all rows in the orders table, even those whose or_price is not greater than 1.50, because or_price is not a key.

```
SELECT or_customer FROM orders WHERE or_price > 1.50
    FOR UPDATE OF
```

The next example, however, locks only rows that meet the restriction clause because the restriction clause uses a unique key (the or_vendor field).

```
SELECT or_customer FROM orders WHERE or_vendor = 41
    FOR UPDATE OF
```

Note the following:

- We don't recommend using transactions because of the high overhead they incur. However, if you use an application that uses transactions for operations that don't update data (e.g., reporting), make sure the transactions are read-only.
- If a Synergy database is read-only (i.e., if groups and tables are set to allow read-only access, which we recommend), a transaction that's read/write will cause an error. If this is the case, don't solve the problem by allowing read/write access to the database. Instead, set the application to use read-only transactions or, if that's not possible, ask the application vendor to update the application to not use transactions.
- *xf*ODBC supports only the following transaction isolation levels: SQL_TXN_READ_UNCOMMITTED and SQL_TXN_READ_COMMITTED.

Accessing Synergy Data with ADO

If you are writing an application in a language that can use Microsoft ActiveX Data Objects (ADO)—languages such as Visual Basic, ASP scripting languages, etc.—you can access a Synergy database via ADO if

- your Synergy database has been prepared for ODBC access—i.e., you have generated a system catalog, created a connect file, and so forth. Note that for client/server configurations, you'll need a DSN for the Synergy database, but for stand-alone configurations, the DSN is optional.
- your application has a valid connection string.

Creating connection strings that include a DSN

To access Synergy data using ADO, you need a connection string that includes all of the information needed to make the connection. The following is the syntax for a connection string that uses a DSN:

`DSN=`*dsn*`;`*[*`UID=`*[user]*`;`*][*`PWD=`*[pwd]][*`;DBQ=`*[connect]]*`;`*]*

where *dsn* is the DSN, *user* is the user name, *pwd* is the password for the user, and *connect* is the name of the connect file for the Synergy database. Note the following:

- The user name, password, and connect file specifications are optional.
- If you are connecting to a Synergy database on a remote server, the server information must be in the DSN.
- If you use the DBQ keyword in a connection string for Crystal Reports, you must prefix DBQ with <CRWDC> (for example, "<CRWDC>DBQ=sodbc_sa").

The following is an ADO example uses Visual Basic. In this example, a DSN (MyDSN) is used to make a connection to the sample Synergy database.

```
<%
DIM sConnStr
sConnStr = "DSN=MyDSN;UID=DBADMIN;PWD=MANAGER;"

DIM SQLQuery
SQLQuery = "SELECT * FROM Customers"

Set OBJdbConn = Server.CreateObject("ADODB.Connection")
OBJdbConn.Open sConnStr

Set RsCustomerList = OBJdbConn.Execute(SQLQuery)
%>
```

DSN-less connections

To connect to a local Synergy database without using a DSN, use the following syntax for the connection string:

`DRIVER=xfODBC;UID=`*[user][*`;PWD=`*[pwd]][*`;DBQ=`*[connect]*`;`*]*

or

`DRIVER=xfODBC;UID=`*[user]*`/`*[pwd][*`/`*connect]*`;`

where *user* is the user name, *pwd* is the password for the user, and *connect* is the name of the connect file for the database. Note the following:

- DSN-less connection strings are not supported for remote connections.
- If you don't supply a user name, password, or connect file name, and one is needed for the connection, the user will be prompted to supply the missing information.
- With DSN-less connections, you cannot change the settings listed in step 5 of "Adding a user or system DSN" on page 8-5 (settings such as the maximum number of columns that can be returned for a query). With a DSN-less connection, these settings cannot be changed from their defaults.

The following example is a line of Visual Basic code that connects to the sample Synergy database without using a DSN:

```
conn.Open "DRIVER=xfODBC;UID=DBADMIN/MANAGER/sodbc_sa;"
```

Accessing Synergy Data in a .NET Environment

There are two ways to access Synergy data from non-Synergy programs in a .NET environment. Both use ADO.NET:

- Use the .NET Framework Data Provider for ODBC.
- Use the Synergy/DE Data Provider for .NET.

If you access Synergy data with the .NET Framework Data Provider for ODBC, which is included with .NET Framework, you will have access to all of the features of that provider, but you won't be able to use the ADO.NET Entity Framework. The .NET Framework Data Provider for ODBC does not support it.

The Synergy/DE Data Provider for .NET includes all of the functionality of the .NET Framework Data Provider for ODBC, which it wraps, plus it enables you to use the Entity Framework to access Synergy databases (so you can use entity data models, LINQ to Entities, and so forth). It also includes a Visual Studio plug-in that provides integration with Visual Studio (e.g., enables you to create a data connection to a Synergy database). Once you have installed the Synergy/DE Data Provider for .NET and prepared a Synergy database for ODBC access, you can create a Visual Studio data connection to the database, you can generate and manipulate an entity data model (EDM) for the database, and you can use the EDM to access the Synergy data. To query the database via the EDM, you can use Entity SQL in any .NET language, and you can use LINQ to Entities queries in any .NET language that supports LINQ (Visual C#, VB.NET, and so forth).

Note the following:

- We recommend using the Synergy/DE Data Provider for .NET, even if you do not plan to use the Entity Framework. The Synergy/DE Data Provider for .NET not only wraps the .NET Framework Data Provider for ODBC, it includes additional optimization, a debugging tool, and other enhancements. (For information on the debugging tool, see "Synergex.Data.SynergyDBMSClient.SdeCommand" on page 9-30.)
- We recommend using ISAM files with the Synergy/DE Data Provider for .NET and the .NET Framework Data Provider for ODBC. ASCII data files are not supported, and inserts, updates, and for relative files joins are limited because the only key is the record number. With ISAM files, the *xf*ODBC driver is able to create additional keys for optimization.
- The Synergy/DE Data Provider for .NET does not support stored procedures, the Verify SQL Syntax feature of Visual Studio (Query Designer > Verify SQL Syntax), or some standard query operators and canonical functions. See "LINQ to Entities support" on page 9-22 and "Entity SQL support" on page 9-23 for information on standard query operators and canonical functions that are supported.
- For inserting, updating, and deleting records, use *xf*ServerPlus methods rather than the Synergy/DE Data Provider for .NET or the .NET Framework Data Provider for ODBC. ADO.NET is not efficient for these operations, and ODBC access can lead to record locking issues.

You must do the following to use the .NET Framework Data Provider for ODBC or the Synergy/DE Data Provider for .NET:

1. Prepare your Synergy database for ODBC access—i.e., generate a system catalog, create a connect file, and so forth. See "The Steps to ODBC Access" on page 1-13.
2. Supply connection information in your application. See "Configuring a data connection in Visual Studio" on page 9-12 and "Connection strings" on page 9-18.

   See the rest of this section for information on using the Synergy/DE Data Provider for .NET. For information on using the .NET Framework Data Provider for ODBC with Synergy databases, see Microsoft's ADO.NET and Visual Studio documentation, and see

   - "Third-Party Software Requirements" on page 9-3.
   - "Connection strings" on page 9-18. The information in this section about the data provider connection string also applies to the .NET Framework Data Provider for ODBC.
   - "Time columns and ADO.NET" on page 9-20. The information in this section applies to both the Synergy/DE Data Provider for .NET and the .NET Framework Data Provider for ODBC.
   - **xfODBC_DataReader_v3.zip** and **xfODBC_v2.zip**, available from Synergy CodeExchange in the Resource Center on the Synergex web site.

System requirements for the Synergy/DE Data Provider for .NET

To use the Synergy/DE Data Provider for .NET, you must have the following:

Required Software	Versions
Operating system	Windows 8 Windows 7 Windows Vista with Service Pack 2 or higher Windows XP with Service Pack 3 Windows Server 2012 Windows Server 2008 R2 Windows Server 2008 with Service Pack 2 or higher Windows Server 2003 with Service Pack 2
.NET Framework	Version 4 or 4.5
Visual Studio (required only for development)	Visual Studio 2010 SP1 or 2012 (Express is not supported)
Entity Framework	4.0 or higher for Visual Studio 2010 5.0 or higher for Visual Studio 2012 Versions 4.1 and higher are partially distributed with Visual Studio and .NET Framework. For the full version, which is required, see **http://nuget.org/packages/entityframework**.

Using the Synergy/DE Data Provider for .NET

The following is the general procedure for using the Synergy/DE Data Provider for .NET. The last step is for deployment machines. Prior steps are for a development machine.

1. Make sure your operating system, the .NET Framework, and Visual Studio meet the requirements listed above. Then install the Synergy/DE Data Provider for .NET. (If you install the Synergy/DE Data Provider for .NET on a system that does not have Visual Studio, only the deployment portion of the data provider will be installed. See the release notes, **REL_SDP.TXT**, for details.)
2. Generate a system catalog for the Synergy database, create a connect file, create a DSN, and set necessary data-access options. (See "The Steps to ODBC Access" on page 1-13 for an overview of this process.) Note the following:
    - To work with the Synergy/DE Data Provider for .NET, a DSN must specify a connect file.
    - If you plan to use the Entity Framework, make sure the "Null allowed" property for all fields used in key definitions is set to No. See "Preventing null updates and interpreting spaces, zeros, and null values" on page 3-27.
    - To use the Entity Framework, each data file must have a unique key. The Entity Framework requires a primary key, and with *xf*ODBC, the first unique key is considered the primary key.
3. In Visual Studio, add a data connection to the Synergy database, and then test it. See "Configuring a data connection in Visual Studio" below. (You can also add connection strings manually to the application. See "Connection strings" on page 9-18.)
4. Create an EDM or a DataSet and write a query for the EDM or DataSet, or use Query Designer or a DataReader to directly query the data source. See "Querying Synergy data from a .NET application" on page 9-14.
5. Before you deploy your application, make sure deployment machines meet requirements (see "System requirements for the Synergy/DE Data Provider for .NET" above), install the Synergy/DE Data Provider for .NET, and make sure they have database access, including the correct DSN.

Configuring a data connection in Visual Studio

Once you've installed the Synergy/DE Data Provider for .NET and prepared the Synergy data for ODBC access, you can set up a Visual Studio data connection to the Synergy database. A data connection enables you to access your Synergy database in various ways from Visual Studio (see "Using the Retrieve Data function" on page 9-37 and "Using Query Designer" on page 9-38), and it makes it easier to add connection information to your programs. When you use a data connection to generate an EDM, a connection string is added to the **App.Config** file. You can also manually code a connection string in your application. See "Connection strings" on page 9-18 for information.

Except for the following, the procedure for configuring a data connection for a Synergy database is the same as for any other data source, so see Microsoft's Visual Studio documentation for information. For an example, see "Adding a data connection for the sample database" on page 9-35. Note the following:

- "Synergy Database" must be selected as the data source (in the Add Connection, Modify Connection, Change Data Source, and Choose Data Source windows).
- "Synergy/DE Data Provider for .NET" must be selected as the data provider (in the Change Data Source and Choose Data Source windows).
- The DSN you use must specify a connect file.

Also note that the Synergy/DE Data Provider for .NET includes its own versions of the Add Connection and Modify Connection windows. See "The Add Connection and Modify Connection windows" below.

The Add Connection and Modify Connection windows

The Add Connection and Modify Connection windows for the Synergy/DE Data Provider for .NET have the following fields:

Data source. This read-only field displays the type of data connection that will be created. To use the Synergy/DE Data Provider for .NET, this should be set to "Synergy Database (SDEClient)." To change this, use the Change button.

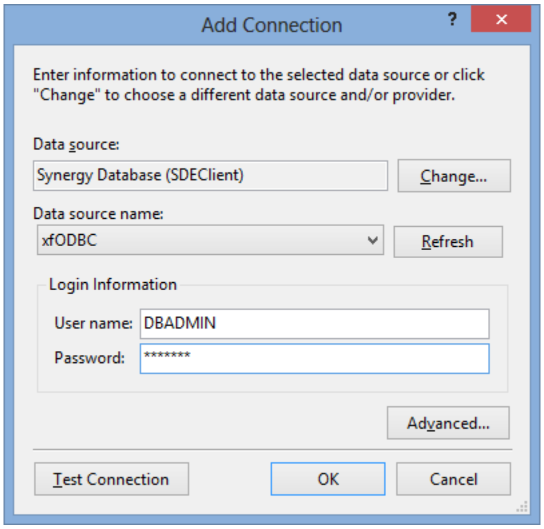
Data source name. Select the DSN for the Synergy database you will access. This is required, and note that the DSN you select here must include connect file information. The DSN does not need to specify user and password information, but if it does, the user and password in the DSN will be the default for the connection.

Click the Refresh button to update the drop-down list to reflect DSNs added or removed since the Add Connection or Modify Connection window was opened.

User name. Enter a user name (case sensitive) for the Synergy database if necessary. When the Add Connection window first opens, this field displays the user name specified in the DSN, if there is one. If you enter a user name in this field, this name will override the user name information in the DSN for the duration of the Visual Studio session.

Password. Enter the password (case sensitive) for the user name if necessary. If you enter a password here, or clear asterisks that represent a password in the DSN, that change will override the password information in the DSN for the duration of the Visual Studio session.

The Add Connection and Modify Connection windows also have an Advanced button, which gives you another way to change information in these windows. And they have a Test Connection button, which enables you to test the validity of data connection settings without leaving the Add Connection or Modify Connection window.



Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Synergy Database (SDEClient)

Data source name: xfODBC

Login Information

User name: DBADMIN

Password: *******

Figure 9-1. Configuring a data connection with the Add Connection window.

If you use the Test Connection button, note that a “Data Source name is missing” error could indicate that the DSN does not specify a connect file. (For a DSN to work with the Synergy/DE Data Provider for .NET, it must specify a connect file.) For other errors (e.g., “Authorization failure”), check the information on the Add Connection or Modify Connection window. If it appears to be correct, see “Troubleshooting Data Access” on page 9-4.

See “Adding a data connection for the sample database” on page 9-35 for an example that uses the Add Connection window.

Querying Synergy data from a .NET application

Once you’ve set up a Visual Studio data connection to a Synergy database, you can

- use the Visual Studio Query Designer to directly query the database.
- create a DataSet object (by using the Visual Studio DataSet Designer, for example). Then access the object model for the DataSet object and/or add LINQ to DataSet queries to your code.
- use System.Data.Odbc.OdbcDataReader, which is the DataReader for the .NET Framework Data Provider for ODBC. (This works with the Synergy/DE Data Provider for .NET because that provider wraps the .NET Framework Data Provider for ODBC.)
- use the Visual Studio Entity Designer to create an EDM, and then add Entity SQL queries and LINQ to Entities queries to your code to query the EDM and, in turn, the Synergy database.

See Microsoft's Visual Studio documentation for information on these topics, and see

- "Appendix B: SQL Support" for information on *xf*ODBC's support for SQL. Ultimately, all ADO.NET access to Synergy data takes the form of SQL statements querying the database through *xf*ODBC. (For example, the Synergy/DE Data Provider for .NET translates Entity SQL and LINQ to Entities queries into SQL for the *xf*ODBC driver.) So the SQL support and limitations for *xf*ODBC apply to all of the methods listed above.
- "Generating an EDM for a Synergy database" on page 9-15 and Microsoft's Visual Studio documentation for information on creating and using an EDM. (See "Using an EDM to query Synergy data" on page 9-43 for an example that queries the sample database using Entity SQL and LINQ to Entities.)
- "Operators, functions, classes, and exceptions" on page 9-21 for information on operators and functions supported by the Synergy/DE Data Provider for .NET.

TIP

The Visual Studio Entity Designer, DataSet Designer, and Query Designer display data objects and information differently. For example,

- the Entity Designer displays associations for navigational properties as connectors between tables. These navigational properties represent foreign keys. See figure 9-2 on page 9-17.
- the DataSet Designer displays foreign key relationships as connectors between tables.
- the Query Designer shows join operations (rather than foreign key relationships) as connectors between tables, and connectors may look different depending on the columns used in the join operation—e.g., a key icon is displayed if the join uses a key column. See figure 9-8 on page 9-39.

Generating an EDM for a Synergy database

The Synergy/DE Data Provider for .NET enables you to generate an entity data model (EDM) from a Synergy database, manipulate it, and use it to access Synergy data. An EDM is a conceptual model that represents a database schema as a set of entities and relationships, which are created from database objects selected in the EDM wizard. If you write a LINQ to Entities or Entity SQL query, you query these entities, and this results in a standard SQL query against the Synergy database. See Microsoft's ADO.NET Entity Framework documentation for more information on EDM objects, and see "Generating and manipulating an EDM" on page 9-40 for an example that generates an EDM for the sample database. Note the following:

- Entity types are created from tables and views. However, if all columns are nullable for a table or view that doesn't have a primary key, no entity type will be created for the table or view. (Errors, warnings, or messages may be displayed in the Visual Studio output window or Error List pane, but no exceptions will be thrown.)
- Entity keys are created from primary keys. However, if a table or view does not have a primary key, scalar properties for one or more non-nullable columns will be treated as keys. Note that this can result in unexpected mappings in an EDM if you create an entity that inherits from a entity with multiple keys.

- Associations between entity types are created from foreign key relations. Note, however, that relations that use at least one table column, but not all columns in a primary key will not be mapped as associations in an EDM.
- Scalar properties and navigation properties are created from database columns. Columns that have a role in an association appear as navigation properties. Other columns appear as scalar properties.

Also note that the SQL data types for system catalog columns are mapped to the following data types in the Entity Framework. For information on system catalog column types, see "Data types" on page 3-13.

Data Type Mappings for Entity Framework

SQLDescribeCol type	Entity Framework type
SQL_BIGINT	Int64
SQL_BINARY	Binary
SQL_BIT	Boolean
SQL_DECIMAL	Decimal
SQL_FLOAT	Decimal
SQL_INTEGER	Int64 for unsigned, Int32 for signed
SQL_SMALLINT	Int32 for unsigned, Int16 for signed
SQL_TINYINT	Byte for unsigned, Int16 for signed
SQL_TYPE_DATE	DateTime
SQL_TYPE_TIME	DateTime
SQL_TYPE_TIMESTAMP	DateTime
SQL_VARCHAR	String

When EDM generation is complete, a visual representation of the EDM is displayed in the Entity Model Designer (see figure 9-2), and the Synergy/DE Data Provider for .NET adds the following to your Visual Studio project:

- An XML file (***model_name*.edmx**) that defines the conceptual model, the data store, and mappings between the conceptual model and data store.
- A code file containing the classes for each entity type. The filename for this is the filename used for the **.edmx** file (without the **.edmx** extension) followed by "Designer" and the extension for the .NET language (for example, ***model_name*.Designer.cs** for C#).

- The following assemblies. (You can see these under References in the Solutions Explorer.)
  - System.Core
  - System.Data
  - System.Data.Entity
  - System.Runtime.Serialization
  - System.Security
  - System.Xml

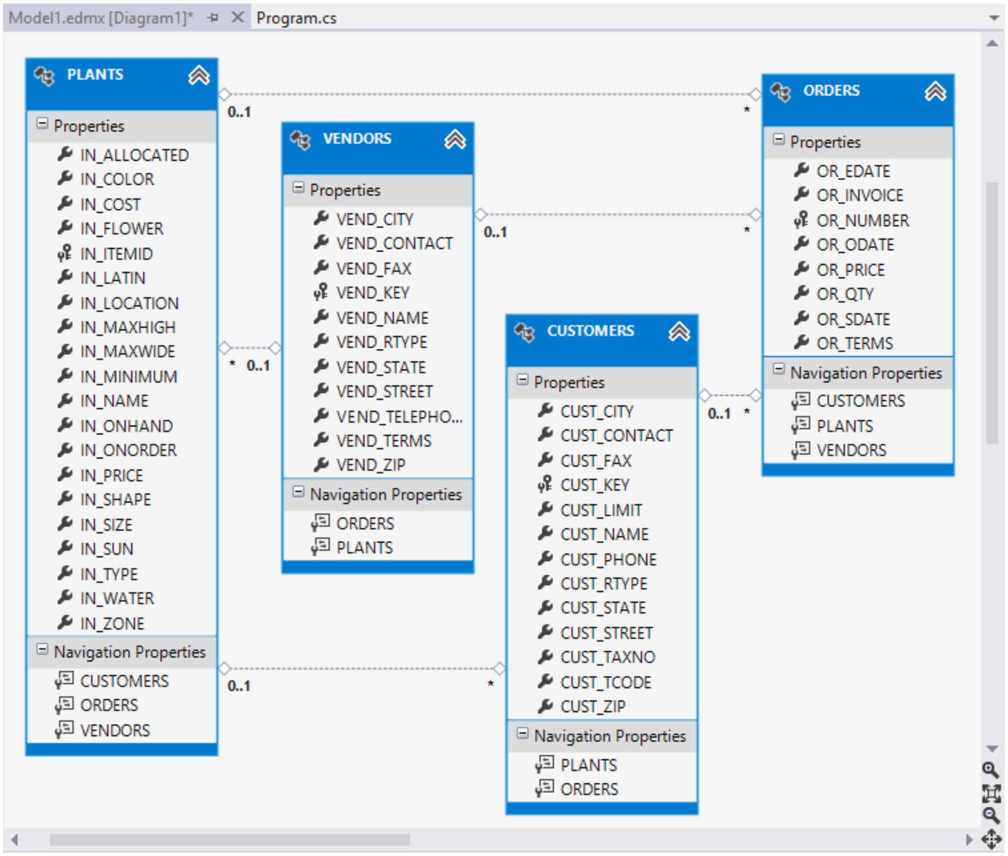


Figure 9-2. Visual representation of an EDM generated for the sample database.

Connection strings

Generally, you'll establish a connection to a Synergy database by setting up a data connection (see "Configuring a data connection in Visual Studio" on page 9-12) and then selecting the data connection when you generate an EDM. In the process, Visual Studio generates a connection string and saves it to the **App.Config** file. However, you can manually add connection strings to your application configuration file (**App.Config**), and you can add connection strings to your code by

- instantiating EntityConnection if you're using the Entity Framework. (See Microsoft's ADO.NET documentation for information on EntityConnection.)
- instantiating SdeConnection if you're using the Synergy/DE Data Provider for .NET but *not* the Entity Framework. (See "Synergex.Data.SynergyDBMSClient.SdeConnection" on page 9-31.)

If you're using the Entity Framework, the connection string is actually made up of two connection strings for two components of the Synergy/DE Data Provider for .NET: an outer connection string for the Entity Framework provider and an inner connection string for the data provider. (See the examples below.) If you're not using the Entity Framework, the connection string consists only of the connection string for the data provider. Note that the Entity Framework connection string must conform to the connection string syntax for the Entity Framework, and the data provider connection string must conform to the connection string syntax for the .NET Framework Data Provider for ODBC (because the data provider component of Synergy/DE Data Provider for .NET wraps the .NET Framework Data Provider for ODBC). See Microsoft's ADO.NET documentation for information on the syntax of these connection strings, and note the following:

- The provider= setting for the data provider connection string must be set to Synergex.Data.SynergyDBMSClient (i.e., provider=Synergex.Data.SynergyDBMSClient).
- If you're using the Entity Framework, the providerName= setting of the Entity Framework provider connection string must be set to System.Data.EntityClient (i.e., providerName="System.Data.EntityClient").

Also note that you must specify either a connect file or a DSN that specifies a connect file for your Synergy database. However, if you create a connection string that doesn't specify a DSN (by using DBQ= to specify a connect file and Driver= to specify "xfODBC"), you cannot change the settings listed in step 5 of "Adding a user or system DSN" on page 8-5 (settings such as the maximum number of columns that can be returned for a query). With a DSN-less connection, these settings cannot be changed from their defaults.

Example App.Config connection strings

The following example was generated by Visual Studio for a data connection and was saved to the **App.Config** file for a project. The outer connection string starts with “connectionString=” and the inner connection string starts with “provider connection string=”.

```
<configuration>
<connectionStrings>
<add name="Entities" connectionString="metadata=res://*/Model.csdl| res:
//*/Model.ssdl|res://*/Model.msl;provider=Synergex.Data.SynergyDBMSClient
;provider connection string=&quot;Dsn=my_DSN;uid=DBA&quot;"providerName=
"System.Data.EntityClient" />
</connectionStrings>
</configuration>
```

The next connection string example specifies the Synergy/DE Data Provider for .NET, but it doesn’t include the Entity Framework connection string, so an application couldn’t use the Entity Framework with this connection.

```
<configuration>
  <connectionStrings>
   <add name="DPConnection" connectionString="Dsn=MsConnect;
   uid=DBA;pwd=MANAGER" providerName="Synergex.Data.SynergyDBMSClient" />
  </connectionStrings>
</configuration>
```

Example connection strings in code

The following C# example uses EntityConnection and includes the Entity Framework connection string, so the program can use the Entity Framework:

```
string dataProviderConnectionString = "DSN=xfODBC;UID=DBA;PWD=MANAGER";

EntityConnectionStringBuilder entityBuilder = new
                                   EntityConnectionStringBuilder();

entityBuilder.Provider = "Synergex.Data.SynergyDBMSClient";
entityBuilder.ProviderConnectionString = dataProviderConnectionString;
entityBuilder.Metadata = @"res://*/Model.csdl|
                         res://*/Model.ssdl|res://*/Model.msl";

using (EntityConnection connection = new
EntityConnection(entityBuilder.ToString()))
{
  connection.Open();
  Console.WriteLine("Just testing the connection.");
  connection.Close();
}
```

The next C# example uses SdeConnection, so the program can use the data provider component of the Synergy/DE Data Provider for .NET, but can't use the Entity Framework.

```
using (SdeConnection connect = new
         SdeConnection("DSN=xfODBC;UID=DBA;PWD=Manager"))
{
  connect.Open();
  SdeCommand command = new
    SdeCommand("Select * from Customers", connect);
  OdbcDataReader reader = (OdbcDataReader) command.ExecuteReader();
  while (reader.Read())
  {
    ...
```

Time columns and ADO.NET

ADO.NET retrieves SQL_TIME columns (SQL_TIME is the default for time columns) as System.TimeSpan, which is a .NET data type that represents a time interval rather than a specific time. Generally, this means that applications accessing the data need to be written to calculate the time from TimeSpan values. However, *xf*ODBC includes the VORTEX_ODBC_TIME environment variable, which can make these calculations unnecessary by instructing the *xf*ODBC driver to describe SQL_TIME columns as SQL_TIMESTAMP. Note that the Synergy/DE Data Provider for .NET sets this automatically; see "Formats for returned dates and times" on page 8-13 for more information.

Troubleshooting

If you are unable to create a connection to a Synergy database, make sure your system meets the requirements listed in "System requirements for the Synergy/DE Data Provider for .NET" on page 9-11, and make sure you have correctly set up your Synergy database for ODBC access. Start by attempting to connect locally to the database using an application such as ODBC Test. See the following for more information:

- "Troubleshooting Data Access" on page 9-4
- Chapter 11, "Data Access Errors and Error Logging"

For information on troubleshooting EDMs, see Microsoft's Visual Studio documentation. Visual Studio logs some errors in the Visual Studio Output pane and Error List pane and others as comments in the EDMX file.

As for queries, one of the benefits of using LINQ to Entities is that Visual Studio is able to check the validity of a query, so you can use IntelliSense and other Visual Studio tools to troubleshoot queries. Note, however, that valid queries may result in SQL that is inefficient and that may not be optimizable by the Synergy database driver. (See note in "LINQ to Entities support" on page 9-22.)

- To see the SQL generated for a LINQ to Entities or Entity SQL query, use Vortex API logging, Vortex host logging, or ODBC trace logging (see "Error Logging" on page 11-2), or use the DisplaySqlInDebug property of the SdeConnection class to print the SQL to the Visual Studio Output window when debugging. See "Synergex.Data.SynergyDBMSClient.SdeConnection" on page 9-31.
- For information on what *xf*ODBC can optimize, see chapter 10, "Optimizing Data Access."

Also note that the Synergy/DE Data Provider for .NET is subject to .NET Framework exceptions and the Synergy exceptions documented in "Synergy/DE Data Provider for .NET exceptions" on page 9-32. For information on .NET Framework exceptions, see Microsoft's .NET Framework class library documentation.

Operators, functions, classes, and exceptions

The Synergy/DE Data Provider for .NET supports the standard query operators, canonical functions, classes, and custom exceptions listed in this section.

- The "LINQ to Entities support" section below documents support for LINQ to Entities by listing the standard query operators you can use with the Synergy/DE Data Provider for .NET. These operators are methods in the Standard Query Operators API, and they enable you to create queries that include filtering, projection, joins, grouping, aggregation, set operations, and paging/element operations.
- "Entity SQL support" on page 9-23 lists supported canonical functions. Canonical functions work with the entity data model (EDM), have direct SQL translations, and are common to all providers for the .NET Framework.
- "Synergy/DE Data Provider for .NET classes" on page 9-30 documents Synergex.Data.SynergyDBMSClient classes, which are specific to the Synergy/DE Data Provider for .NET. (The Synergy/DE Data Provider for .NET also uses the System.Data.ODBC and System.Data.Common classes. See Microsoft's .NET Framework documentation for information.)
- "Synergy/DE Data Provider for .NET exceptions" on page 9-32 documents exceptions that are specific to the Synergy/DE Data Provider for .NET and are raised for queries that result in unsupported SQL. (For information on .NET Framework exceptions, see Microsoft's .NET Framework documentation.)

LINQ to Entities support

The following list of standard query operators documents the Synergy/DE Data Provider for .NET's support for LINQ to Entities. These operators are methods in the Standard Query Operators API, and they enable you to create queries. Standard query operators are supported for both the query expression syntax and the method-based query syntax. They operate on sequences (i.e., objects that implement an IEnumerable<T> or IQueryable<T> interface), and results are returned in sequences.

Note that the Synergy/DE Data Provider for .NET translates LINQ to Entities queries into SQL queries for the ODBC layer. For example, Where (a standard query operator) results in a WHERE clause in the generated SQL, so both of the following queries result in the clause "WHERE cust_key = 5":

```
var query = Db.Customers.Where(c => c.CUST_KEY == 5);
```

```
var query = from c in Db.Customers where c.CUST_KEY == 5 select c;
```

The first query above is a C# method-based query that limits the elements returned in the Db.Customers sequence to those whose CUST_KEY value is 5. The second is the equivalent in query expression syntax.

Not all operators translate so neatly into SQL. Some have no direct SQL equivalent. And the SQL generated from LINQ to Entities queries (especially for complex queries) may be inefficient and may not be optimizable by the Synergy database driver.

The Synergy/DE Data Provider for .NET supports the following standard query operators. For information on standard query operators, see Microsoft's LINQ to Entities documentation.

Standard Query Operators

Type	Operators
Sorting Data	OrderBy OrderByDescending Then ThenByDescending
Set Operations	Distinct Union
Filtering Data	Where
Quantifier Operations	All Any Contains
Projection Operations	Select SelectMany

Standard Query Operators (Continued)

Type	Operators
Partitioning Data	Skip Take
Join Operations	Join GroupJoin
Grouping Data	GroupBy
Element Operations	First FirstOrDefault
Concatenation Operations	Concat
Aggregation Operations	Average Count LongCount Max Min Sum

Entity SQL support

The Synergy/DE Data Provider for .NET supports the following Entity SQL canonical functions. All other aspects of Entity SQL are supported as documented in Microsoft's Entity SQL documentation. In order to query a Synergy database, the Synergy/DE Data Provider for .NET translates all Entity SQL into standard SQL for *xf*ODBC. Canonical functions, for example, have direct SQL translations as listed in the table below.

See "Troubleshooting" on page 9-20 for information on viewing the SQL that's generated for a query, see "Appendix B: SQL Support" for information on the SQL supported by *xf*ODBC, and see chapter 10, "Optimizing Data Access," for information on what *xf*ODBC can optimize.

Canonical Functions				
Function	**Description**	**Valid parameter types**	**Return type**	**SQL translation**
Abs(*value*)	A math function that returns the absolute value of *value*.	Int16, Int32, Int64, Byte, Single, Double, Decimal	Type of *value*	ABS(*num_exp*)
Avg(*expression*)	An aggregate function that returns the average of non-null values.	Int32, Int64, UInt32, UInt64, Double, Decimal (Null if input values are all null)	Type of *expression* or null if all input values are null	AVG(*col*)
BigCount(*expression*)	An aggregate function that returns the size of the aggregate, including null and duplicate values.	Any type	Int64	COUNT(*col*)
BitWiseAnd(*value1, value2*)	A bitwise function that returns the results of a bitwise AND operation performed on *value1* and *value2*.	Byte, Int16, Int32, Int64	Type of *value1, value2*	BITAND(*num_exp, num_exp2*)
BitWiseOr(*value1, value2*)	A bitwise function that returns the result of a bitwise OR operation performed on *value1* and *value2*.	Byte, Int16, Int32, Int64	Type of *value1, value2*	BITOR(*num_exp, num_exp2*)
BitWiseXor(*value1, value2*)	A bitwise function that returns the result of a bitwise exclusive OR operation performed on *value1* and *value2*.	Byte, Int16, Int32, Int64	Type of *value1, value2*	BITXOR(*num_exp, num_exp2*)
Ceiling(*value1, value2*)	A math function that returns the largest of two values.	Single, Double, DateTime, String, Char, Decimal	Type of *value1*, *value2*	GREATEST (*expression1, expression2*)

Canonical Functions (Continued)

Function	Description	Valid parameter types	Return type	SQL translation
Concat(*string1, string2*)	A string function that returns a string that's a combination of *string1* and *string2*. Note that an error will occur if the length of the return value string is greater than the maximum length allowed.	String	String	CONCAT(*str_exp1, str_exp2*)
Count(*expression*)	An aggregate function that returns the size of the aggregate, including null and duplicate values.	Any type	Int32	COUNT(*col*)
CurrentDateTime()	A date/time function that returns the current date.	n/a	System.datetime	CURDATE()
Day(*datetime*)	A date/time function that returns the day portion of *datetime* as a value from 1 to 31.	Datetime constant or a datetime column	Int32	CAST(TO_CHAR (*value*, 'DD') AS SQL_INTEGER)
Floor(*value1, value2*)	A math function that returns the lesser of two values.	Single, Double, DateTime, String, Char, Decimal	Type of *value1, value2*	LEAST (*expression1, expression2*)
Hour(*datetime*)	A date/time function that returns the hour portion of *datetime* as a value from 00 to 23.	Datetime constant or a datetime column	Int32	CAST(TO_CHAR (*value*, 'HH') AS SQL_INTEGER)
IndexOf(*string1, string2*)	A string function that returns the position of *string1* in *string2*, or returns 0 if not found. A return value of 1 indicates that it starts at the beginning of *string2*.	String	Int32	INSTR(*str_exp1, str_exp2*)

Canonical Functions (Continued)

Function	Description	Valid parameter types	Return type	SQL translation
Left(*string, n*)	A string function that returns the first *n* characters of *string*. Note that *n* cannot be less than zero and that you must use the ODBC escape sequence with Left: {fn Left(*string, n*)}	String and one of the following for *n*: SByte, Int16, Int32, Int64, Byte, UInt16, UInt32, UInt64	String	LEFT(*str_exp, n*)
Length(*string*)	A string function that returns the length in characters of *string*.	String	Int32	LENGTH(*str_exp*)
LTrim(*string*)	A string function that returns *string* with any leading blanks removed.	String	String	LTRIM(*str_exp*)
Max(*expression*)	An aggregate function that returns the maximum non-null value.	Byte, Int16, Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Decimal, DateTime, DateTimeOffset, Time, String, Binary (Null if input values are all null)	Type of *expression* or null if all input values are null	MAX(*col*)
Millisecond(*datetime*)	A date/time function that returns the millisecond portion of *datetime* as a value from 0 to 9999.	Datetime constant or a datetime column	Int32	CAST(TO_CHAR (Value, 'UUUUUU') AS SQL_INTEGER

Canonical Functions (Continued)

Function	Description	Valid parameter types	Return type	SQL translation
Min(*expression*)	An aggregate function that returns the minimum non-null value.	Byte, Int16, Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Decimal, DateTime, DateTimeOffset, Time, String, Binary (Null if input values are all null)	Type of *expression* or null if all input values are null	MIN(*col*)
Minute(*datetime*)	A date/time function that returns the minute portion of *datetime* as a value from 00 to 59.	Datetime constant or a datetime column	Int32	CAST(TO_CHAR (*value*, 'MI') AS SQL_INTEGER)
Month(*datetime*)	A date/time function that returns the month portion of *datetime* as a value from 1 to 12.	Datetime constant or a datetime column	Int32	CAST(TO_CHAR (*value*, 'MM') AS SQL_INTEGER)
Replace(*string1, string2, string3*)	A string function that searches for a string (*string2*) in a string (*string1*) and replaces occurrences of the found string with another string (*string3*).	String	String	REPLACE(*str_exp, str_exp2, str_exp3*)
Reverse(*string*)	A string function that reverses the order of the characters in the returned string.	String	String	REVERSE(*str_exp*)

Canonical Functions (Continued)

Function	Description	Valid parameter types	Return type	SQL translation
Right(*string*, *n*)	A string function that returns the last *n* characters from *string*. Note that *n* cannot be less than zero and that you must use the ODBC escape sequence with Right: {fn Right(*string, n*)}	String and one of the following for *n*: SByte, Int16, Int32, Int64, Byte, UInt16, UInt32, UInt64	String	RIGHT(*str_exp, n*)
Round(*value*)	A math function that returns the integer portion of a *value* rounded to the nearest integer.	Single, Double, Decimal	The type of *value*	ROUND (*expression*)
RTrim(*string*)	A string function that returns *string* with any trailing blanks removed.	String	String	RTRIM(*string*)
Second(*datetime*)	A date/time function that returns the second portion of *datetime* as a value from 00 to 59.	Datetime constant or a datetime column	Int32	CAST(TO_CHAR (*value*, 'SS') AS SQL_INTEGER)
Substring(*string, start, length*)	A string function that returns a substring of *string* that begins at the position *start* and is *length* characters long. A *start* of 1 indicates that the substring starts at the beginning of *string*. Note that *start* cannot be less than 1. *Length* cannot be less than 0.	String and one of the following for *start* and *length*: SByte, Int16, Int32, Int64, Byte, UInt16, UInt32, or UInt64	String	SUBSTR(*str_exp, start, length*)

Canonical Functions (Continued)				
Function	**Description**	**Valid parameter types**	**Return type**	**SQL translation**
Sum(*expression*)	An aggregate function that returns the sum of non-null values.	Int32, Int64, UInt32, UInt64, Double, Decimal (Null if input values are all null)	Type of *expression* or null if all input values are null	SUM(*col*)
ToLower(*string*)	A string function that converts any uppercase characters in *string* to lowercase characters, and returns the resulting string.	String	String	LCASE(*str_exp*)
ToUpper(*string*)	A string function that converts any lowercase characters in *string* to uppercase characters, and returns the resulting string.	String	String	U_CASE(*str_exp*)
Trim(*string*)	A string function that returns *string* with leading and trailing blanks removed.	String	String	LTRIM(RTRIM (*str_exp*))
Year(*datetime*)	A date/time function that returns the year portion of *datetime*.	Datetime constant or a datetime column	Int32	CAST(TO_CHAR (*value*, ’YYYY’) AS SQL_INTEGER)

Synergy/DE Data Provider for .NET classes

The Synergy/DE Data Provider for .NET includes the System.Data.Odbc classes and the following Synergex.Data.SynergyDBMSClient classes (which are wrapped versions of System.Data.Odbc classes). These classes generally apply only if you are using ADO.NET without the Entity Framework (though you may want to use the DisplayInDebug property of SdeConnection for debugging in any case).

Because the Synergex.Data.SynergyDBMSClient classes wrap System.Data.Odbc classes, the following summarize these classes, but don't fully document them. For more information, see Microsoft's ADO.NET documentation for System.Data.Odbc classes.

Note the following:

- To use the Synergex.Data.SynergyDBMSClient classes, you must add a reference to **Synergex.Data.SynergyDBMSClient.dll** to your Visual Studio project.
- We recommend that you use the Synergex.Data.SynergyDBMSClient classes documented below rather than their System.Data.Odbc counterparts. This will make your code more compatible with future versions of the Synergy/DE Data Provider for .NET and will prevent unnecessary type conversions that happen if you use both Synergex.Data.SynergyDBMSClient classes and their System.Data.Odbc counterparts.

Synergex.Data.SynergyDBMSClient.SdeClientFactory

This wraps the OdbcFactory class in the System.Data.Odbc namespace. It has methods for creating objects for data source classes.

```
SdeCommand command = SdeClientFactory.Instance.CreateCommmand()
    as SdeCommand;
```

Synergex.Data.SynergyDBMSClient.SdeCommand

This wraps the OdbcCommand class in the System.Data.Odbc namespace, and it cannot be inherited. SdeCommand represents a SQL statement or command, and must be used with a data set or the data reader for the .NET Framework Data Provider for ODBC. For example, the following C# code uses SdeCommand to set up an SQL statement and to execute it with the data reader:

```
{
  ...
  SdeCommand command = new SdeCommand("Select * from Customers", connect);
  DbDataReader reader = command.ExecuteReader();
  while (reader.Read())
  {
  ...
  }
}
```

Synergex.Data.SynergyDBMSClient.SdeConnection

This wraps the OdbcConnection class in the System.Data.Odbc namespace. It represents a connection to a Synergy database, and it cannot be inherited. Note that SdeConnection has an additional property that OdbcConnection doesn't have. SdeConnection includes the static property DisplaySqlInDebug which can be set to true or false (false is the default). When set to true, this instructs the Synergy/DE Data Provider for .NET to print the SQL generated from an Entity SQL or LINQ to Entities query to the Visual Studio Output window when debugging.

The following C# example uses the SdeConnection constructor and Open method to connect using a DSN named xfODBC:

```
string cnString = ConfigurationManager.ConnectionStrings
  ["Connect1"].ConnectionString;

using (SdeConnection connect = new SdeConnection(cnString))
{
  connect.Open();
  SdeCommand command = new SdeCommand("Select * from Customers", connect);
  DbDataReader reader = command.ExecuteReader();
  while (reader.Read())
  {
  ...
  }
}
```

The following C# example uses the ConnectStrings method of the ConfigurationManager class (System.Configuration) to retrieve a connection string labeled “Connect1” from **App.Config**. It then uses the SdeConnection constructor to connect using the retrieved connection string:

```
string cnString = ConfigurationManager.ConnectionStrings
  ["Connect1"].ConnectionString;
SdeConnection connection = new SdeConnection(cnString);

using (SdeConnection connect =
       new SdeConnection("DSN=xfODBC;UID=DBA;PWD=Manager"))
{
  connect.Open();
  SdeCommand command = new SdeCommand("Select * from Customers", connect);
  DbDataReader reader = command.ExecuteReader();
  while (reader.Read())
  {
  ...
  }
}
```

See “Connection strings” on page 9-18 for more information.

Synergy/DE Data Provider for .NET exceptions

The Synergy/DE Data Provider for .NET is subject to .NET Framework exceptions and the following Synergy exceptions. For information on .NET Framework exceptions, see Microsoft's .NET Framework class library documentation.

InvalidCastException

Raised if a query results in invalid casting. See "Appendix B: SQL Support" for information on casting supported by *xf*ODBC.

InvalidExpressionTreeException

Raised if an expression tree passed to the Synergy/DE Data Provider for .NET is not a valid query tree. For example, this may indicate that the query contains an aggregate function or operator without a GROUP BY clause or a GroupBy operator.

InvalidSqlException

Raised if the query is well-formed, but the SQL generated by the Synergy/DE Data Provider for .NET is invalid or not supported by the *xf*ODBC driver. Contact Synergy/DE Developer Support.

UnsupportedFunctionException

Raised if the query uses a canonical function that is not supported by the Synergy/DE Data Provider for .NET. Rewrite the query to use only canonical functions listed in "Entity SQL support" on page 9-23.

UnsupportedOperatorException

Raised if the query (LINQ to Entities or Entity SQL) uses an operator that is not supported by the Synergy/DE Data Provider for .NET. Rewrite the query to use only operators listed in "LINQ to Entities support" on page 9-22.

Examples

The following examples connect to the sample Synergy database provided with the *xf*ODBC installation. These examples were created using Visual Studio 2010 and 2012 and ODBC Test 2.7. Procedures may differ for other versions.

Note that before you follow the steps in these examples, you must generate a system catalog for the sample database, create a connect file and DSN for the sample database, and create a user that can access the data. If you initialized users and groups, you can use the default user DBADMIN.

For more examples, see the following Synergex KnowledgeBase articles:

- 100001969—an example that accesses Synergy data with Microsoft Access
- 100001970—an example that uses Microsoft Word to create a form letter from Synergy data
- 100001972—an example that uses Crystal Reports to create a report with Synergy data

Using ODBC Test to test a query

ODBC Test (**Odbcte32.exe**) is an ODBC-enabled application distributed by Microsoft. It's useful for testing your *xf*ODBC setup and for testing SQL statements. If a query works with ODBC Test, the query and the setup (DSN, connect file, system catalog, data files, environment variable settings, etc.) are working. If you find that the same setup and SQL statement fail with another ODBC-enabled application, chances are the problem lies with the application (for example, with the way it generates SQL).

As with the other examples in this section, we'll access the sample Synergy database.

1. Open ODBC Test.
2. Select Full Connect from the Conn menu.
3. In the Data Source field of the Full Connect dialog box, select the DSN you created for the sample Synergy database, enter the user name and password for the DSN (optional), set the ODBC Behavior field to ODBC 2.0, and then click OK. (Leave the Cursor Library field set to Default.)

   Note that the user you log in as must have read privileges (but not necessarily write privileges) to complete this example.
4. If the xfODBC Info window opens, enter the user name, password, or connect file—whatever information is missing—and click OK. (This window opens if any one of these is not stored in the user DSN you selected and was not entered in the Full Connect dialog box.)

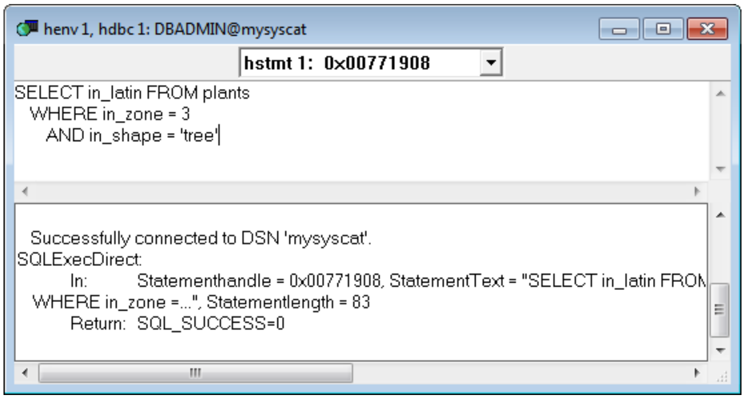
   If *xf*ODBC isn't able to connect to the sample database, you'll get an error message ("Connection failed...") that should give you some idea about the problem. If you are unable to solve the problem from the information provided by the error message, see "Troubleshooting Data Access" on page 9-4.

5. If ODBC Test is able to connect to the sample database, a window with two panes will open. (The user name and the DSN name—for example "DBADMIN@sample"—are part of the window title.) The lower pane displays information on the connection, including any errors or warnings.

   Enter the following SQL statement in the upper pane (see figure 9-3):

```
SELECT in_latin FROM plants
    WHERE in_zone = 3
        AND in_shape = 'tree'
```

6. Select SQLExecDirect from the Stmt menu, and then click OK in the SQLExecDirect window. The results, which should include SQL_SUCCESS=0, are displayed in the lower pane of the window for the connection. (If you get an error, the SQL statement entered in step 5 is probably incorrect.)



```
SELECT in_latin FROM plants
  WHERE in_zone = 3
    AND in_shape = 'tree'
```

```
  Successfully connected to DSN 'mysyscat'.
SQLExecDirect:
        In:         Statementhandle = 0x00771908, StatementText = "SELECT in_latin FRON
  WHERE in_zone =...", Statementlength = 83
        Return:  SQL_SUCCESS=0
```

Figure 9-3. Query and the results of SQLExecDirect.

7. If step 6 resulted in SQL_SUCCESS=0, select Get Data All from the Results menu. The results are appended to the information in the lower pane.

8. To disconnect, Select Full Disconnect from the Conn menu.

   There are many other ODBC API calls you can invoke from ODBC Test. See the online help for ODBC Test for information.

Adding a data connection and retrieving data in Visual Studio

The following examples, access Synergy data from Visual Studio by using the Entity Framework, so you will need the Synergy/DE Data Provider for .NET installed on a system that meets the requirements listed in "System requirements for the Synergy/DE Data Provider for .NET" on page 9-11.

These examples access the sample Synergy database, so before you start, you must follow the steps in chapter 2, "Using the Sample Database As a Tutorial," to set up the sample database for ODBC access. Note that the DSN you create for the sample database should specify the user name, password, and connect file.

Adding a data connection for the sample database

We'll start by adding a data connection, which you can use to

- retrieve data in Visual Studio. We'll do this in "Using the Retrieve Data function" on page 9-37 and "Using Query Designer" on page 9-38.
- set up connection information in an application developed in Visual Studio. We'll do this in "Generating and manipulating an EDM" on page 9-40.

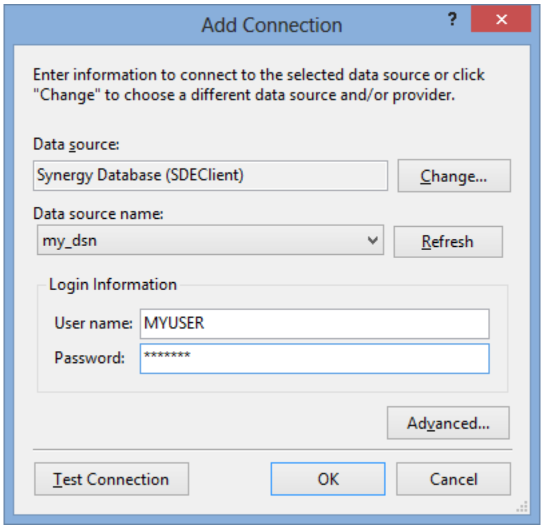
1. In Visual Studio, open the Server Explorer (View > Server Explorer) and select Tools > Connect to Database (or right-click the Data Connections node and select Add Connection from the context menu). Data connections are independent of solutions and projects, so you don't need to open a solution or project.

   The Add Connection window or Choose Data Source window opens.

2. Do one of the following:

   - If the Add Connection window opens, make sure the data source is set to "Synergy Database (SDEClient)." If it isn't, click the Change button and then, in the Change Data Source window, set the data source to "Synergy Database," which sets the data provider to "Synergy Data Provider for .NET," and click OK.
   - If the Choose Data Source window opens, set the data source to "Synergy Database," which sets the data provider to "Synergy/DE Data Provider for .NET," and click Continue.

3. In the Add Connection window, select the DSN you created for the sample database. As recommended above, the DSN should specify a user name and password, so there's no need to enter that information here. (If you do, it will override the information in the DSN.)



Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Synergy Database (SDEClient)

Data source name: my_dsn

Login Information

User name: MYUSER

Password: *******

Figure 9-4. Entering data connection information in the Add Connection window.

4. Click the Test Connection button at the bottom of the Add Connection window to make sure the connection information is correct. Note that a "Data Source name is missing" error could indicate that the DSN does not specify a connect file. (For a DSN to work with the Synergy/DE Data Provider for .NET, it must specify a connect file.) For other errors (e.g., "Authorization failure"), check the information on the Add Connection window. If it appears to be correct, see "Troubleshooting Data Access" on page 9-4.

5. Once you're able to connect with the Test Connection button, click OK to close the "Test connection succeeded" message, and then click OK in the Add Connection window. The Add Connection window closes, and a node for the data connection is added to the Data Connections branch of Server Explorer.

Using the Retrieve Data function

Once you’ve established a data connection, you can access the Synergy data from Visual Studio in various ways, including the Retrieve Data function, which displays the data for a table or view.

1. In Server Explorer, expand the data connection node added in step 5 of “Adding a data connection for the sample database” (above), expand the Tables node under that, and then right-click one of the tables (CUSTOMERS, ORDERS, PLANTS, or VENDORS).
2. From the context menu, select Retrieve Data. (See figure 9-5.)

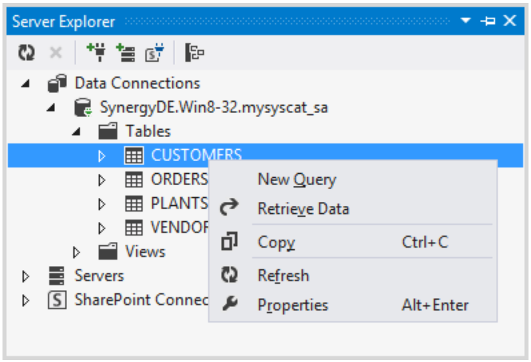
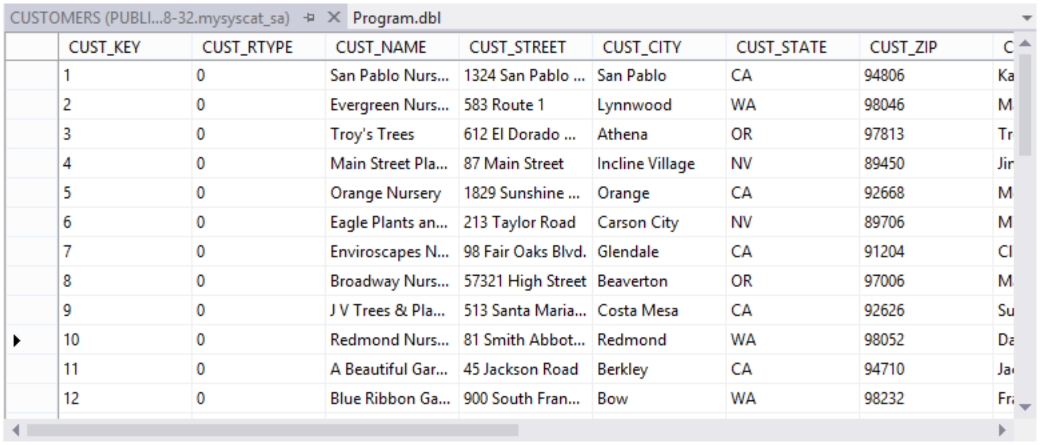


Figure 9-5. Selecting Retrieve Data from the context menu for the CUSTOMERS table.

A tab with the name of the table is added to the editor pane of Visual Studio. The tab displays all of the columns and rows for the table. (See figure 9-6.)


CUSTOMERS (PUBLI...8-32.mysyscat_sa) Program.dbl

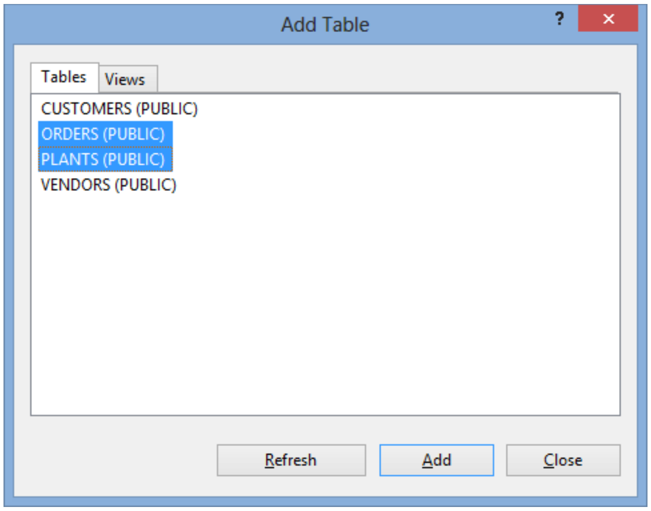
	CUST_KEY	CUST_RTYPE	CUST_NAME	CUST_STREET	CUST_CITY	CUST_STATE	CUST_ZIP	C
	1	0	San Pablo Nurs...	1324 San Pablo ...	San Pablo	CA	94806	Ka
	2	0	Evergreen Nurs...	583 Route 1	Lynnwood	WA	98046	M
	3	0	Troy's Trees	612 El Dorado ...	Athena	OR	97813	Tr
	4	0	Main Street Pla...	87 Main Street	Incline Village	NV	89450	Jir
	5	0	Orange Nursery	1829 Sunshine ...	Orange	CA	92668	M
	6	0	Eagle Plants an...	213 Taylor Road	Carson City	NV	89706	M
	7	0	Enviroscapes N...	98 Fair Oaks Blvd.	Glendale	CA	91204	Cl
	8	0	Broadway Nurs...	57321 High Street	Beaverton	OR	97006	M
	9	0	J V Trees & Pla...	513 Santa Maria...	Costa Mesa	CA	92626	Su
▶	10	0	Redmond Nurs...	81 Smith Abbot...	Redmond	WA	98052	Da
	11	0	A Beautiful Gar...	45 Jackson Road	Berkley	CA	94710	Ja
	12	0	Blue Ribbon Ga...	900 South Fran...	Bow	WA	98232	Fr

Figure 9-6. Results of Retrieve Data function for CUSTOMERS table.

Using Query Designer

Another Visual Studio tool you can use with a data connection is Query Designer, which enables you to enter SQL or use graphical tools to query a relational database.

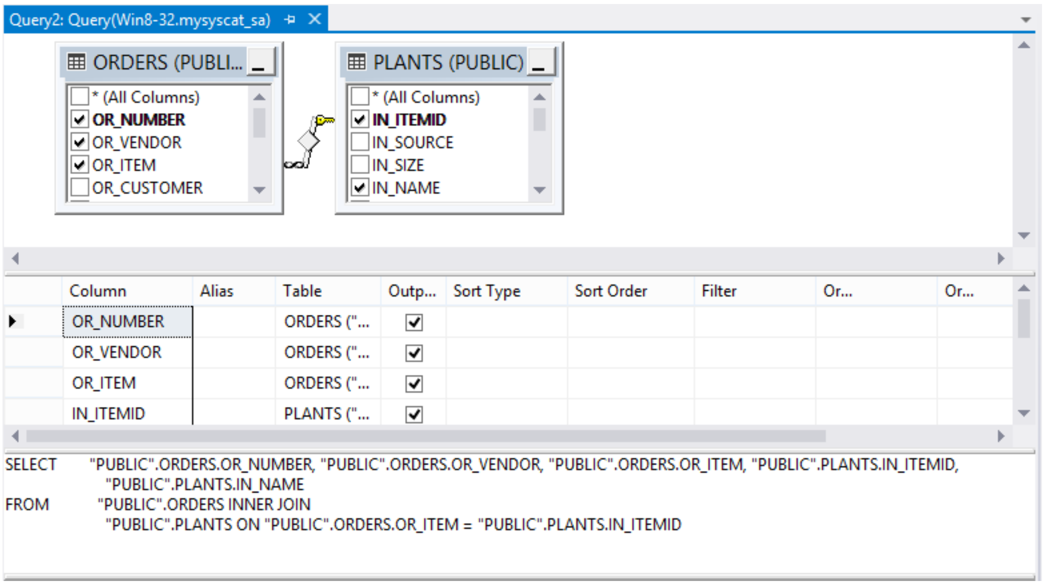
1. In Server Explorer, right-click the data connection node added in step 5 of "Adding a data connection for the sample database" on page 9-35 and select New Query from the context menu. If there is no New Query entry on the context menu, select Refresh from the context menu, and then right-click again and select New Query.
2. In the Add Table window (figure 9-7), select the tables you want to query (you can use CTRL+click to select more than one) and click Add. The selected tables are displayed in the Diagram pane, the Criteria pane, and as part of the SQL pane.



CUSTOMERS (PUBLIC)
ORDERS (PUBLIC)
PLANTS (PUBLIC)
VENDORS (PUBLIC)

Figure 9-7. Selecting tables in the Add Table window.

3. Click Close to close the Add Table window.
4. Select columns for the query by selecting columns in the Diagram pane (figure 9-8). The SQL pane displays the revised query.
5. Execute the query by selecting Query Designer > Execute SQL from the menu. The retrieved columns and rows are displayed in the Results pane.



Column	Alias	Table	Outp...	Sort Type	Sort Order	Filter	Or...	Or...
OR_NUMBER		ORDERS ("...	✓					
OR_VENDOR		ORDERS ("...	✓					
OR_ITEM		ORDERS ("...	✓					
IN_ITEMID		PLANTS ("...	✓					

```
SELECT     "PUBLIC".ORDERS.OR_NUMBER, "PUBLIC".ORDERS.OR_VENDOR, "PUBLIC".ORDERS.OR_ITEM, "PUBLIC".PLANTS.IN_ITEMID,
             "PUBLIC".PLANTS.IN_NAME
FROM       "PUBLIC".ORDERS INNER JOIN
             "PUBLIC".PLANTS ON "PUBLIC".ORDERS.OR_ITEM = "PUBLIC".PLANTS.IN_ITEMID
```

Figure 9-8. Selecting columns in the Diagram pane.

Generating and manipulating an EDM

The Synergy/DE Data Provider for .NET enables you to generate an entity data model (EDM) for a Synergy database, manipulate it, and use it to access Synergy data. In this example, we will generate an EDM and change some scalar properties.

1. In Visual Studio, create a C# console application project. (Select File > New > Project, and when the New Project window opens, select "Visual C#" > "Windows" > "Console Application." Then click OK.)

2. In Solution Explorer, right-click the project name, and then select Add > New Item from the context menu. The Add New Item window opens.

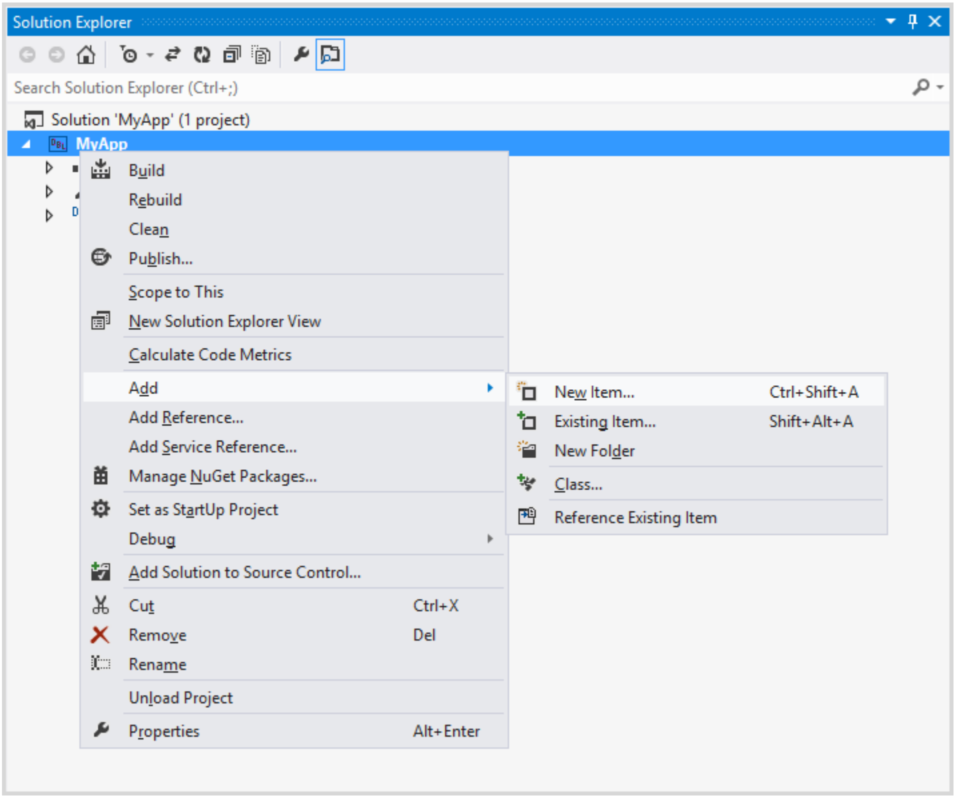


Figure 9-9. Selecting Add > New Item from the context menu for the project node.

3. In the Add New Item window, select Visual C# Items > Data, and then select "ADO.NET Entity Data Model. Then enter a name for the EDM in the Name field (or leave the default name), and click Add. The Entity Data Model Wizard opens.

4. Select "Generate from database" in the first screen of the wizard, and then click Next and set up the EDM in subsequent wizard screens by doing the following:

   - In the "Choose Your Data Connection" screen, use the "Which data connection…" field to specify the data connection you created for the sample database (in "Adding a data connection for the sample database" on page 9-35). Specify whether you want sensitive data stored in the connection string (for this example, select Yes). And then, in the "Save entity connection settings…" field, enter a name for the settings saved in the **App.Config** file for your project. Then click Next.
   - In the "Choose Your Database Objects and Settings" screen, select all four tables: Customers, Orders, Plants, and Vendors. Then specify a namespace for the model in the "Model Namespace" field, clear the "Pluralize or singularize…" and "Include foreign key columns…" options, and click Finish.

     The wizard and the Add New Item window close, and the EDM is generated. The design-time representation of the model is displayed in the Entity Model Designer as a tab in the Visual Studio content pane, and files are added to the project. This includes the EDMX file for the model and a file containing the classes for each entity type. See "Generating an EDM for a Synergy database" on page 9-15 for details.

At this point, you can write queries against the model by using Entity SQL or LINQ to Entities (see "Using an EDM to query Synergy data" on page 9-43). First, however, we'll take advantage of one of the primary benefits of having an EDM, which is the ability to rework the EDM into a form that's more suitable for the programming context. In this example, we'll simply rename scalar properties, but you can do many other things with an EDM, such as remove scalar properties, delete entity types, add and remove associations, create entity types that inherit from other entity types, create entity types that include scalar properties from more than one table (vertically partitioned entity types), and create entity types that represent a subset of the rows included in a table (horizontally partitioned entity types).

Renaming scalar properties

To simplify the queries we'll write in "Using an EDM to query Synergy data" on page 9-43, we'll rename some of the scalar properties (which represent columns in the database) with more easily recognizable names.

1. In Visual Studio, make sure the design-time representation of the model is visible in the Visual Studio content pane. (If it isn't, double-click the *model_name*.edmx node in Solution Explorer.) Then right-click on IN_NAME in the PLANTS entity, and select Rename from the context menu.

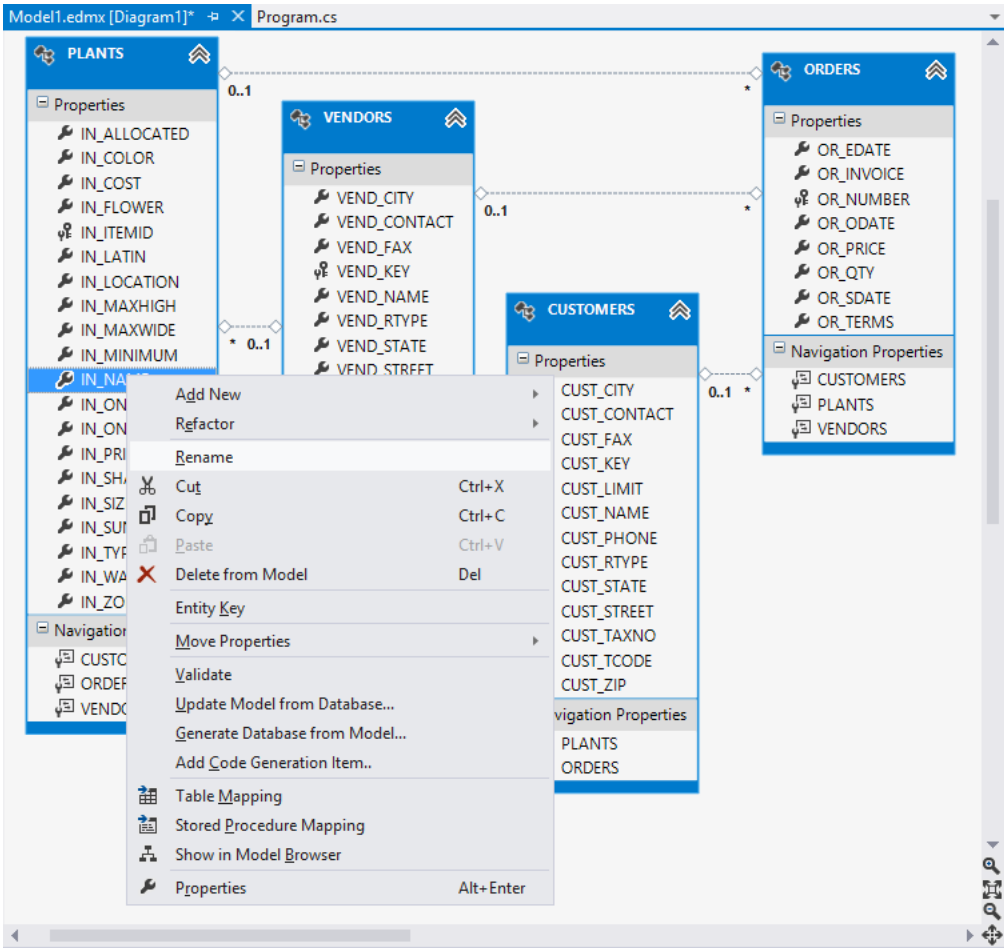


Figure 9-10. Renaming the IN_NAME scalar property.

2. Change the name to NAME.

3. Use the same process to change IN_ITEMID to ID and OR_QTY (in ORDERS) to QUANTITY.

Using an EDM to query Synergy data

Now that we've generated an EDM and simplified some scalar property names, we can query the EDM (and thereby query the database) by using Entity SQL or LINQ to Entities.

We'll start by creating a small C# program with an Entity SQL query.

1. Using the project you used when you created the EDM in "Generating and manipulating an EDM" on page 9-40, replace the code in the **program.cs** file with the code below, which contains an Entity SQL query. (To open the **program.cs** file, double-click it in the Solution Explorer.) Note the following:

   - Make sure the namespace is set to the namespace for your program.
   - Make sure that "name=" in the "using (EntityConnection...)" line is set to the name of the connection string in the **App.Config** file. (In the code below, this is set to Entities because when this example was created, the **App.Config** file had name="Entities".) And make sure the same name is used to qualify the tables in the query (e.g. "Entities.Orders").

   Here's the code with the Entity SQL query:

```
using System;
using System.Data;
using System.Data.EntityClient;

namespace MyApp
{
  class Program
  {
    static void Main(string[] args)
    {
      using (EntityConnection conn = new
             EntityConnection("name=Entities"))
      {
        //The following query should be on one line.
        string sql = "SELECT P.NAME, SUM(O.QUANTITY) FROM Entities.ORDERS
AS O, Entities.PLANTS AS P WHERE O.PLANTS.ID = P.ID GROUP BY P.NAME";
        conn.Open();
        EntityCommand comm = new EntityCommand(sql, conn);
        try
        {
          EntityDataReader edr =
            comm.ExecuteReader(CommandBehavior.SequentialAccess);
          while (edr.Read())
          {
            Console.WriteLine(edr.GetValue(0).ToString() +
                              " " + edr.GetValue(1).ToString());
          }
          edr.Close();
          }
```

```
                catch (Exception ex)
                {
                  Console.WriteLine(ex.Message);
                }
                Console.ReadLine();
            }
        }
    }
}
```

2. From the Debug menu of Visual Studio, select Start Debugging. You should get the following output in the console:

   ```
   English Daisy 70
   European Hackberry 124
   Fountain Butterfly Bush 130
   Lemon Verbena 40
   Paper Mulberry 120
   ```

   If you instead see an error in the console or in Visual Studio, use the error information to determine how to modify your code. For example, the following error most likely indicates that the name of the connection string in **App.Config** is something other than Entities.

   ```
   'Entities.ORDERS' could not be resolved in the current scope or context.
   Make sure that all referenced variables are in scope, that required sche-
   mas are loaded, and that namespaces are referenced correctly, near multi-
   part identifier, line 1, column 38.
   ```

3. Close the console application window. (If the application ran correctly, you can do this by pressing ENTER.) If you got an error in the previous step, correct the code and try step 2 again.

4. Try step 1 through step 3 with the following code, which includes LINQ to Entities queries. As it is, it uses the expression query syntax. If you change the commenting, it uses the method-based syntax. (You may also need to change commenting for the Entity Framework version.)

   - Make sure the namespace is set to the namespace for your program.
   - For Entity Framework 5.0 and higher, make sure the “using” line specifies the name of the EDM container:

     ```
     using (container context = new container())
     ```

     For Entity Framework versions prior to 5.0, make sure the “using” line specifies the name of the EDM container and the name of the connection string as follows:

     ```
     using (container context = new container("name=connection_string_name"))
     ```

     You can find the name of the EDM container by opening the **.Designer** file for the EDM (under the **.edmx** node in Visual Studio Solution Explorer). The name of the first class in this file is the name of the container. You can find the name of the connection string in the **App.Config** file as noted in step 1 above.

Here's the code:

```
using System;
using System.Linq;

namespace MyApp
{
  class Program
  {
    static void Main(string[] args)
    {
      //The following works for Entity Framework 5.0 and higher:
      using (Entities context = new Entities())

      //The following works for Entity Framework versions prior to 5.0. To
      //use it, uncomment it, and then comment the using statement above.
      //using (Entities context = new Entities("name=Entities"))

      {
        //This query uses the expression query syntax. Comment it if
        //you want to use the method-query below.
        var q = from o in context.ORDERS
                group o by o.PLANTS.NAME into op
                select new { op.Key, Total = op.Sum(o => o.QUANTITY) };

        //This query uses the method-based syntax. To use it, uncomment
        //it and comment the above query.
        /*var q = context.ORDERS.GroupBy(o => o.PLANTS.NAME).Select
                (op => new { op.Key, Total = op.Sum(o => o.QUANTITY) });*/
        try
        {
          foreach (var item in q)
          {
            Console.WriteLine(item.Key + " - " + item.Total.ToString());
          }
        }
        catch (Exception ex)
        {
           Console.WriteLine(ex.Message);
        }
      }
      Console.ReadLine();
    }
  }
}
```

10

Optimizing Data Access

Efficient queries and updates are the product of more than just well-constructed SQL statements. They are the result of a well-planned repository, sufficient and well-managed resources, in addition to SQL statements that take advantage of *xf*ODBC's ability to optimize SQL statement processing. This chapter introduces some of the issues that affect performance—particularly those that relate to *xf*ODBC. You should also refer to some general SQL reference works, "Appendix B: SQL Support", the documentation for your ODBC-enabled application, and "System Catalog Caching" on page 8-18.

Optimizing with Keys

Optimization starts with the design of your database and repository (see "Setting Up a Repository" on page 3-2). For *xf*ODBC to quickly process SQL statements, the data files and the repository should have well-chosen keys, keys that reflect the way users access data.

What are keys?

A *key* is a portion of a record structure that individually identifies records and enables records to be quickly accessed and sorted. For ISAM files, a key can be a portion (*segment*) or a group of separate portions of the record structure. For repositories, a key can be a single field or a group of fields. The fields that make up a repository key are also called *segments*.

Keys are created at two different points in your database's development: when you create the data files and when you define the repository for those files. When you create an ISAM file, for instance, you define the primary key and any secondary keys for that file. When you create a relative file, the record number is automatically defined as the primary key (this is the only key that can be defined for a relative file). ASCII sequential files, on the other hand, don't have keys.

In addition to defining keys when you create data files, you also define keys when you define the repository that describes your data files. At this point, there are two types of keys you can create: access and foreign. *Access keys* correspond to the keys you created when you created the data file, are used to locate and sort records, and can be used to define relationships between tables. *Foreign keys*, however, are *not* keys in the data file, but can be used to create relationships between tables. (Note that foreign keys are useful only with ODBC-enabled applications that support the ODBC API function SQLForeignKeys.)

For information on defining keys for ISAM files and relative files, see the "Synergy DBMS" chapter of *Synergy Tools*.

How *xf*ODBC uses keys

As *xf*ODBC processes an SQL statement, it looks to the database's system catalog for indexes (keys) it can use to speed the processing of the statement. To determine which key to use, *xf*ODBC evaluates two types of SQL clause: restriction clauses and sort clauses. Restriction clauses include WHERE, HAVING, and JOIN clauses. Sort clauses include ORDER BY and GROUP BY clauses.

To evaluate a restriction clause, *xf*ODBC attempts to match the columns (field names) in the restriction clause with the key's segments. To evaluate a sort clause, *xf*ODBC matches the sort clause's order (ASC or DESC) with the key's segment order and matches the columns in the sort clause with the key's segments. If no key can be used with the sort clause, *xf*ODBC will then create a temporary table sort, which results in many more I/O operations and poorer performance.

Note the following:

- *xf*ODBC evaluates JOINs before other restriction clauses.
- *xf*ODBC evaluates restriction clauses before ORDER BY clauses. If for the restriction clause *xf*ODBC uses a key that includes the field specified for the ORDER BY, and that field is the first segment in the key, *xf*ODBC won't perform an additional sort because the result set will already be in the correct order.
- If there is an ORDER BY clause but no restriction clause, *xf*ODBC may choose a key based on the ORDER BY to avoid a sort of the result set.
- *xf*ODBC will optimize a sort clause only if there is no join, the clause has a constant predicate, and the clause has one of the following operators: >, >=, <, or <=.
- For *xf*ODBC to use a key for a restriction or sort clause, the clause doesn't need to have a column for every segment in the key, but it must contain columns for one or more contiguous segments starting with the first segment. For example, to use the fourth segment of a key that has four segments—seg1, seg2, seg3, and seg4—there must be columns for seg1, seg2, and seg3 as well as seg4. If, for example, a clause has columns that correspond only to seg1 and seg3, *xf*ODBC can use the first segment, seg1, of the key, but not the third, seg3.
- If *xf*ODBC cannot use a key with either a sort or a non-join restriction clause, *xf*ODBC uses the primary key as a sequential read key. This means the driver must read the entire file.
- For inner joins, *xf*ODBC creates a temporary index for each join table whose columns are not part of any key. Each temporary index includes all of the join columns for its table. You can prevent *xf*ODBC from creating this temporary index by setting TMPINDEX to OFF. (For information, see SET OPTION on page B-57.)

  If you use *xf*ODBC with inner joins that result in temporary indexes, note that your data files must use static RFAs if update, store, or delete operations will occur while the *xf*ODBC driver is using a temporary index to access a file.
- A key can have a literal segment in any position. See "Keys with literals" on page 10-6.
- If a key contains a field that overlays other fields, **dbcreate** creates an alternate index made up of the individual fields that make up the overlay (if it overlays more than one field). See "Indexes for overlay segments" below.
- Crystal Reports 9 and higher do not support foreign keys. These versions of Crystal Reports will not automatically use foreign keys to optimize a query, so users must understand database relationships in order to optimize joins.

Indexes for overlay segments

If a structure has a key with an overlay segment, **dbcreate** creates an index that includes the non-overlay fields (i.e., the fields that the segment overlays) if the overlay field

- overlays more than one primary field.
- does not have an offset.

This index is named as follows in GENESIS_INDEXES: *KeyName*_GENIX_*n* (where *n* is an ascending numeric value). For instance, if a key has an **a22** segment that overlays four fields (as in the following example), **dbcreate** creates an index consisting of the four fields (custid, date, code, and text) in addition to an index consisting of seg1.

```
record
    custid   ,d4
    date     ,d6
    code     ,d2
    text     ,a10

record  ,x
    seg1     ,a22    ;key
```

This would look like the following in GENESIS_INDEXES:

KEY1
KEY1_GENIX_0

If a structure has a tag index and a key named KEY_0 that overlays two fields, **dbcreate** will create the following indexes:

$_VTX_TAG_VIX_0001
KEY_0
KEY_0_GENIX_0

Note, however, that if a field is only partially included in an overlay, that field will be omitted from any GENIX index created for that overlay.

On the other hand, if multiple fields overlay a key segment defined as a primary field (the overlaid field)—the inverse of the first example—**dbcreate** will *not* create an index for the overlaying fields. Instead, it will create a single index for the key segment. For example, if four overlay fields—custid (a **d4**), date (a **d6**), code (a **d2**), and text (an **a10**)—overlay an **a22** segment (as in the following structure), **dbcreate** creates only one index, an index for the **a22** key field.

```
record
    key1     ,a22     ;key

record  ,x
    custid   ,d4
    date     ,d6
    code     ,d2
    text     ,a10
```

In this case, *xf*ODBC won't use an index for a statement with a restriction or sort clause for the custid, date, code, or text fields.

Finally, if multiple overlay fields are defined as a key (as in the following example), *xf*ODBC won't be able to use the primary field (the overlaid field) for optimization. For the following example, *xf*ODBC won't be able to use the key1 field for optimization.

```
record
    key1      ,a22

record  ,x
    custid    ,d4      ;key segment
    date      ,d6      ;key segment
    code      ,d2      ;key segment
    text      ,a10     ;key segment
```

Defining keys

Keep the following in mind when defining keys:

- Analyze your data. Remember that adding keys slows down the add/update process. Create keys only if they'll be used often with restriction clauses or sort clauses. Then write your SQL statements to use these keys. Note that it's better to define your keys to work with restriction clauses than sort clauses. A key used with a restriction clause has precedence over a key used with a sort clause. It's even better to define a key that works with both the restriction and sort clause. This improves performance by eliminating the sort phase.
- Define an access key for each key in the data files. *xf*ODBC recognizes keys only if they're defined in the system catalog, which is generated from the repository, not the data files.
- Avoid creating keys that contain null values. *xf*ODBC uses them only for join optimization.
- Avoid creating keys that are case insensitive or unsigned. *xf*ODBC can't use them for optimization.
- Avoid creating keys that have a date field with a rolling century (*RR*). *xf*ODBC can't use them for optimization.
- Not all date formats can be used as key segments. Only JJJJJ dates and dates with formats that begin with YYYY can be key segments. (Note, however, that the YYYYMMDDHHMISS date format listed in "Date and time fields" on page 3-15 should *not* be used as a key segment because you have to use a user type field to specify it.)
- Avoid creating access or foreign keys that contain user-defined fields. The *xf*ODBC query optimizer ignores them. However, there are three exceptions (i.e., cases where *xf*ODBC *can* use keys with user-defined fields):
    - a key with a user-defined date field that has a ^CLASS^ format
    - a foreign key with an alpha user-defined field
    - an access key with an alpha user-defined field whose key type override is set as alpha. (For information on setting the type override, see "Defining Keys" in the "Working with Files" chapter of the *Repository User's Guide*.)

Note the following:

- If a key contains multiple field segments, *xf*ODBC uses only those segments that are defined prior to a user-defined field segment.
- If a user-defined field is the first segment in a key, *xf*ODBC won't use the key.

- If the data file and the repository have different data types for the same segments, use the Type option (in the Key Definition window of Repository) to override decimal segments to alpha segments if the segments will have only positive values. This prevents **fcompare** (see "Validate, verify, and compare" on page 3-7) from generating warnings when it discovers that the key has a different type than some of its segments. If a segment can have negative values, the segment must be decimal, so do not override the segment data type. For information on the Type option, see "Defining Keys" in the "Working with Files" chapter of the *Repository User's Guide*.
- If a key segment is an overlay field, the overlay is not usually relevant to the query. Because of this, **dbcreate** creates an alternate index that consists of all the fields that make up the overlay. However, if a key segment is overlaid with multiple overlay fields, **dbcreate** will *not* create an index for the primary fields (the overlaid fields). See "Indexes for overlay segments" on page 10-3 for more information.
- If you plan to import data from your Synergy database to another database, such as Oracle or SQL Server, make sure that no keys are null. Many databases, such as SQL Server, do not allow null key fields.
- If a file has a tag field, we recommend including the tag as the first segment in each key for the file. See "Tags and optimization" on page 10-8 for more information.

Keys with literals

A key can have a literal segment in any position in the key. This means that for foreign keys, a literal can be used to correspond to a literal tag in a related table. For example, in the sample database included with the Connectivity Series distribution (the repository is in connect\synodbc\dict), the literal at the beginning of TAG_KEY_VEND in the ORDERS table enables a relation to be created between TAG_KEY_VEND and TAG_KEY, as illustrated in figure 10-1. Without this literal, we couldn't use the tag for the VENDORS table, and the keys wouldn't correspond. We need an equivalent segment as the first segment in TAG_KEY_VEND.

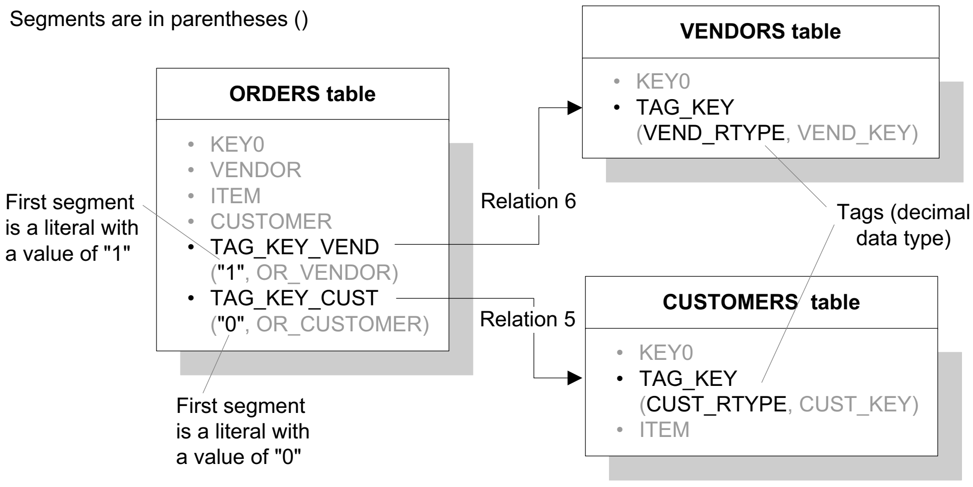

Segments are in parentheses ()
ORDERS table
• KEY0
• VENDOR
• ITEM
• CUSTOMER
• TAG_KEY_VEND ("1", OR_VENDOR)
• TAG_KEY_CUST ("0", OR_CUSTOMER)
First segment is a literal with a value of "1"
First segment is a literal with a value of "0"
Relation 6
Relation 5
VENDORS table
• KEY0
• TAG_KEY (VEND_RTYPE, VEND_KEY)
Tags (decimal data type)
CUSTOMERS table
• KEY0
• TAG_KEY (CUST_RTYPE, CUST_KEY)
• ITEM

Figure 10-1. Keys with literals in the sample database.

If the ORDERS and VENDORS tables had the following values there would be matches for all of the rows in the orders table except the row with the OR_VENDOR value of 42:

ORDERS table

OR_VENDOR	TAG_KEY_VEND (derived)
41	"1" + 41
42	"1" + 42
43	"1" + 43
44	"1" + 44

VENDORS table

VENDOR_R_TYPE	VEND_KEY
1	41
1	43
1	44
2	40

Likewise, all but one of the rows in the VENDOR table would have a match. The row with the VEND_KEY value of 40 wouldn't have a match because there is no row in the ORDERS table with an OR_VENDOR value of 40 and because this row has a VEND_R_TYPE value of 2, while the corresponding literal segment in TAG_KEY_VEND is always 1.

Tags and optimization

Keep the following in mind when defining tags or creating keys for a file with a tag:

- Make sure the tags won't be modified by user-defined data routines. The *xf*ODBC driver evaluates tags before invoking user-defined data routines, so the values produced by user-defined data routines won't be evaluated as tag values.
- If a file has a tag field, we recommend including the tag as a segment (preferably the first segment) in each key for the file. Note the following:
  - *xf*ODBC isn't able to use a key that consists solely of a tag for automatic optimization unless it's the first key in the file. We don't recommend creating such keys (though they can be used as part of a join, restriction clause, or sort clause).
  - Like any segment, a tag field can be used for optimization only if it's the first segment in the key or if all the segments that precede it are also part of the join, sort clause, or restriction that uses the equal (=) operator.
  - If a structure has more than one set of tag criteria joined with the OR connect option, *xf*ODBC won't be able to use a tag for optimization.

  If your repository has a set of keys that either do not include the tag or do not include it as the first segment, create a new key with the tag as the first segment, and then use the Repository option "Excluded by ODBC" to exclude similar keys that either have the tag in other positions or that do not have the tag at all. (For information on this Repository option, see "Defining Keys" in the "Working with Files" chapter of the *Repository User's Guide*.)

  Similarly, if your repository has two or more keys that are identical except for the inclusion or placement of a tag field, and one of the keys has the tag as the first segment (as we recommend), do *one* of the following to ensure that the key with the tag as the first segment is used for optimization:

  - Set the S/DE Repository option "Excluded by ODBC" for the keys that don't have the tag as the first field.
  - Make sure the key with the tag as the first segment is lower than or equal to the others in of key of reference (KRF) sequence. If it is equal, make sure it's first in the repository.
- If a tag is based on the Boolean operator EQ, **dbcreate** automatically creates an index with tag-related information. These are named as follows in the GENESIS_INDEXES file: $_VTX_TAG_VIX_*nnnn* (where *nnnn* is a numeric value, starting at 0001 for the first key).
- If a tag is based on an operator other than EQ, *xf*ODBC may not be able to use the tag to optimize queries. If it's based on the NE operator, *xf*ODBC will not be able to use it. If it is based on LE, LT, GE, or GT, *xf*ODBC will be able to use it only if it is the last tag for the structure and all prior tags are based on the EQ operator. If your repository has a tag that for one of these reasons cannot be used for optimization, add a new tag that uses EQ to the file. Then either use the new tag as the first segment for all keys used by *xf*ODBC for the file, or create a new set of keys that use the new tag, and use the Repository option "Excluded by ODBC" to exclude keys that use the unoptimizable tag.

Creating Efficient SQL Statements

Once your data files and repository are set up correctly, the next step in optimization is to make sure each query that accesses the database is well designed and able to take advantage of *xf*ODBC's ability to optimize. Although this section doesn't cover the entire subject of creating efficient SQL statements, it does discuss some basic rules and strategies.

- See "Appendix B: SQL Support" for information on supported SQL commands and syntax.
- See "How xfODBC uses keys" on page 10-2 for information on how *xf*ODBC uses indexes for sort and restriction clauses (including joins).
- See "Determining which indexes are used" on page 10-13 for information on tracking which indexes are used for a query.

Optimizing with restriction clauses

*xf*ODBC uses restriction clauses in two ways: to determine which rows to read (an *initial read* restriction) and to apply criteria to limit rows (a *limit* restriction). For optimization, initial read is the most important, but is available only if *xf*ODBC can use a key with the restriction clause. Depending on the restriction criteria and the key order, initial reads may enable *xf*ODBC to skip records. For example, assume you have the following restriction clause:

```
WHERE last_name >= 'Smith'
```

If there's an ascending key on **last_name**, the initial read starts with 'Smith'. Only records starting with 'Smith' and following will be read, which cuts down the number of reads and improves performance. As another example, say you have a statement with the following clause:

```
WHERE last_name <= 'Doe'
```

In this case, the read starts with 'Doe' back through to the beginning of the file. Once again, the number of reads is reduced, and performance is improved.

Operators and optimization

For relative files, *xf*ODBC supports optimization only for reads that use the equal operator (=) with the field record number. For example, *xf*ODBC can optimize the SQL statement in "Checking the order of the FROM clause for a SQL89 join" on page 10-10 because the last line of the WHERE clause checks for equality:

```
AND part_index.record_number = part.record_number
```

For ISAM and ASCII sequential files, *xf*ODBC supports optimization for reads that use any valid operator (>, >=, =, and so forth).

AND and OR clauses

If keys are available to optimize both sides of an OR clause, *xf*ODBC can optimize the clause. To do this, *xf*ODBC treats each side as a separate statement and then combines the results. If keys are not available to optimize both sides of an OR clause, *xf*ODBC cannot optimize the clause. For more information on OR clause optimization and the SQL command that controls it, see "Max number of rows" on page 8-9 and "Notes on MERGESIZE" on page B-61.

An optimizable OR clause is generally preferable to an AND clause, but if you can't state a restriction clause as an optimizable OR clause, it's generally better, when possible, to use AND clauses rather than an *un*optimizable OR clause. Because all conditions in an AND clause must be met, *xf*ODBC can use the first condition as an initial read if an index can be used with the condition. (See "Optimizing with restriction clauses" on page 10-9 for information on initial reads.) This may limit the number of rows *xf*ODBC is required to read and evaluate. On the other hand, if an OR clause can't be optimized, *xf*ODBC can't use the first condition of the clause as an initial read because a row can be included based on either side of an OR clause; *xf*ODBC must read and evaluate every row.

ORDER BY clauses

If a user-defined field is part of an ORDER BY clause, *xf*ODBC won't use a pre-defined key for optimization.

Checking the order of the FROM clause for a SQL89 join

If your SQL statement has a SQL89 join, the FROM clause is critical. The order of the tables listed in a FROM clause determines the order the tables are evaluated, the contents of the final result set, as well as the time required to generate the result set. The first table specified in the FROM clause is the primary table, and unless there's a restriction on the table, all rows in the primary table are selected. Rows in other tables are selected only if they meet the criteria specified in the WHERE clause. Because of this, you should order the tables in the FROM clause so that the tables are listed in the order that the rows relate to each other. For example, for the following SQL statement, assume that the **order_detail** and **part** tables are relative files with **record_number** as an index; in addition, assume the **part_index** table is an ISAM file with **part_number** as an index. Note that the primary table is **order_detail**.

```
SELECT
    order_detail.customer_number,
    order_detail.cust_name,
    order_detail.record_number,
    order_detail.billed_amt,
    order_detail.allowed_amt,
    order_detail.invoice_date,
    part_index.order_number,
    part.name
```

```
  FROM
    order_detail, part_index, part
  WHERE
    order_detail.order_number > 0
    AND order_detail.record_number > 1
    AND order_detail.part_number = part_index.part_number
    AND part_index.record_number = part.record_number
    ORDER BY
      order_detail.order_number
```

To process the above statement, *xf*ODBC reads **order_detail** (the primary table) sequentially from the first row to the last row by **record_number**. Each row of **order_detail** is tested against the restriction criteria. When a row meets the criteria (in this case **order_number** > 0 and **record_number** > 1), the associated row in **part_index** is located directly by **part_number**, and then the associated row in the **part** table is located directly by **record_number**. When the row is found, the items specified in the SELECT statement are written to a temporary sort file. Then, after reading the last row in the **order_detail** table, the temporary sort file is sorted by **order_number** and returned as the resulting set of data.

Avoid mixing SQL92 and SQL89 syntax

When writing a SQL92 join, avoid using a WHERE clause that uses a SQL89 inner join—i.e., matches columns (table1.field1 = table1.field2). This will result in a separate SQL89 join, which will generally reduce performance and may produce incorrect results. For example:

```
SELECT plants.in_itemid, orders.or_number
FROM plants
  LEFT JOIN orders
  ON plants.in_itemid = orders.or_item
WHERE plants.in_price = orders.or_price
```

This could be rewritten as the following, which can be optimized:

```
SELECT plants.in_itemid, orders.or_number
FROM plants
  LEFT JOIN orders
    ON plants.in_itemid = orders.or_item
      AND plants.in_price = orders.or_price
```

Using an ODBC-Enabled Application

When using an ODBC-enabled application to access a Synergy database, keep the following in mind:

- Don't rely on automatically created queries. These queries are seldom the most efficient way to access a database for a given situation. Instead, create your own queries that adhere to the guidelines listed in "Creating Efficient SQL Statements" on page 10-9.
- If you use an application, such as Microsoft Access, where selecting a table opens the entire table in a grid, use pass-through queries to update. If you update a field by selecting it in the grid, the application will update every field in the row including fields whose values haven't changed. *xf*ODBC does *not* support updates of this kind; they may cause an error.
- When using Microsoft Access to access Synergy data, consider whether you should import the database or link to the database. Depending on the query, the size of your database, requirements for sharing data, and the client's resources, one method may be better than another. For databases that rarely change (for example, databases consisting of ZIP codes, product codes, etc.), importing may improve performance. However, if you are modifying the database or need to see the most current picture of the database, you must use a link.

Optimizing with pass-through queries

One way to optimize performance when using Microsoft Access is to use pass-through queries, queries that are passed directly to the database server. If you use a pass-through query, the database server does all of the processing and returns only the result of the query. This bypasses the Microsoft Jet database engine. The reason this can improve performance is that Jet is key-based. So, for example, if you have the SQL statement "SELECT * FROM part", Jet will first read all of the keys; then it will perform a series of sequential statements like the following:

```
SELECT * FROM part WHERE part.number = <key>
```

For simple queries, this can result in many duplicate reads, making a pass-through query the more efficient alternative.

Tracking Performance

The first step in tracking performance is to determine which indexes (keys) are used for a query. A query's performance will often vary greatly depending on the keys used to optimize it. (See "Optimizing with Keys" on page 10-2 for information on creating well-chosen keys.) If you're still having performance problems, the next step is to use Synergy DBMS logging to see what calls the *xf*ODBC driver is making to the Synergy database.

Determining which indexes are used

To find out which indexes are used for a query, use SET OPTION PLAN in conjunction with SET OPTION LOGFILE. This generates a log file that includes a "Pushed key#" line that lists the index used to optimize the query, and it includes a table that lists index information. You can also use information from this log file to get key of reference (KRF) information. See "Notes on PLAN" on page B-61.

If you have an optimization issue that you plan to log with Synergy/DE Developer Support, use SET OPTION PLAN and SET OPTION LOGFILE.

Using Synergy DBMS logging

Synergy DBMS logging enables you to log the calls made from the *xf*ODBC driver to an ISAM database. By recording the number of reads to an ISAM file, this log can help you determine if an SQL statement is processed optimally. For example, assume you have a database with the following:

- Two tables: **order** with 8 rows and **plants** with 121 rows
- A valid **plant** table entry for each order
- An index defined for **plants.in_itemid**

To list all the order numbers and their associated plant names, you could create the following query:

```
SELECT orders.or_number, plants.in_name
    FROM orders, plants
    WHERE orders.or_item = plants.in_itemid
```

You could then use Synergy DBMS logging to find out how many reads result from the query. To do this,

1. Turn on Synergy DBMS logging. (See "Synergy DBMS logging" on page 11-8 for information.)
2. Run the query.

3. Open the resulting log file, find the open() statement for the **orders** table, and note the file handle identifier. In the following, for example, the file handle identifier is 218580a8:

   ```
   open(21580a8, I:I, 'XFDBTUT:orders')
   ```

4. Use the file handle identifier to count the number of reads() for the table—e.g.,

   ```
   reads(21580a8, '', rfa= 00, 100)
   ```

   Because of the order of the tables in the FROM clause in the above query, there are nine reads() for **orders** (eight successful and one unsuccessful). *xf*ODBC reads each row in the **orders** table and then attempts to read each **plants.in_itemid** that matches **orders.or_item**. With the **plants.in_itemid** index, *xf*ODBC is able to position to the first occurrence of **plants.in_itemid**, so that each **orders** table reads() reads the **plants** table only once. If the **plants.in_itemid** index is not unique (if duplicates are allowed), there will be two or more **plants** reads() entries for each **orders** table reads().

11

Data Access Errors and Error Logging

Error Logging

If you encounter errors when using an ODBC-enabled application to access a Synergy database, start by making sure your system is set up correctly (see "Troubleshooting Data Access" on page 9-4). You can then use *xf*ODBC logging (see list below) to diagnose the problem. Generally, it's best to use logging in a stand-alone configuration first. Then test your network connection. Finally, when you've got the stand-alone configuration and the network layer working correctly, re-create your client/server configuration and, if necessary, use logging to diagnose any remaining problems.

If you can't solve a problem by examining the log files, save the log files, collect all pertinent information about the problem, and contact Synergy/DE Developer Support. In addition to the log files, Support will need a description of the problem and the version numbers of all relevant software and hardware—especially the Synergy/DE version, operating system, and third-party ODBC-compliant application. For client/server configurations, supply this information for both the client and the server.

The following are the utilities and logging facilities you can use to debug problems and, in some cases, verify optimization with *xf*ODBC. See figure 11-1 on page 11-4 for a diagram that illustrates where the different types of logging fit in the data access process.

dltest	Lists needed Connectivity Series DLLs and states whether they can be found by *xf*ODBC. To use this utility, run it from the command line. The **dltest** utility is in the synergyde\connect directory. It has no options.
ODBC trace logging	Records ODBC API calls passed from the ODBC-enabled application to the ODBC Driver Manager. See "ODBC trace logging (Windows)" on page 11-5 for information.
Vortex API logging	Records API calls made by the *xf*ODBC driver on a Windows client. This enables you to see the exact SQL statement issued to the database, debug SQL statement errors, and verify optimization. See "Vortex API logging (Windows)" on page 11-5 for more information.
Vortex host logging	Records API calls made from the SQL OpenNet server to the Synergy database driver. See "Vortex host logging" on page 11-6 for information. Note that we recommend Vortex API logging instead of Vortex host logging. Vortex API and Vortex host logging generally record identical information, but Vortex API logging is easier to use. Additionally, on Windows, Vortex host logging can be used only with **vtxnet2**, not **vtxnetd**.
SET OPTION logging	Records information about indexes used to optimize a query (as well as internal information for use by Synergy/DE Developer Support). For information, see SET OPTION on page B-57 and "Creating a file for query processing options" on page 8-17.

Synergy driver logging	Enables you to determine if a system catalog is cached. "Using logging to determine if a system catalog is cached" on page 8-21.
Synergy DBMS logging	Records ISAM calls made from the Synergy database driver to your Synergy database. This enables you to debug open file errors, licensing errors, and connection failures. See "Synergy DBMS logging" on page 11-8 for information.
vtxping and **synxfpng**	Enable you to ping an SQL OpenNet server so you can verify that you can connect in a client/server configuration. **Vtxping** and **synxfpng** (when used with the **-x** option) are identical, except that **synxfpng** has a verbose option (**-v**) that lists socket calls as they succeed or fail, which can be useful when debugging. For more information, see "The vtxping Utility" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide* and "The synxfpng Utility" in the "Configuring xfServer" chapter of the *Installation Configuration Guide*.
vtxnetd/vtxnet2 logging	If you set the **log** option for either of these programs, a log file (**tcm_*pid*.log**) records connection requests and, if the program can't start a worker thread or process, logs the reason for the failure. You may be able to use this information to determine why a connection is failing in a client/server configuration. Note that this log also records ping and kill requests. For more information, see "The vtxnetd and vtxnet2 Programs" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.

Generally problems are caused by an ODBC-enabled application, such as Microsoft Query, sending unsupported SQL statements to the *xf*ODBC driver. (See "Appendix B: SQL Support" for information on the SQL statements supported by *xf*ODBC.) To test the validity of an SQL statement, you can use Microsoft ODBC Test (**Odbcte32.exe**). (You can also use this utility to test your setup. If you're able to connect with ODBC Test, your setup is working. See "Using ODBC Test to test a query" on page 9-33 for an example.)

In addition to the logging options listed above, we automatically set the environment variable VORTEX_HOST_SYSLOG, which instructs the SQL OpenNet sever to generate messages for the event log (Windows), **syslog** (UNIX), or the operator console (OpenVMS) when an attempt to connect to an SQL OpenNet server causes fatal errors. We don't recommend changing this setting. See VORTEX_HOST_SYSLOG in the "Environment Variables" chapter of *Environment Variables & System Options* for more information.

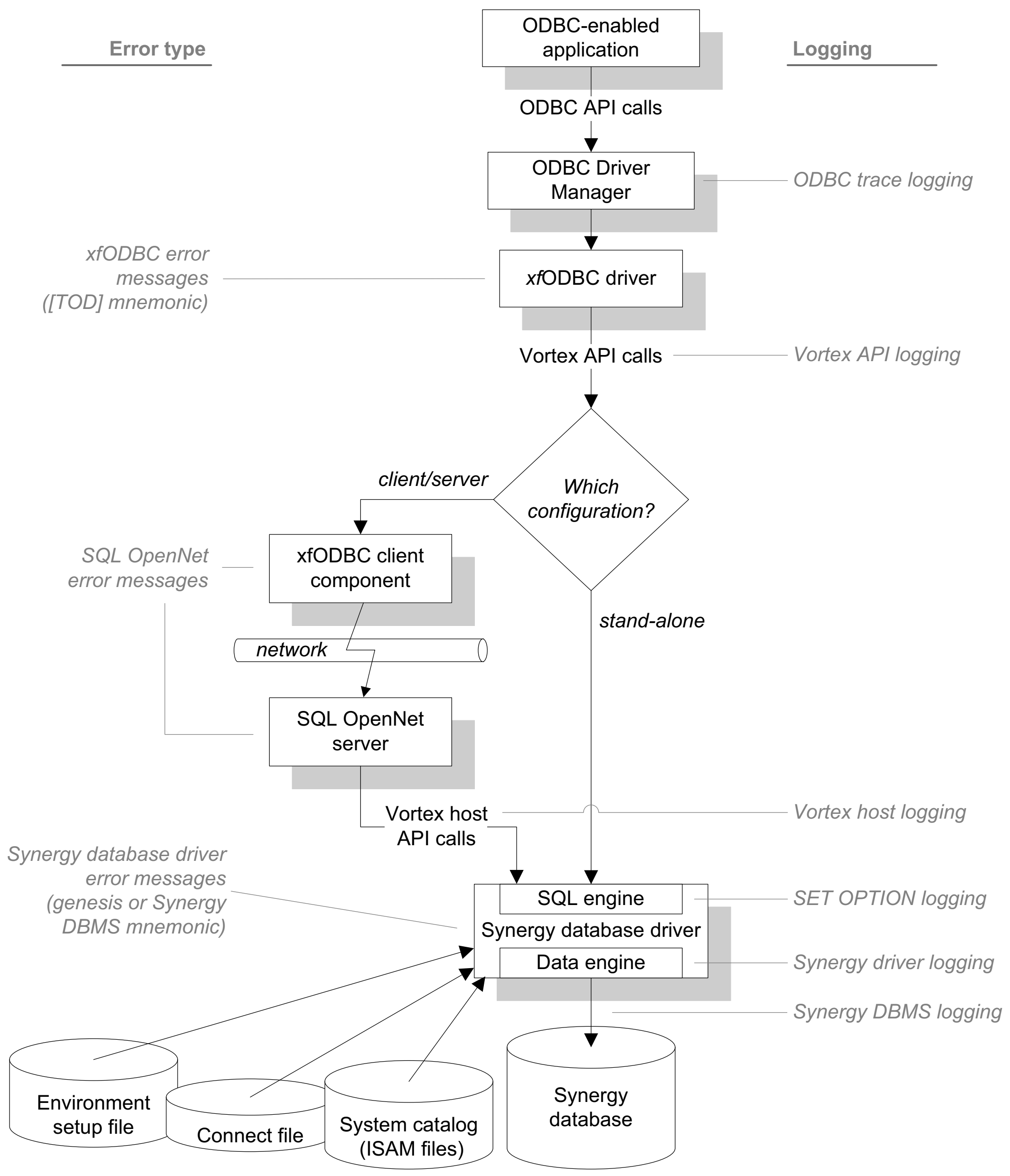


Figure 11-1. xfODBC error logging and messages.

Using the log files

ODBC trace logging (Windows)

ODBC trace logging records ODBC API calls passed from the ODBC-enabled application to the ODBC Driver Manager. We recommend that you use log files to debug in a stand-alone configuration. If you need to use ODBC trace logging in a client/server configuration, do this on the *client*.

1. Exit your ODBC-enabled application.
2. Open the ODBC Data Source Administrator available from Windows Control Panel.
3. Go to the Tracing tab and click the Start Tracing Now button.
4. Make a note of the log file name displayed on the Tracing tab so that you can find it later.
5. Re-run the application until you receive the error(s), and then examine the log file.
6. Once you have successfully logged the error, turn tracing off.

Vortex API logging (Windows)

Vortex API logging records statements issued to the database by the *xf*ODBC driver. (Note that connect strings are omitted.) We recommend that you use log files to debug in a stand-alone configuration. If you need to use Vortex API logging in a client/server configuration, set the environment variables on the *client*.

Note that Vortex API logging is not supported for multi-threaded applications.

1. Exit your ODBC-enabled application.
2. Use the VORTEX_API_LOGFILE and VORTEX_API_LOGOPTS environment variables to specify a name for the log and turn logging on. Note the following:
   - Set VORTEX_API_LOGFILE to the path and filename of the logfile you want to produce. Don’t specify an extension for the filename (or version number on OpenVMS). *xf*ODBC automatically appends the process ID (*filename_pid*) and an extension (**.log**). If you specify an extension on OpenVMS, no log file will be created.
   - Set VORTEX_API_LOGOPTS to one or more of the following. You can set more than one option by separating the options with a plus sign—for example, FULL+TIME.

APPEND	Appends logging information to existing log file
ERROR	Logs only statements with errors
FULL	Specifies full logging
PLAY	Sets an option that enables Synergy/DE Developer Support to playback an operation

RECORD	Logs data for Synergy/DE Developer Support
SQL	Creates a file that contains diagnostic information. The filename (minus extension) is specified with VORTEX_API_LOGFILE. The extension is **.sql**. (Client only)
TIME	Logs execution time for statements

Note that if you set VORTEX_API_LOGFILE without setting VORTEX_API_LOGOPTS, the log file will include a list of all operations along with a total count for each operation.

- Set these environment variables in the environment.
- For client/server configurations, set the environment variables on the client. (For services such as web servers that use the *xf*ODBC driver, you can use the Env. variables field in the xfODBC Setup window to set these environment variables on the client. For information, see "Adding a user or system DSN" on page 8-5.)

For example:

```
VORTEX_API_LOGFILE=c:\vortex
VORTEX_API_LOGOPTS=FULL
```

3. Re-run the application until you receive the error(s), and then exit the ODBC application.
4. Examine the log file.
5. Once you have successfully logged the error, turn logging off by unsetting the environment variables (and reboot if necessary). Logging slows performance, and the log files can quickly fill a disk.

Vortex host logging

Vortex host logging records statements (connect strings are omitted) passed to SQL OpenNet from the *xf*ODBC driver. Vortex host logging applies only to client/server configurations. Note that we recommend Vortex API logging instead of Vortex host logging. Vortex API logging and Vortex host logging generally record identical information, but Vortex API logging is easier to use. (In special cases, however, Synergy/DE Developer Support may instruct you to use Vortex host logging.)

Vortex host logging is not supported for multi-threaded applications, so use this only with **vtxnet2**, not **vtxnetd**.

1. Exit your ODBC-enabled application on the client.
2. Set the VORTEX_HOST_LOGFILE and VORTEX_HOST_LOGOPTS environment variables on the server to specify a name for the log and to turn logging on. For example:

```
VORTEX_HOST_LOGFILE=c:\vortex
VORTEX_HOST_LOGOPTS=FULL
```

Note the following:

- Set VORTEX_HOST_LOGFILE to the path and filename of the log file you want to produce. Don't specify an extension for the filename (or version number on OpenVMS). *xf*ODBC automatically appends the process ID (*filename_pid*) and an extension (**.log**). If you specify an extension on OpenVMS, no log file will be created.
- Set VORTEX_HOST_LOGOPTS to one or more of the following. You can set more than one option by separating the options with a plus sign—for example, FULL+TIME.

APPEND	Appends logging information to existing log file
ERROR	Logs only statements with errors
FULL	Specifies full logging
PLAY	Sets an option that enables Synergy/DE Developer Support to playback an operation
RECORD	Logs data for Synergy/DE Developer Support
SQL	Creates a file that contains diagnostic information. The filename (minus extension) is specified with VORTEX_HOST_LOGFILE. The extension is **.sql**. (Client only)
TIME	Logs execution time for statements

  Note that if you set VORTEX_HOST_LOGFILE without setting VORTEX_HOST_LOGOPTS, the log file will include a list of all operations along with a total count for each operation.
- On Windows, set these environment variables in the **opennet.srv** file before starting **vtxnet2**. Note that these environment variables do *not* work if you use **vtxnetd**. Use **vtxnet2** to enable logging.
- On UNIX, set these environment variables in your environment before starting **vtxnetd**.
- On OpenVMS, set these environment variables with system-wide logicals before starting the server program.

3. Go to the client, re-run the application until you receive the error(s). Then exit the application.
4. Go to the server and examine the log file to determine the problem. The log file will be named ***filename_pid*.log**, where *filename* is the name you specified with the VORTEX_HOST_LOGFILE variable and *pid* is the process ID.
5. Once you have successfully logged the error, turn logging off by unsetting the environment variables on the server. Logging slows performance, and the log files can quickly fill a disk.

Synergy DBMS logging

Synergy DBMS logging records ISAM calls made to your Synergy database from the *xf*ODBC driver. We recommend that you use log files to debug in a stand-alone configuration. If you need to use Synergy DBMS logging in a client/server configuration, set the environment variables on the *server*. For information on using Synergy DBMS to resolve performance issues, see "Tracking Performance" on page 10-13.

Synergy DBMS logging can significantly reduce performance. Use it for diagnostic purposes only; then turn it off.

Synergy DBMS logging on Windows and UNIX

1. Exit your ODBC-enabled application.
2. Set one or more of the following environment variables.

SDMS_AUDIT	Set this to the path and filename for the Synergy DBMS audit log file for non-server operations. Note: Use SDMS_AUDIT_SRV instead of SDMS_AUDIT on Windows (even in non-server situations) to audit threaded programs.
SDMS_AUDIT_FULL	To extend the logging output by adding the first 50 bytes of each record to the log file, set this variable to any value.
SDMS_AUDIT_MODE	To specify that I/O control modes are logged for each file operation, set SDMS_AUDIT_MODE to any value.
SDMS_AUDIT_SRV	Set this to the path and filename for Synergy DBMS audit logs for operations on a server or threaded Windows program. See SDMS_AUDIT_SRV in the "Environment Variables" chapter of *Environment Variables & System Options* for more information, and note that the thread ID and current time are appended to the extension for each log filename.
SDMS2_FULL	To record additional ODBC calls to the Synergy database, set SDMS2_FULL to 1. Use this variable with SDMS_AUDIT or SDMS_AUDIT_SRV.

Note the following:

- For stand-alone configurations, set these in the environment.
- On Windows, set these as system-wide environment variables before starting the SQL OpenNet server. This may require stopping and restarting the server. For example:

```
SDMS_AUDIT_SRV=c:\sdms
SDMS_AUDIT_FULL=1
SDMS2_FULL=1
```

- On UNIX, set these prior to starting the SQL OpenNet server. This may require stopping and restarting the server. For example, you can set them in the **setodbc** file:

```
SDMS_AUDIT=/usr2/sdms.log ;export SDMS_AUDIT
SDMS_AUDIT_FULL=1         ;export SDMS_AUDIT_FULL
SDMS2_FULL=1              ;export SDMS2_FULL
```

3. Run the ODBC-enabled application until you receive the error(s), and then exit the application.
4. Examine the log file (on the server in a client/server configuration) to determine the problem.
5. Turn logging off by unsetting the environment variables. Logging slows performance, and the log files can quickly fill a disk.

Synergy DBMS logging on OpenVMS

1. Set the following environment variables to specify a location for the log file and turn logging on.

SDMS2_LOG	Set this to the path and filename for the SDMS2 audit log file.
SDMS2_FULL	To record additional ODBC calls to the Synergy database, set SDMS2_FULL to 1.

These should be set with system-wide logicals. For example:

```
$ DEF/SYS/EXE SDMS2_LOG DEVICE:[DIRECTORY]FILE
$ DEF/SYS/EXE SDMS2_FULL 1
```

2. Stop and restart the SQL OpenNet server. For example:

```
$ SET DEF CONNECTDIR:
$ VTXKILL TIGER
$ @STARTNET
```

For information on stopping and starting the SQL OpenNet server, see "Starting and Stopping SQL OpenNet for xfODBC" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.

3. Go to the client system and run your application until you receive the error. Then completely exit the ODBC application on the client.
4. Go back to the server and use **vtxkill** to stop the OpenNet server. For example:

```
$ VTXKILL TIGER
```

5. Examine the log file. It should be in the location you specified with the SDMS2_LOG logical.
6. Turn logging off by unsetting the logicals. Logging slows performance, and the log files can quickly fill a disk.

Editing the SQL Message File

The SQL error message file contains messages that display when the *xf*ODBC driver, SQL OpenNet client, or SQL OpenNet server encounter errors. You can use the GENESIS_MSG_FILE environment variable to specify the location and name of the SQL message file. By default it is named **sql.msg** and is located in the connect\synodbc\lib subdirectory of the main Synergy/DE installation directory. For more information, see "Specifying the name and location of the error message file" on page 3-20.

Note that to generate a system catalog, *xf*ODBC must be able to locate the SQL error message file.

1. Open **sql.txt** in a text editor that can display hexadecimal numbers and end-of-line characters. For example, you can open this file in Synergy/DE Workbench.

   Sql.txt is installed to the connect\synodbc\lib subdirectory of the main Synergy/DE installation directory.

2. Edit the messages. Note the following:

   - A message can be only 64 characters long.
   - Each message must end with a newline character (ASCII hexadecimal character number 0x0a), and the newline character must come after the sixty-fourth character in the line. Any character within the first 64 character positions in a line will be part of the display.

3. Save the text file.

4. Move to the connect subdirectory of the main Synergy/DE installation directory. (If you are on a Windows system, open a Command Prompt window, and then change to the synergyde\connect subdirectory.)

5. From the command line, enter a command with the following syntax:

   - On Windows:

     `bldemf.exe` *source_file message_file*

   - On UNIX:

     `bldemf` *source_file message_file*

   - On OpenVMS:

     `$ BLDEMF` *source_file message_file*

   where *source_file* is the path and filename of the text file you saved in step 3 and *message_file* is the path and filename of the SQL message file to be created.

   You can now verify that the changes were made by opening the new SQL message file in a text editor. Note that this file is formatted differently than text file you saved in step 3; there are no newline characters. Close the SQL message file *without* saving it. If you have further changes to make to the error message file, follow step 1 through step 5 again.

Data Access Errors

This section lists errors you may encounter when accessing data from an ODBC-enabled application. For an illustration of where the different types of error occur in the data access process, see figure 11-1 on page 11-4.

- *xf*ODBC errors document problems on a client in a client/server configuration and problems in a stand-alone configuration.
- SQL OpenNet error messages document problems you may encounter in a client/server configuration.
- Synergy database driver (Synergy driver) error messages document problems you may encounter on a client in a client/server configuration and problems in a stand-alone configuration.

If you're using system catalog caching, see "System Catalog Caching" on page 8-18 for additional errors you may encounter.

For information on errors you may encounter while generating a system catalog, see "Dbcreate error and warning messages" on page 4-15. For information on DBA errors, see "DBA error messages" on page 6-7.

number: Unknown node (type: *name*)

(Synergy driver) This is an internal error. Turn on Vortex API logging (use the FULL option) and SET OPTION logging (use the LOGFILE, PLAN, and TRACE options), repeat the steps that caused the error, and then contact Synergy/DE Developer Support. (For information on Vortex API logging, see "Vortex API logging (Windows)" on page 11-5. For information on SET OPTION logging, see SET OPTION on page B-57.)

AUTHBAD: Invalid authentication syntax

(SQL OpenNet) The connect string syntax is invalid. Make sure that the user name and password follow the host name. Note that this user ID and password are for the host machine, *not* the database. See "Adding a user or system DSN" on page 8-5 for more information.

Authentication failed

(SQL OpenNet) User and password authentication failed on the server. If you're using a Windows server, make sure the user has "log on as a batch job" privileges on the server. In addition, make sure the user and password are correct in the connect string and encryption is set to the same value on both the client and the server.

AUTHFAIL: Authentication on *service* failed

(SQL OpenNet) You are not authorized to run the requested host service. Make sure the user name and password follow the host name, or contact your system administrator. Note that this user ID and password are for the host machine, *not* the database. See "Adding a user or system DSN" on page 8-5 for more information.

AUTHREQ: Host ‘*host_name*’ requires authentication

(SQL OpenNet) The host you are connecting to requires additional authentication. If the **-a** option is set for **vtxnetd** or **vtxnet2**, you must specify a username and password for an account on the machine or an account for a domain that the machine is part of (in addition to any database log-in information). If you’ve done this, make sure the username and password are correct. See “Adding a user or system DSN” on page 8-5 for information.

BADCONV: Data conversion failed (*hostvar*:*number*)

(*xf*ODBC and Vortex API) The requested data conversion failed. Check that the requested data is of the appropriate type. For example, this error occurs if you request a character column to be fetched into an integer and the column includes characters that aren’t numbers.

BADINI: *Filename* is either missing or invalid

(*xf*ODBC and SQL OpenNet) The **.ini** file is missing or its contents are invalid. Make sure the file exists and that it’s correct.

BLOBCOL: Invalid BLOB column ID

(SQL OpenNet) The specified column is not a BLOB column. Check the SQL statement.

BLOBFILE: BLOB file operation (*name*) failed

(SQL OpenNet) I/O error processing BLOB column. This usually occurs on an insert or update.

Buffer overflow: *message*

(Synergy driver) The SQL UPDATE statement caused a buffer overflow. Check *message* for Synergy DBMS error information.

CANCEL: Operation cancelled

(*xf*ODBC) The operation was cancelled by the driver manager. (This is an informational message.)

CANFREE: Cancel treated as SQLFreeStmt with SQL_CLOSE

(*xf*ODBC) No processing was being done on the statement, so the call was treated as a call to SQLFreeStmt with the SQL_CLOSE option. Function returns SQL_SUCCESS_WITH_INFO. (This is an informational message.)

Cannot add temporary index

(Synergy driver) Attempt to create a temporary index to optimize a table join operation failed. Make sure you have write permission and enough available disk space.

Cannot allocate context: *message*

(Synergy driver) The SQL statement could not allocate a Synergy DBMS context. Contact Synergy/DE Developer Support.

Cannot begin transaction: *message*

(Synergy driver) The SQL COMMIT/ROLLBACK statement cannot be performed. Check *message* for Synergy DBMS error information.

Cannot create '*name*': *message*

(Synergy driver) The specified file cannot be created. Check *message* for Synergy DBMS error information.

Cannot define *default_index*: *message*

(Synergy driver) The SQL statement references a table whose file creation failed. Check *message* for Synergy DBMS error information.

Cannot delete from '*name*': *message*

(Synergy driver) The SQL DELETE statement cannot be performed. Check *message* for Synergy DBMS error information.

Cannot drop system tables

(Synergy driver) The system tables, tables that begin with GENESIS_, cannot be dropped. For a list of the system tables, see "System catalog" on page 1-5. Make sure the table in the DROP TABLE statement is not a system table.

Cannot end transaction: *message*

(Synergy driver) The SQL COMMIT statement cannot be performed. Check *message* for Synergy DBMS error information.

Cannot insert into '*name*': *message*

(Synergy driver) The SQL INSERT statement failed. Check *message* for Synergy DBMS error information.

Cannot open '*name*' for update: *message*

(Synergy driver) The specified file cannot be opened for update. Check *message* for Synergy DBMS error information.

Cannot open file *filename error_message*

(Synergy driver) The *xf*ODBC driver can't open the specified file. The path may be wrong, or the file specification may include a missing environment variable or an environment variable with an invalid setting. Alternatively, the file may not exist, permissions may not be set correctly, or the repository file definition may include an invalid character. Be sure to check the spelling and location of *filename*.

Cannot update ‘*name*’: *message*

(Synergy driver) The SQL UPDATE statement failed. Check *message* for Synergy DBMS error information.

Catalog table ‘*name*’ corrupted or out of date

(Synergy driver) The specified GENESIS catalog table cannot be read. It has either been modified directly, or it was created from an unsupported version of **dbcreate**. Regenerate the system catalog with the current version of **dbcreate**.

Character array too big (max: *max size*)

(Synergy driver) The SQL statement contains a character array that is too big. The maximum size is *max size*. Correct the statement.

Column (*#*) is out of range

(Synergy driver) This may indicate that a GROUP BY clause doesn’t include all columns in the select list. Correct the GROUP BY clause to include all columns in the select list. See GROUP BY on page B-22 for more information.

Column ‘*name*’, 8 byte integer not supported on this platform

(Synergy driver) The platform does not support 8-byte integers. Remove any reference to 8-byte integers from the system catalog.

Column *name* already defined

(Synergy driver) The SQL CREATE TABLE/VIEW statement has duplicate column names. Correct the statement.

Column: *‘name’*, DBL decimal overflow

(Synergy driver) The SQL statement is attempting to join a decimal column to a column whose value is too large for the decimal column. (For example, you’ll see this error if you try to join a **d1** column to a column with the value 45; this value is too large for a **d1** column.) Correct the statement.

Column: ‘*name*’, integer overflow

(Synergy driver) An overflow occurred while converting a number to an integer. Correct the system catalog entry for the column.

Column ‘*name*’, invalid date data: *data*

(Synergy driver) The date is not valid. Correct this in your data file.

Column ‘*name*’ not deleted from catalog: *message*

(Synergy driver) The specified column cannot be deleted from the catalog. Check *message* for Synergy DBMS error information.

Column *name* undefined

(Synergy driver) The SQL statement references a column that is not defined in the system catalog. Correct the statement or define the column/table in the system catalog. Note that this can happen a GROUP BY clause includes an alias for a column in an inline view. See GROUP BY on page B-22 for more information.

Column: '*name*', Unsupported data type: *type*

(Synergy driver) The system catalog has an unsupported data type entry for column *name*. Check the system catalog definition for this column.

CONFIG: Expected a CONFIG call

(SQL OpenNet) This is an internal error that typically indicates that the **vtx4** process terminated abnormally. Collect any relevant information from the event log (Windows), **syslog** (UNIX), or the operator console (OpenVMS), and then contact Synergy/DE Developer Support. You may be asked you to use Vortex host logging and/or Synergy DBMS logging to assist. (See "Vortex host logging" on page 11-6 and "Synergy DBMS logging" on page 11-8.)

Connect: Unknown Error

(TCP/IP socket error) Although a connection was gracefully closed by the server, the client was not prepared for the closing of the connection. This is generally caused either by a version mismatch or by network latency issues where the final packet sent by the server is not received before the default server socket shutdown is initiated. This might occur, for example, if the initial connect fails with an error. See the event log (Windows), **syslog** (UNIX), or the operator console (OpenVMS) for information on the problem.

CONNECT-1: Authorization failure

(Synergy driver) The connection failed because of the user's security level, the length of the user name, a user or group authentication failure, the length of the password, or because the password is missing.

CONNECT-2: Dictionary access failure

(Synergy driver) The connection failed because the driver was unable to read the system catalog. Check the GENESIS_HOME setting and the path specified in the connect file.

CONNECT-3: No license available

(Synergy driver) Connectivity Series is not licensed.

CONNECT-4: Init failure see logfile

(Synergy driver) There is an error in the connect file syntax. For information on the cause, create a Synergy DBMS log file by setting the SDMS_AUDIT and SDMS2_FULL environment variables. For more information, see "Synergy DBMS logging" on page 11-8.

Connect:errno:*error*

(TCP/IP socket error) This error indicates either that the connection to the server has been closed, or that the *xf*ODBC driver can't make a connection to the SQL OpenNet server. See "Connection reset by peer (10054 or 54)" on page 11-32 and "Connection refused (10061 or 61)" on page 11-33.

Could not open INITSQL file '*file_name*'

(*xf*ODBC) *xf*ODBC could not locate the file *file_name*, which contains SQL statements.

'CREATE INDEX' not valid for this table type

(Synergy driver) The SQL CREATE INDEX statement is valid only for ISAM files. Make sure the table specified in the CREATE INDEX statement is an ISAM file.

Create view column count mismatch (create: *number*, select *number*)

(Synergy driver) The SQL CREATE VIEW statement's column list does not match the number of columns in the SELECT statement's select list. Correct the statement.

CURDUP: Duplicate cursor name

(*xf*ODBC) The cursor name is already in use. Specify a different name.

Data source name not found and no default driver specified

(*xf*ODBC) The specified DSN doesn't exist or there is a problem with it. Open the Microsoft ODBC Administrator and make sure the DSN exists and is configured correctly.

Note that if you are on a 64-bit Windows machine, it could be that the DSN has not been defined by the right version of the Microsoft ODBC Administrator. For example, if a 32-bit application is accessing your Synergy database via *xf*ODBC, and the system has both 32-bit and 64-bit Synergy/DE, the DSN must be created by the 32-bit ODBC Administrator. And if Visual Studio or an application that's part of or based on Visual Studio (such as Business Intelligence Developer Studio for SQL Server) is accessing your Synergy database, you'll need identical 32-bit and 64-bit DSNs. See "Adding a user or system DSN" on page 8-5.

Data truncation (max: *max_size*)

(Synergy driver) Data has been truncated. Call Synergy/DE Developer Support.

DATATRUNC: Data truncated

(*xf*ODBC) The data has been truncated. Either the data specified is too long or supplied output buffers are too small. Modify the size of the output buffers. (You can do this with the "Fetch buffer size" field in the xfODBC Setup window. See "Adding a user or system DSN" on page 8-5 for information.)

DB error: *error_name*

(Synergy driver) This error is reported by the database. See your database documentation for more information.

DIALOG: Dialog box failed

(*xf*ODBC) The connect dialog box failed. Notify your system administrator.

Divide by ZERO

(Synergy driver) An expression in the SQL statement caused division by zero.

DLLENTRY: Could not find DB driver entry point

(SQL OpenNet) The loaded DLL or shared library does not contain the expected entry point. This typically happens when the wrong DLL or shared library has been loaded and occurs only on machines that support DLLs or shared libraries. Call Synergy/DE Developer Support.

DLLLOAD: Could not load *name*

(SQL OpenNet and Vortex API) One of the Connectivity Series components can't load a needed DLL or shared library. The specified DLL or shared library may be missing, it may be invalid (incorrectly named or an incorrect version), its file specification may be missing from PATH (on Windows) or from the library path (on UNIX), it may not be able to access third-party DLLs or shared libraries it needs, or if you're on UNIX, the setuid (+s) bit may be set for **vtxnetd** or VTX4. DLLLOAD errors occur only on machines that support either DLLs or shared libraries, and these errors are generally caused by a problem with the way Connectivity Series is installed or, on UNIX, by a failure to run **setsde** correctly.

To troubleshoot, run the **dltest** utility from the command line. (The **dltest** utility is in the connect directory and has no options or arguments.) This utility indicates whether needed Connectivity Series DLLs or shared libraries can be accessed, and if you're on UNIX, it tells you the name of the library path environment variable (for example, SHLIB_PATH on HP-UX 32-bit or LIBPATH on IBM AIX 32-bit). In addition, note the following:

- If you are on UNIX, make sure the setuid (+s) bit is not set for **vtxnetd** or VTX4. The setuid bit prevents the library path environment variable from being used. This will cause DLLLOAD errors—though it won't affect the ability of **dltest** to access needed **.so** files. (Note that as distributed, the setuid bit is *not* set for **vtxnetd** and VTX4, so this won't be a problem unless you've added it.)
- Make sure all of the resources that the DLL or shared library needs are available. For example, if you get a DLLLOAD error for **GDS0.DLL**, it may be that Connectivity Series can't find one of the DLLs required by **GDS0.DLL**. (These include **SDMS22.DLL** and **VTXIPC.DLL**.) On Windows, you can use the Dependency Walker utility **(depends.exe)** to determine which resources are required for a DLL. (You can download Dependency Walker from **http://www.dependencywalker.com**.) On most UNIX systems, you can use the **ldd** command to determine which resources are required for a shared library.

For SQL OpenNet, it may be that **setsde** isn't run before **vtxnetd** attempts to implement the SQL OpenNet server. Check the rc file and make sure **setsde** is run before **startnet**. For information on which Synergy shared libraries are causing the error, run **dltest** from the rc file (directly before the **startnet** command).

DLLLOAD: Invalid client DLL version. (expected *version#*, found *version#*)

(*xf*ODBC, SQL OpenNet) A DLL on the client machine is out of date—for example, you could get this error if you have an outdated version of **tod32.dll**.

DRVCONF: Driver not configured

(SQL OpenNet) This is an internal error that typically indicates that the **vtx4** process terminated abnormally. Collect any relevant information from the event log (Windows), **syslog** (UNIX), or the operator console (OpenVMS), and then call Synergy/DE Developer Support.

Error: 8 byte integer not supported on this platform

(Synergy driver) Your platform does not support 8-byte integers. Remove the 8-byte field definition from your repository and regenerate your system catalog.

Error: Could not connect to request pipe (Error:6)

(SQL OpenNet, Windows only) The system catalog you are attempting to load into memory has already been cached.

EXECFAIL: Exec *program_name* failed on host *host_name*

(SQL OpenNet) The service (program) specified in the network connection string could not be started. This occurs when the service cannot be found, does not have the correct permissions, or is not listed as a valid service.

- Make sure the service exists, is listed as a valid service, and that you are connecting with the correct user name and password.
- Use **vtxnetd/vtxnet2** logging and check the resulting **tcm_*pid*.log** file. See "The vtxnetd and vtxnet2 Programs" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide* for information.
- Check your operating system documentation for information on the error number (*nnn*).

Fetch error: *message*

(Synergy driver) The SQL statement caused a fetch error. Check *message* for Synergy DBMS error information.

File '*file_name*' already exists

(Synergy driver) The SQL CREATE TABLE statement references a table that is already created. Make sure the specified table does not already exist.

File ‘*file_name*’ cannot be removed: *message*

(Synergy driver) The specified file cannot be removed. Check *message* for Synergy DBMS error information.

File *file_name* could not be opened

(Synergy driver) The file *file_name* was not found. If *file_name* is **synodbccache.dat**, this error may indicate that the account you’re using can’t be used to unload the system catalog from the cache. A system catalog can be unloaded only by the user (account) that cached it.

File ‘*file_name*’ does not exist

(Synergy driver) The SQL statement references a table stored in a file that does not exist. It may have been deleted by another user using a different system catalog. Disconnect and then reconnect.

File error TRIWRT ‘No space left on device’

(Synergy driver) The sort (order by) work file is out of disk space. Allocate more space on your disk drive.

FLIPOVER: Flip buffer overflow

(SQL OpenNet) This error occurs if too many parameters are specified. The current limit is approximately 250 parameters. Note that multiple dimensions are not included in this limit.

Format error in ‘*datafiles*’

(Synergy driver) The datasource line of the connect file is not defined correctly. See chapter 5, “Setting Up a Connect File.”

FUNCSEQ: Function sequence error

(*xf*ODBC) The sequence of function calls is invalid. Make sure you follow the sequence specified in Microsoft’s *ODBC Programmer’s Reference*. (See the section on ODBC state transition.)

Function *function_name* not implemented yet

(Synergy driver) The function has not been implemented. Call Synergy/DE Developer Support.

‘GENESIS_HOME’ environment variable not found

(Synergy driver) The GENESIS_HOME environment variable is not set. For information on setting this variable, see “Specifying the connect file location (GENESIS_HOME)” on page 3-19.

HOSTNOTFOUND: Host *host_name* not found

(SQL OpenNet) The host you’re trying to connect to has not been found. Make sure the spelling of the host name is correct.

If any numeric operand is NULL then only '==' and '!=' are valid

(Synergy driver) The SQL statement's WHERE clause uses and invalid operator with a null value. When comparing a null value, only IS NULL, =NULL, IS NOT NULL, and <>NULL are valid. (For example, SALARY > NULL is invalid.) Correct the statement.

Illegal data: *data*. Expected format: *format*

(Synergy driver) *Data* is not correctly formatted.

Illegal DECODE format (from,val,code,val,code...[,default])

(Synergy driver) DECODE format is incorrect. Check the DECODE syntax in your Oracle documentation.

Illegal format specification: '*format_qualifier*'

(Synergy driver) The format qualifier *format_qualifier* is not correct.

Illegal number of parameters for builtin function

(Synergy driver) The SQL statement has the wrong number of parameters for the built-in function. Correct the statement.

Illegal parameters for function *function_name*

(Synergy driver) Call Synergy/DE Developer Support.

Index '*name*' not deleted from catalog: *message*

(Synergy driver) The specified index cannot be deleted from the system catalog. Check *message* for Synergy DBMS error information.

Index column '*column_name*' not deleted from catalog: *message*

(Synergy driver) The specified index column cannot be deleted from the system catalog. Check *message* for Synergy DBMS error information.

INTERNAL: *operation_number*

(Synergy driver) Call Synergy/DE Developer Support with the *operation_number.*

Invalid connect syntax (uid/pwd/datasource)

(SQL OpenNet) The DSN is invalid or there's a problem with encryption. Make sure the DSN is configured correctly, and make sure both client and server are running versions of Connectivity Series that support encryption (version 8.1 and later). In addition, make sure the encryption setting in **net.ini** (on the client) matches the encryption setting on the server (set with the **vtxnetd/vtxnet2 -k** option), make sure the **net.ini** file is not corrupt, and make sure VORTEX_HOME is set to the correct directory. (Note that if you install both 32-bit and 64-bit Connectivity Series on the same 64-bit Windows machine, the last version installed determines the VORTEX_HOME setting by overwriting the previous setting.) See "SQL OpenNet Client Options in net.ini" on page 8-26.

Invalid integer

(SQL OpenNet, *xf*ODBC) The number specified for the encryption key is invalid or the **net.ini** file is corrupt. Verify that the key is set to an integer value in the correct range, and make sure the **net.ini** file has no control characters.

Invalid parameter

(Synergy driver) An invalid parameter was sent for the command. Make sure the parameters you use are valid.

Invalid password

(Synergy driver) The password is incorrect. Note that passwords are case sensitive. Make sure the spelling and case are correct.

Invalid predicate result (NULL or invalid datatype)

(Synergy driver) The SQL statement's WHERE clause returned a null or a result with an invalid data type. Correct the statement.

Invalid USERID and/or PASSWORD

(*xf*ODBC) This indicates that you've entered an invalid user name or password or that there may be a problem with the users or groups in your system catalog. Start by checking the spelling and case of the user name and password you used to access the database. (These are case sensitive.) If that isn't the problem, open your system catalog in DBA and check make sure the users and groups are set up correctly (see "Viewing groups" on page 6-14 and "Viewing users in a group" on page 6-16). If these don't reveal the problem, check encryption settings on the server and the client and make sure VORTEX_HOME is set to the correct directory (see "SQL OpenNet Client Options in net.ini" on page 8-26).

INVARG: Invalid argument

(*xf*ODBC) An argument specified for an ODBC API function is invalid. Consult your ODBC documentation for the correct syntax.

INVAUTH: Invalid authorization

(*xf*ODBC) The user is not authorized to connect to the specified data source. Check your user name and password.

INVBUFLEN: Invalid string or buffer length

(*xf*ODBC) The length specified is invalid. Negative values, such as SQL_NTS, have special meaning, but not all negative values are valid. Check your ODBC documentation for valid length specifiers.

INVCOLNUM: Invalid column number

(*xf*ODBC) The specified column number is out of range. Use the correct column number.

INVCURNAM: Invalid cursor name

(*xf*ODBC) The specified cursor name is invalid. Refer to your ODBC documentation for the maximum allowed length.

INVCURSTA: Invalid cursor state

(*xf*ODBC) The state of the cursor (OPENed, CLOSEd, and so forth) is not valid for the current operation. Make sure you've followed the necessary steps before calling this function. See the ODBC state transition section of Microsoft's *ODBC Programmer's Reference* for information.

INVDATA: Invalid data

(*xf*ODBC) The data for the specified operation is invalid. Verify the data.

INVDATE: Invalid date/time

(SQL OpenNet) The format of the date and/or time data is invalid. Verify the data. See "Formats for returned dates and times" on page 8-13 for information.

INVDRVVER: DB version mismatch (expected: *driver_name*, found *version_number*)

(SQL OpenNet) The version of the database driver is not at the same level as the SQL OpenNet runtime library. This error is most common when SQL OpenNet client/server is being used, but can also occur if an older driver has been linked with a newer runtime library. Make sure the client and the server are running the same version of Connectivity Series.

INVHOSTSYN: Invalid host/service name syntax

(SQL OpenNet) The *host*/*service* syntax is incorrect. Correct the syntax.

INVNUM: Invalid (internal) number

(SQL OpenNet) The data being converted is invalid. Check the data.

INVVER: NET version mismatch (host: *host_ver*, client: *client*_ver)

(SQL OpenNet) The version of SQL OpenNet on the server is different than the version on the client. Make sure both client and server use the same version of Connectivity Series.

KEEPALIVE: Setting SO_KEEPALIVE failed

(SQL OpenNet) This indicates that the socket option KEEPALIVE failed or was not set. Call Synergy/DE Developer Support.

Licensing error

(*xf*ODBC) On OpenVMS, this error may indicate that there is insufficient memory or other resources. Open STARTNET.COM and make sure that the buffer, file_limit, page_file, queue_limit, and subprocess options are set according to the recommendations in STARTNET.COM. Or, as an alternative, you can check the following OpenVMS **sysgen** parameters and make sure their settings are equal to or greater than the settings listed below. (Note, though, that we recommend using the STARTNET.COM settings.)

PQL_DPGFLQUOTA 164593
PQL_DENQLM 800
PQL_DASTLM 256
PQL_DBIOLM 128
PQL_DDIOLM 128

If one of these parameter settings is missing, or if any of the settings don't meet these minimums, add a MIN_*xxx* setting or modify the setting in **MODPARAMS.DAT** for your system. For example, set

```
MIN_PQL_DENQLM=800
```

Then use **AUTOGEN.COM** to re-configure your system and re-boot.

LINGER: Setting SO_LINGER failed

(SQL OpenNet) This indicates that the socket option LINGER failed or was not set. Call Synergy/DE Developer Support.

MANYCONN: Too many connections

(*xf*ODBC and SQL OpenNet) The limit for concurrent ODBC handles has been exceeded. *xf*ODBC supports a maximum of 1024 concurrent ODBC handles. However, system limitations may reduce the number *xf*ODBC can use. (Handles are typically have a one-to-one correspondence with connections. However, and ODBC program could incorrectly close connections without deallocating handles.)

For OpenVMS systems, **STARTNET.COM** is distributed with settings that allow approximately 10 concurrent handles (which are processes on OpenVMS). For information on changing these settings, see the comments in **STARTNET.COM**.

MANYSTMT: Too many statements

(*xf*ODBC) The number of allowable statements has been exceeded. Either increase the limit in the Statements field in the *xf*ODBC Setup dialog box or free any statements you are not using or do not need.

MISSENV: Missing environment variable

(*xf*ODBC) The GENESIS_HOME or VORTEX_HOME environment variable is missing. Make sure these environment variables are set. For more information, see "Specifying the connect file location (GENESIS_HOME)" on page 3-19 and "VORTEX_HOME" on page A-7.

Missing column separator (row: *row_number*, col: *column_number*)

(Synergy driver) A column separator is missing at row *row_number*, column *column_number.*

Missing string delimiter (row: *row_number*, col: *column_number*)

(Synergy driver) A string delimiter is missing at row *row_number*, column *column_number.*

Multi RID overflow

(*xf*ODBC) This generally indicates that MERGESIZE is set too a value that's too small for the query. Increase this setting by using SET OPTION or the "Max number of rows" field in the xfODBC Setup dialog box (the dialog box that enables you to add and configure *xf*ODBC DSNs). Then try the query again. See "Notes on MERGESIZE" on page B-61 for more information.

No data source

(Synergy driver) The connect string does not specify a connect file. For information on connection strings, see "Building Connect Strings" in the "Creating SQL Connection Programs" chapter of the *SQL Connection Reference Manual*.

No datasource specified

(Synergy driver) The connect file does not contain a datasource specification (which specifies the location of the data files). For information, see chapter 5, "Setting Up a Connect File."

No dictionary source directory defined

(Synergy driver) The connect file does not contain a dictsource definition. For information, see chapter 5, "Setting Up a Connect File."

No directory defined

(Synergy driver) The connect file does not contain a datasource definition. For information, see chapter 5, "Setting Up a Connect File."

NOCONN: Not connected

(*xf*ODBC and SQL OpenNet) A connection must be performed before any other operation.

NOCURNAM: No cursor name available

(*xf*ODBC) A cursor name has not been assigned. Assign a cursor name.

NODSN: No DSN specified

(*xf*ODBC) No data source name (DSN) was specified. Make sure the connect string specifies a DSN.

NOINTR: Host cannot be interrupted

(Synergy driver) Your ODBC-enabled application called SQLCancel, but the *xf*ODBC driver is not in a position to cancel its operation.

NOMEM: Out of memory

(SQL OpenNet and *xf*ODBC) This is a fatal error. Either there is no more heap memory available (which is rare), or the heaps have been corrupted. Notify your system administrator.

Non aggregates require a GROUP BY expression

(Synergy driver) This generally indicates that the SELECT statement contains aggregate and non-aggregate select list items, which requires a GROUP BY expression. Use GROUP BY or change the statement's structure.

If the statement has a GROUP BY clause, this could indicate that a column specified in the GROUP BY isn't in the select list. All columns in a GROUP BY expression must be in the select list for the statement.

If you're using the Synergy/DE Data Provider for .NET, this could indicate that you are running a version of Connectivity Series (or *xf*ODBC Client if it's a deployment machine) that doesn't support the Synergy/DE Data Provider for .NET. Support was added in Synergy/DE 9.1.5b, so make sure your system has 9.1.5b or higher.

Not implemented yet

(Synergy driver) The function, statement, or subroutine has not been implemented.

NOTCAP: Driver not capable

(*xf*ODBC) *xf*ODBC does not support the requested capability.

NOTIMP: *feature_name* not implemented

(*xf*ODBC) The feature has not been implemented yet. Use ODBC API logging to find out more about the missing feature, and then call Synergy/DE Developer Support.

NOWHDL: No window handle available

(*xf*ODBC) No window handle is available to open the connect dialog. Notify your system administrator.

NULL not allowed for column

(Synergy driver) Either the SQL statement (INSERT or UPDATE) specifies a null value for a column that cannot accept null values, or a column that cannot accept null values has been omitted from an INSERT statement that affects that column's table. Correct the statement. (Note that omitting a column from an INSERT statement is equivalent to specifying a null value for the column. See INSERT on page B-47.)

Number of columns does not match number of values

(Synergy driver) The SQL INSERT statement's values do not match the number of columns defined for the table or listed in the column list of the SQL statement. Correct the statement.

ODBC call failed, [TOD] [ODBC] [GENESIS] Synergy DBMS: Table 'DBA.GENESIS_COLUMNS' not found [#31] or ODBC call failed, [TOD] [ODBC] [GENESIS] Synergy DBMS: Fetch error, Insufficient memory for attempted operation

(*xf*ODBC) This error indicates that there is insufficient memory or other resources on OpenVMS. For example, when trying to access data from a client machine with an ODBC application, such as Microsoft Access, you may be able to link to the first table you select, but get this error when you select a second table. Or you may get this error on subsequent attempts to link to the tables you've selected. See "Licensing error" on page 11-23 for information on resolving this.

Only '=' is allowed with ROWID

(Synergy driver) The SQL statement's WHERE clause contains an invalid ROWID predicate. Correct the statement.

Operands not compatible

(Synergy driver) The specified operands cannot be used together and/or cannot be used on this type of data. Correct the statement.

Operation requires *named* authorization

(Synergy driver) The SQL statement requires the specified authorization. Make sure you have the authority to issue the statement.

OPTCHG: Option value changed

(*xf*ODBC) The value of an option has changed. This is an informational message.

Out of memory (Save all work then exit and restart)

(Synergy driver) There is insufficient system memory available for the requested operation. Save all work; then exit and restart.

PARMCNT: Wrong number of parameters

(*xf*ODBC) The number of parameters specified does not match the number of parameters required by the statement. Modify your program to use the correct number of parameters for the statement.

PARMDTY: Invalid parameter data type

(*xf*ODBC) The data type specified for the parameter is unknown. Consult the ODBC documentation for valid data types.

PARMNUM: Invalid parameter number

(*xf*ODBC) The parameter number specified is out of range. Verify that your program uses the correct parameter number.

(position: *position_number*) - End of buffer reached

(Synergy driver) The SQL statement ended prematurely. Check the syntax for the command you are using.

(position: *position_number*) - Ending quote missing

(Synergy driver) The ending quotation mark is missing at position *position_number* in the SQL statement. Add the missing quotation mark.

(position: *position_number*) - Identifier too long

(Synergy driver) The SQL statement contains an identifier at position *position_number* that is too long. Identifiers are limited to 30 characters. Rename the identifier.

(position: *position_number*) - Illegal character

(Synergy driver) The SQL statement contains an illegal character at the given position. Check the statement.

(position: *position_number*) - String too long

(Synergy driver) The SQL statement contains a string that is too long at position *position_number.* Use a bind variable.

Premature end of line (row: *row_number*)

(Synergy driver) There is a premature end of line at row *row_number.*

Read error: *message*

(Synergy driver) The SQL statement caused a read error. Check *message* for Synergy DBMS error information. (See "Synergy DBMS Errors" in the "Error Messages" chapter of *Synergy Tools*.)

RECEIVE INFO ERROR: No such file or directory

(SQL OpenNet) The socket connection disappeared. The server database component crashed or has been terminated. Call Synergy/DE Developer Support.

Recv:errno:*error*

(TCP/IP socket error) This error indicates either that the connection to the server has been closed, or that the *xf*ODBC driver can't make a connection to the SQL OpenNet server. See "Connection reset by peer (10054 or 54)" on page 11-32.

Recv: Unknown Error

(TCP/IP socket error) Although a connection was gracefully closed by the server, the client was not prepared for the closing of the connection. This is generally caused either by a version mismatch or by network latency issues where the final packet sent by the server is not received before the default server socket shutdown is initiated. This might occur, for example, if the initial connect fails with an error. See the event log (Windows), **syslog** (UNIX), or the operator console (OpenVMS) for information on the problem.

SERVNOTFOUND: Service/Protocol *name* not found

(SQL OpenNet) The service or protocol cannot be found. Ensure that **vtxnetd** or **vtxnet2** is specified in your services file. If it isn't, you must either add it to the services file or specify the port number in the xfODBC Setup window. See "Adding a user or system DSN" on page 8-5 and "Configuring SQL Connection (client)" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide* for details.

SOCKET: Socket() failed

(SQL OpenNet) *xf*ODBC is unable to open a socket. The operating system may have run out of descriptors. Notify your system administrator.

Sort buffer overflow

(Synergy driver) This generally indicates that the memory available for SQL sort operations (ORDER BY or GROUP BY) is insufficient. Use SET OPTION SORTPAGES to increase the available memory. See SET OPTION on page B-57 for information.

Sort column *name* out of range (1 - *number*)

(Synergy driver) The ORDER BY clause for the SELECT statement references a column number that is out of range. Correct the statement.

Sub-query must return a single column

(Synergy driver) The SQL statement contains a subquery whose result set has more than one column. Correct the statement.

Synergy DBMS: File ‘GENESIS_USERS’ does not exist

(Synergy driver) This indicates that although the system catalog describes GENESIS_USERS, no GENESIS_USERS.ISM file exists for the catalog. To resolve this, regenerate your system catalog. The **dbcreate** utility no longer describes GENESIS_USERS or creates this file, so regenerating the system catalog prevents this situation.

Synergy DBMS: Cannot open “Filename”, No privilege to this directory

(Synergy driver) This may indicate that security levels will not allow the operation. Check access levels for tables and the user’s group to make sure the user has read and (if necessary) write access to the tables. See “Setting Security Levels” on page 8-2.

Synergy DBMS: LIST error. Index *index_number* not created

(SQL OpenNet) **Syngenload** did not create the system catalog cache correctly. To fix this, follow the instructions in “Correcting other caching problems” on page 8-25.

SYNLEV: Insufficient GENESIS syntax level

(Synergy driver) There is a version mismatch between the client and the server at the Synergy database driver (**VTX4**) level. Make sure that both sides of the network connection use the same version.

Table ‘*name*’ not deleted from catalog

(Synergy driver) The SQL DROP TABLE statement failed due to a Synergy DBMS error. Call Synergy/DE Developer Support.

Table ‘*name*’ still open by other cursors

(Synergy driver) The DROP TABLE statement references a table that is still being accessed by other cursors. Make sure all cursors that use the table are closed before issuing the DROP TABLE statement.

Table ‘*name*’ undefined

(Synergy driver) The SQL statement references a table that is not defined in the system catalog. Correct the statement to reference a defined table or define the table in the system catalog.

Table/View ‘*name*’ already in catalog

(Synergy driver) The SQL CREATE TABLE/VIEW statement references a table or view that is already defined. Check the tables/views specified in the statement.

TIDUSED: Statement already in use

(*xf*ODBC) Another thread is currently using the statement. Make sure your program is not using a statement that’s used by another thread.

to_char/date/number’s format mask must be a constant string

(Synergy driver) The SQL statement uses a data conversion function with a non-constant format mask string. Correct the statement.

Too many columns *number* (max: *max_number*)

(Synergy driver) The SQL statement references a table which exceeds the maximum number of columns. (*Number* is the number of columns in your system catalog, plus one for ROWID.) You can increase the maximum number of columns that *xf*ODBC allows by changing the Columns setting in the *xf*ODBC Setup dialog box. Note that changing the Columns setting does not affect the number of columns your ODBC-enabled application can handle. For information on the Columns setting, see “Columns” on page 8-8.

Too many columns specified

(Synergy driver) The SQL statement has too many columns defined. Correct the statement.

Too many cursors opened

(Synergy driver) Too many cursors are open at once. Close some cursors and retry.

Too many sort columns (max: *number*)

(Synergy driver) The SQL SELECT statement has too many columns in the ORDER BY clause. Reduce the number of columns in the ORDER BY clause.

Too many sub-queries at level *level_number* (max: *max_number*)

(Synergy driver) The SQL statement contains too many subqueries. Correct the statement.

Too many tables in SELECT (max: *max size*)

(Synergy driver) The SQL SELECT statement contains too many tables. Correct the statement.

UNDESTYP: Unknown descriptor type

(*xf*ODBC) The fDescType for SQLColAttributes() is unknown. Modify your program to use the correct descriptor.

UNFETTYP: Unknown fetch type

(*xf*ODBC) Currently, only SQL_FETCH_NEXT is supported. Modify your program to use only SQL_FETCH_NEXT.

UNINTYP: Unsupported InfoType: *type*

(*xf*ODBC) *Type* is not supported. Consult your ODBC documentation for valid values.

Unknown command

(Synergy driver) You’ve used an unrecognized command.

Unknown error code

(Synergy driver) This is an internal error. Turn on Synergy DBMS logging and Synergy driver logging, repeat the steps that caused the error, and then call Synergy/DE Developer Support. (For information on Synergy DBMS logging, see "Synergy DBMS logging" on page 11-8. For information on Synergy driver logging, see "Synergy Driver Logging" on page 5-5.)

Unknown executable node (type: $1)

(Synergy driver) Unexpected virtual machine code. This is an internal error and shouldn't occur. If it does, turn on Synergy driver logging and SET OPTION logging, repeat the steps that caused the error, and then call Synergy/DE Developer Support. (For information on Synergy driver logging, see "Using logging to determine if a system catalog is cached" on page 8-21. For information on SET OPTION logging, see SET OPTION on page B-57.)

UNOPT: Unknown option

(*xf*ODBC) The *xf*ODBC driver doesn't recognize the option. Consult your ODBC documentation for valid options.

UNUNOPT: Unknown Uniqueness option

(*xf*ODBC) The *xf*ODBC driver doesn't recognize the uniqueness option. Consult your ODBC documentation for valid values.

UNXACOPR: Unknown transaction operation

(*xf*ODBC) The *xf*ODBC driver doesn't recognize the transaction operation. Consult your ODBC documentation for valid values.

User does not have drop table permission

(Synergy driver) The SQL DROP TABLE statement cannot be performed by this user. Check the table specified in the statement.

VM Error: Virtual memory space exceeded

(Synergy driver) This generally indicates that the amount of memory available for SQL sort operations is insufficient. Use SET OPTION SORTPAGES to increase this amount (see SET OPTION on page B-57 for information).

Troubleshooting Socket Errors

Connection reset by peer (10054 or 54)

The "Connection reset by peer" socket error, which is 10054 (WSAECONNRESET) on Windows and generally 54 (ECONNRESET) on UNIX and OpenVMS, indicates that a connection to the server has been closed. This could be caused by a fatal error on the server, the server stopping, a network problem, or even a connection problem.

1. If the error has the form "connect:errno:*error*", use **vtxping** (or **synxfpng** with the **-x** option) to test your ability to connect to the server. Otherwise, skip to step 2. The **vtxping** and **synxfpng** utilities print reports to the screen. This information can be used by your network administrator to resolve TCP/IP network socket communication problems.
   - If you can connect, then the network, the server, and the Synergy/DE OpenNet Server service (**SynSQL**) are working. Continue with step 2.
   - If you can't connect, make sure the server is running, the **SynSQL** service is running on the server, and that you either specified the correct port number in the **vtxping** or **synxfpng** command or are using the default port.

   For information on **vtxping**, see "The vtxping Utility" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*. For information on **synxfpng**, see "The synxfpng Utility" in the "Configuring *xf*Server" chapter of the *Installation Configuration Guide*.

2. Check and correct the following, which may solve the problem if it is caused by network timing issues.
   - (Windows) If you're using network licensing, make sure License Manager is configured as a network server.
   - Make sure the server was rebooted after the Connectivity Series components were installed.
   - (Windows) On the server, make sure the GENESIS_HOME environment variable is set in the environment or in **opennet.srv** (not at the user level).
   - Check the system-level PATH on the server. It should include the connect directory.
   - Use **vtxnetd** or **vtxnet2** logging and check the resulting **tcm_*pid*.log** file. Then check the event log on Windows, **syslog** on UNIX, or the operator console on OpenVMS. (For information on **vtxnetd** and **vtxnet2** logging, see "The vtxnetd and vtxnet2 Programs" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*.)
   - On the server, make sure the file(s) for the Synergy database driver (**vtx4**) are in the connect directory. On Windows, these files are **vtx4.exe** and **vtx4.dll**. On UNIX, this is **vtx4.so**. On OpenVMS, this is **vtx4.exe**.
   - Make sure there is no more than one **vtxnet2** process running on the server. If there's more than one, use **vtxkill** to kill the processes; then restart the service.

3. If you are on Windows or UNIX, run the **dltest** utility to make sure Connectivity Series DLLs or shared libraries are loading properly. This is a command line utility (in the synergyde\connect directory) and has no options.

4. Make sure the dictsource and datasource lines in the connect file on the server have the correct settings. Then check the settings in the DSN. For example, if the DSN specifies a port, it should be the port that the SQL OpenNet server is running on. (If the SQL OpenNet server is running on the port that's the default for the client, the DSN doesn't need to specify a port. See the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide* for more information.)

5. If you get this socket error again, use **vtxnetd** or **vtxnet2** logging and check the event log on Windows, **syslog** on UNIX, or the operator console on OpenVMS.

Connection refused (10061 or 61)

The "Connection refused" socket error, which is 10061 (WSAECONNREFUSED) on Windows and is generally 61 (ECONNREFUSED) on UNIX and OpenVMS, indicates that the *xf*ODBC driver can't make a connection to the SQL OpenNet server. The SQL OpenNet server may not be running, it may not use the port that's specified in the DSN (or the default port if you didn't specify a port in the DSN), or the host specified in the DSN may be incorrect.

1. Use **vtxping** (or **synxfpng** with the **-x** option) to test your ability to connect to the server. (For information on **vtxping**, see "The vtxping Utility" in the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide*. For information on **synxfpng**, see "The synxfpng Utility" in the "Configuring *xf*Server" chapter of the *Installation Configuration Guide*.)

   The **vtxping** and **synxfpng** utilities print reports to the screen. This information can be used by your network administrator to resolve TCP/IP network socket communication problems.

   - If you can connect, then the network, the server, and the Synergy/DE OpenNet Server service (**SynSQL**) are working. Continue with step 2.
   - If you can't connect, make sure the server is running, the **SynSQL** service is running on the server, and that you either specified the correct port number in the **vtxping** or **synxfpng** command or are using the default port.

2. Make sure the host name in the DSN is correct.

   UNIX

   If you find that the SQL OpenNet server is running and the port is correct, it may be that server is terminating when the user that started it logs out. To run the server in the background and keep it running after you log out, use the **nohup** command. For example:

   ```
   nohup vtxnetd &
   ```

   For more information, see "Starting and stopping SQL OpenNet for xfODBC" in the UNIX section of the "Configuration Connectivity Series" chapter of the *Installation Configuration Guide*.

Appendices

Appendix A: Environment Variables

Lists environment variables used by *xf*ODBC as it generates a system catalog or accesses a database. Also lists other environment variables that are required and are automatically set during installation or configuration.

Appendix B: SQL Support

Lists the SQL statements, commands, and functions supported by *xf*ODBC driver, and documents how they are implemented for *xf*ODBC.

A

Environment Variables

Data Access Variables

The following environment variables set options that are used as the *xf*ODBC driver accesses the database. Unless otherwise noted, the *xf*ODBC Database Administrator (DBA) and **dbcreate** do not use these environment variables, and they do not affect the way the system catalog is generated. For information on environment variables that affect the system catalog, see "System Catalog Generation Variables" on page A-5.

Data Access Variables

This variable...	Does this...	For information, see...
GENESIS_HOME	Specifies the location of the connect file. (DBA and **dbcreate** also use this environment variable.)	"Specifying the connect file location (GENESIS_HOME)" on page 3-19
GENESIS_INITSQL	Specifies a file that contains predefined SQL statements.	"Creating a file for query processing options" on page 8-17
GENESIS_MSG_FILE	Sets the name and location of the error message file. (DBA and **dbcreate** also use this environment variable.)	"Specifying the name and location of the error message file" on page 3-20
SDMS_AUDIT	(UNIX only) Specifies the path and filename of the Synergy DBMS log file in UNIX environments. (Use SDMS_AUDIT_SRV for Windows.)	"Synergy DBMS logging" on page 11-8
SDMS_AUDIT_FULL	Turns on Synergy DBMS logging.	"Synergy DBMS logging" on page 11-8
SDMS_AUDIT_MODE	Instructs *xf*ODBC to log I/O control modes for each file operation.	"Synergy DBMS logging" on page 11-8
SDMS_AUDIT_SRV	(Windows and UNIX) Specifies the path and filename for generated auditing files for file operations on a server or a threaded Windows program.	"Synergy DBMS logging" on page 11-8
SDMS2_FULL	Records additional ODBC calls to the Synergy database. Unlike SDMS2_LOG, this environment variable can be set in Windows, UNIX, and OpenVMS environments.	"Synergy DBMS logging" on page 11-8 and "Synergy DBMS logging on OpenVMS" on page 11-9

Data Access Variables (Continued)

This variable...	Does this...	For information, see...
SDMS2_LOG	(OpenVMS only) Specifies the path and filename of the Synergy DBMS log file.	"Synergy DBMS logging on OpenVMS" on page 11-9
SODBC_INIFIL	Specifies the path and filename of the environment setup file.	"Specifying the location of an environment setup file" on page 3-20
SODBC_MCBA	Instructs *xf*ODBC to skip records that contain the MCBA deleted-record characters—four right brackets (]]]]) at the beginning or end of a record.	"Recognizing the MCBA deleted-record characters" on page 8-16
SYNBASEDATE	Sets the base date for conversion of date fields with the JJJJJJ format. The default date is 14 Sept. 1752. Note that once you've modified data in the database, don't change the SYNBASEDATE value or the database will be corrupted. Each date is stored as the difference between the date and the SYNBASEDATE value. Changing this value changes the way dates are stored and read.	"Setting the base date for Julian day conversions" on page 8-15
SYNCENTURY	Specifies the cutoff year for a sliding window mechanism that converts two-digit years to four-digit years. SYNCENTURY determines the cutoff year and affects only input dates with two-digit year formats (YYMMDD, YYJJJ, etc.).	"Converting dates returned without centuries" on page 8-14
TRIM_HOME	Specifies the location of the lib directory that contains **trim.ini** (UNIX only) and **trim.msg** files used for system catalog caching.	"Invalid parameter or argument (UNIX)" on page 8-22
VORTEX_API_LOGFILE	Turns on logging and specifies the path and filename of a log file used to log statements issued to the database by the *xf*ODBC driver.	"Vortex API logging (Windows)" on page 11-5
VORTEX_API_LOGOPTS	Specifies options for the file produced by VORTEX_API_LOGFILE.	"Vortex API logging (Windows)" on page 11-5

Data Access Variables (Continued)

This variable...	Does this...	For information, see...
VORTEX_HOST_LOGFILE	Turns on logging and specifies the path and filename of the logfile used to log statements passed to SQL OpenNet from the *xf*ODBC driver.	"Vortex host logging" on page 11-6
VORTEX_HOST_LOGOPTS	Specifies options for the file produced by VORTEX_HOST_LOGFILE.	"Vortex host logging" on page 11-6
VORTEX_HOST_SYSLOG	Instructs the SQL OpenNet server to generate messages for the event log (Windows), **syslog** (UNIX), or the operator console (OpenVMS) when an attempt to connect to an SQL OpenNet server causes fatal errors.	"Error Logging" on page 11-2
VORTEX_ODBC_CHAR	Determines how *xf*ODBC describes strings (SQL_VARCHAR or SQL_CHAR).	"Changing the way xfODBC describes strings" on page 8-16
VORTEX_ODBC_DATETIME	Determines how date/time columns are retrieved.	"Formats for returned dates and times" on page 8-13
VORTEX_ODBC_TIME	Determines how time columns are described.	"Formats for returned dates and times" on page 8-13 and "Time columns and ADO.NET" on page 9-20
VORTEX_SHM_BASE	Sets the base address for system catalog caching.	"System Catalog Caching" on page 8-18
VORTEX_SHM_FILE	Specifies the location and name of **synodbccache.dat**, a file used for system catalog caching.	"System Catalog Caching" on page 8-18

System Catalog Generation Variables

The following environment variables enable you to change the way system catalogs are generated and modified. Unless otherwise noted, only **dbcreate** and DBA use these variables. These variables do *not* affect the way the *xf*ODBC driver handles data. For information on environment variables that affect the way data is read from and written to the database, see "Data Access Variables" on page A-2.

System Catalog Generation Variables

This variable...	Does this...	For information, see...
GENESIS_HOME	Specifies the location of the connect file. (The *xf*ODBC driver also uses this environment variable.)	"Specifying the connect file location (GENESIS_HOME)" on page 3-19
GENESIS_MSG_FILE	Sets the name and location of the error message file. (The *xf*ODBC driver also uses this environment variable.)	"Specifying the name and location of the error message file" on page 3-20
RPSDAT	Specifies the location of the repository files. The repository files must be named **rpsmain.ism** and **rpstext.ism** for this environment variable to locate them.	"Specifying repository file locations" on page 3-22
RPSMFIL	Specifies the path and filename of the repository main file.	"Specifying repository file locations" on page 3-22
RPSTFIL	Specifies the path and filename of the repository text file.	"Specifying repository file locations" on page 3-22
SODBC_CNVFIL	Specifies both that a conversion setup file is to be used and the path and filename of the conversion setup file.	"Specifying a conversion setup file" on page 3-26
SODBC_CNVOPT	Instructs **dbcreate** (or DBA) to convert *all* fields in a structure, regardless of the report view flag setting in the repository.	"Including and omitting fields" on page 3-23
SODBC_COLLAPSE	Specifies when array elements should be compressed into a single column. Elements will be compressed into a single column when the number of elements is greater than or equal to the setting for this variable.	"Generating one column for an array field" on page 3-25
SODBC_DBA	Specifies the location of the DBA program.	"SODBC_DBA" on page A-7

System Catalog Generation Variables (Continued)

This variable...	Does this...	For information, see...
SODBC_NOGROUPNAME	Instructs **dbcreate** (or DBA) to omit group names, group prefixes, and struct field names from column names in the system catalog.	"Removing group and struct names from column names" on page 3-25
SODBC_NONULL	Deprecated. Determines how the "Null allowed" property for a system catalog column is set if the Repository "Null allowed" option for the repository field is set to Default.	SODBC_NONULL in the "Environment Variables" chapter of *Environment Variables & System Options*
SODBC_NOUNSIGNED	Instructs **dbcreate** (or DBA) to ignore the Repository field "Negative allowed?" and set all fields to signed unless they had validation ranges that were limited to positive values.	"Instructing dbcreate to ignore the "Negative allowed?" field in Repository" on page 3-24
SODBC_ODBCNAME	Instructs **dbcreate** (or DBA) to use Repository Alternate name field attribute as the column name.	"Renaming columns for clarity" on page 3-24
SODBC_TMPOPT	Instructs **dbcreate** (or DBA) to convert only those tables not attached to temporary files in the repository.	"Excluding tables attached to temporary files" on page 3-26
SODBC_TOKEN	Specifies a character for **dbcreate** (or DBA) to use to delineate position values in column names generated for elements in arrayed fields and groups.	"Changing the position delimiter used for arrays" on page 3-25
SODBC_USEFORMAT	Instructs **dbcreate** (or DBA) to use decimal information in the repository format string.	"Using decimal information in the repository format string" on page 3-27

Other Environment Variables Used by *xf*ODBC

The following environment variables are required and are automatically set by the installation, by the **setsde** script, or by the **SYS$MANAGER:CONNECT_STARTUP.COM** command file. See the "Configuring Connectivity Series" chapter of the *Installation Configuration Guide* for more information on the script and command files.

CONNECTDIR

The CONNECTDIR environment variable is set to the synergyde\connect directory. This environment variable is set

- by the installation on 32-bit Windows operating systems or when you run **dblvars64.bat** or **dblvars32.bat** to set up 64-bit or 32-bit versions of Synergy/DE on a 64-bit Windows operating system. (The installation does not set this environment variable on 64-bit Windows.)
- when you run **setsde** on UNIX or **SYS$MANAGER:CONNECT_STARTUP.COM** on OpenVMS.

SODBC_DBA

The SODBC_DBA environment variable is automatically set to the location of the DBA program. (The filename for the DBA program on UNIX and Windows is **xfdba.dbr**. On OpenVMS, it's **xfdba.exe**.) Although it's seldom necessary, you can change this setting. Note the following:

- SODBC_DBA is required and is automatically set when you install *xf*ODBC (except on 64-bit Windows systems).
- For Windows, SODBC_DBA must be set system-wide or in **synergy.ini** under the [synergy] heading.
- For UNIX, SODBC_DBA must be set by running the **setsde** script file (located in the synergyde directory).
- For OpenVMS, SODBC_DBA must be set by executing **CONNECT_STARTUP.COM**.
- For client/server configurations, SODBC_DBA must be set on the server.

VORTEX_HOME

The VORTEX_HOME environment variable specifies the location of a directory named lib that contains the **net.ini** file, a file used for encryption and other SQL OpenNet settings. The Connectivity Series installation (Windows), **setsde** (UNIX), or SYS$MANAGER:CONNECT_STARTUP.COM (OpenVMS) sets VORTEX_HOME to the connect\synodbc directory. Do not change this setting.

Note that because VORTEX_HOME is set at the system level, if you install both 64-bit and 32-bit versions of Connectivity Series on the same 64-bit Windows machine, the last version installed determines the VORTEX_HOME setting by overwriting the previous setting. So if you do change **net.ini**, you'll need to make sure you change the correct one.

VORTEX_HOST_HIDEGPF (Windows)

When set, the VORTEX_HOST_HIDEGPF environment variable prevents the SQL OpenNet server from shutting down if a thread fails. This is set in **opennet.std**, and we don't recommend changing it. See VORTEX_HOST_HIDEGPF in the "Environment Variables" chapter of *Environment Variables & System Options* for more information.

VORTEX_HOST_NOSEM

When set to 1, the VORTEX_HOST_NOSEM environment variable causes SQL OpenNet to crash when there's an access violation, enabling you to attach the Windows debugger. This should be used only at the request of Synergy/DE Developer Support. See VORTEX_HOST_NOSEM in the "Environment Variables" chapter of *Environment Variables & System Options* for more information.

VTXIPC_SO (OpenVMS)

The VTXIPC_SO logical is set to the full path and filename of **vtxipc_so.exe**, a shared image distributed with *xf*ODBC. VTXIPC_SO is set by **CONNECT_STARTUP.COM** and should not be changed.

The following environment variable is used by SQL OpenNet when you use *xf*ODBC in a client/server configuration and is automatically set by the installation.

XFODBCUSR_SO (OpenVMS)

The XFODBCUSR_SO logical is set to the full path and filename of the **xfodbcusr_so.exe** shared image. For example:

```
DEFINE/SYS XFODBCUSR_SO CONNECTDIR:XFODBCUSR_SO.EXE
```

Note the following:

- XFODBCUSR_SO is automatically set. The *xf*ODBC installation sets the XFODBCUSR_SO logical in **CONNECT_STARTUP.COM**, a file that's read when the system is started.
- The filename must be **xfodbcusr_so.exe**, and it must be included in the XFODBCUSR_SO logical setting.

B

SQL Support

This appendix lists the SQL statements, commands, and functions supported by the Synergy database driver (**vtx4**), and documents how they are implemented for *xf*ODBC. Statements, commands, and functions that are not listed are not supported. Note that this is not intended as a general guide to SQL.

Many examples in this section are written for the sample database included in the Connectivity Series distribution. Examples that use the Customers, Orders, Plants, and Vendors tables are written for this database. For information on setting up this database for *xf*ODBC access, see chapter 2, "Using the Sample Database As a Tutorial."

Note that we strongly recommend that you do not use ODBC to update Synergy data. See "Statements that Modify Data" on page B-46 for notes on updating databases.

Conventions, Names, and Identifier Case

Along with the conventions listed in "Manual conventions" on page ix, the following are used in this appendix:

- SQL constructs, such as commands, functions, and operators, are in uppercase—for example, SELECT, GROUP BY, and AND.
- Table names, column names, and other non-SQL constructs are in lowercase—for example, column_id. In your SQL statements, however, capitalization for identifiers must match the capitalization for the entities they refer to.

Note that identifiers are converted to all uppercase characters unless they are enclosed in quotation marks ("") or square brackets ([]). (We recommend using quotation marks rather than brackets.) For example, *table_name* is converted to TABLE_NAME in the following:

```
SELECT * FROM table_name
```

However, for the next statements, both Col_name and Table_name retain the capitalization specified in the statement:

```
SELECT [Col_name] FROM [Table_name]
```

```
SELECT "Col_name" FROM "Table_name"
```

In all cases, however, the case of the identifier must ultimately match the database entity that it refers to (column name, table name, and so forth). So, if you don't use quotation marks or square brackets, the name of the database entity must be in all capital letters, and if you do use quotation marks or square brackets, the identifier's capitalization must match the capitalization for the entity's name. The following will not work, for example, because the table alias is established as "O", (capital letter) while the column name specifies "o" (lowercase):

```
SELECT "o".or_number FROM orders [O]
```

*xf*ODBC supports identifiers (column names, table names, etc.) that are up to 30 characters long. Identifiers must start with an alphabetic character and can include numbers as well as the underscore characters (_). They can also include some special characters, such as the minus (-) and plus (+) signs, but if you use these, you must always put quotation marks ("") around the identifier.

Italicized square brackets indicate that the enclosed keyword or argument is optional—not bracketed to preserve case as described above. For example, *table_alias* (which is the syntax for the *table_list* argument for SELECT) in the following is enclosed in italicized square brackets because it is optional. Don't use square brackets (or quotation marks) in a SELECT statement's table list unless you want to preserve case.

[owner_name.]table_name [table_alias]

Statements that Access Data

*xf*ODBC supports the following, which enable you to access data:

- SELECT statements (see SELECT below)
- Subqueries (see "Creating subqueries and inline views" on page B-8)
- Joins (see "Joins" on page B-10)

SELECT

The SELECT command enables you to create queries (SQL statements that retrieve data from a database). It has the following syntax:

```
SELECT [SKIP n] [TOP n] [ALL|DISTINCT] column_list
   FROM table_list
   [WHERE search_conditions]
   [GROUP BY column_id[, ...]
   [HAVING search_conditions]
   [ORDER BY sort_specification_1 [ASC|DESC][, sort_specification_2 [ASC|DESC], ...]]
   [UNION [ALL] sel_stmt] [...]
```

where the arguments are as follows:

column_list

One or more column specifications. This can include column names, CASE statements, functions or other value expressions (which can be literals or calculations—e.g., or_qty * 3). A column name can be preceded by a table name or by an owner and table name:

[[owner_name .]table_name .]column_name [column_alias]

For information on column aliases, see "Column aliases" on page B-28.

table_list

One or more table or view names. An owner name can precede a table or view name.

[owner_name .]table_name [table_alias]

Note that you can also use inline views (see "Creating subqueries and inline views" on page B-8). For information on table aliases, see "Table aliases" on page B-27.

search_conditions

One or more search criteria for rows.

column_id

A column name, a column name preceded by a table name, or a column name preceded by an owner name and a table name:

[[owner_name .]table_name .]column_name

sort_specification_n

The column name or select list position number that will be used to sort the rows plus an optional ASC for ascending order (the default) or DESC for descending order.

sel_stmt

A SELECT statement whose results will be combined with other SELECT statements connected with the UNION operator.

When you create a SELECT statement, you specify which rows and columns you want the statement to retrieve. You can retrieve a subset of rows from one or more tables, you can retrieve a subset of columns from one or more tables, and you can link rows from two or more tables. SELECT statements can contain

- the FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY, and FOR UPDATE OF clauses. See "Notes on Clauses, Columns, and Aliases" on page B-15 for information.
- the UNION operator. See "Notes on Clauses, Columns, and Aliases" on page B-15 for information.
- aggregate functions, which are functions that return a single value from a group of values (for example, SUM). See "Aggregate Functions" on page B-29 for information.
- scalar functions, which are functions that return a value for each returned row. See "Scalar Functions" on page B-32 for information.
- bitwise functions. See "Bitwise Functions" on page B-45 for information.
- expressions, which can be used in the SELECT statement's column list or in a WHERE clause.
- joins, including inner, outer, and full joins. See "Joins" on page B-10 for information.
- subqueries (including inline views). You can use SELECT statements within SELECT statements. See "Creating subqueries and inline views" on page B-8 for information.

You can also use

- the asterisk (*) wildcard in place of a column list. To select all of the columns in the tables specified in *table_list*, use an asterisk (*). To select all of the columns from one table in *table_list*, use the following:

  *table_name . **

- double quotation marks around identifiers. These instruct the driver to be case-sensitive when evaluating an identifier.

SELECT statements can be used in definitions of views and as subqueries. You can also use them to create derived tables (inline views) by including them in FROM clauses that are part of SQL92 outer joins. See "Creating subqueries and inline views" on page B-8 and "FROM clauses in SQL92 outer joins" on page B-14.

Note the following:

- To use a SELECT statement, you must have access privileges to all tables accessed by the statement.
- The SELECT keyword can be preceded by spaces, tab, carriage return, and line feed, but not by any other character unless the SELECT statement is a subquery. See "Creating subqueries and inline views" on page B-8.
- SELECT statement clauses are evaluated in the following order: FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY.
- Any select list item consisting of an expression, aggregate function, or scalar function that is not a single column in the database is assigned a column name as 'EXPR*n*', where *n* is the ordinal position of the select list item. You can use an alias to override this assigned column name.
- You must qualify a column when multiple tables are selected and the same column name is used in more than one table. For example:

  ```
  SELECT table1.col_1, table2.col_1 FROM table1, table2
  ```

- If a column name is a reserved word, you must enclose the name in double quotes. For example:

  ```
  SELECT "date", "order" FROM orders
  ```

  See "ODBC Reserved Words" on page B-65 for a list of reserved words.
- If the SELECT list (*column_list*) includes a mix of aggregate functions and value functions or column names, the value functions and column names must be specified in a GROUP BY clause.
- We recommend always using an ORDER BY clause. Without it, the order of result sets may change when you update to a new version of Connectivity Series or even apply a Connectivity Series patch.

The following select one, several, or all columns from a table:

```
SELECT col_1 FROM my_table

SELECT col_1, col_2, col_3 FROM my_table

SELECT * FROM my_table

SELECT table1.* FROM table1

SELECT owner_name.table1.* FROM table1
```

The following are similar examples with double quotes around some identifiers:

```
SELECT "COL_1" FROM "MY_TABLE"

SELECT "COL_1", col_2 FROM my_table

SELECT "OWNER_NAME".table1.* FROM table1

SELECT "OWNER_NAME"."TABLE1".* FROM "TABLE1"
```

The following select one, several, or all columns from two tables:

```
SELECT col, col_1 FROM table1, table2

SELECT table1.col_1, table2.col_1 FROM table1, table2

SELECT owner_name.table1.col_1, owner_name.table2.col_1
    FROM table1, table2

SELECT * FROM table1, table2
```

SELECT ALL

SELECT ALL returns every row that meets a query's criteria, even if some rows are duplicates. This is the default for SELECT statements, so you don't need to include "ALL". For example, the following statements produce identical results:

```
SELECT ALL or_vendor, or_edate FROM orders
SELECT or_vendor, or_edate FROM orders
```

SELECT DISTINCT

SELECT DISTINCT returns only one copy of duplicate rows. For example, if the orders table has two or more records whose or_vendor and or_edate columns are identical, the following returns only one row for those records.

```
SELECT DISTINCT or_vendor, or_edate FROM orders
```

SKIP

SKIP is a sub-clause that specifies how many rows to trim from the beginning of the result set for a query. SKIP can be used with TOP for paging, which is particularly useful for creating cached result sets for websites that use ADO.NET.

A SKIP clause must immediately follow the SELECT keyword. The syntax is

`SKIP` *n*

where *n* is a numeric expression. For example, the following returns all but the four tallest plants in the sample database:

```
SELECT SKIP 4 DISTINCT in_name
    FROM plants
    ORDER BY in_maxhigh DESC
```

Note the following:

- SKIP and TOP can be used only once in a query. For example, you cannot use SKIP or TOP in the main query and then use it again in a subquery.
- If the result set (before SKIP is applied) is equal to or less than *n*, no rows will be returned.
- If a query has both SKIP and TOP, SKIP is used first to trim the result set, and TOP is applied if any rows remain.
- SKIP and TOP are evaluated after GROUP BY and ORDER BY clauses.

TOP

TOP is a sub-clause that specifies the number of rows to be returned. The syntax is

`TOP` *n*

where *n* is a numeric expression. For example, the following returns the names of the four tallest plants in the sample database:

```
SELECT TOP 4 DISTINCT in_name
    FROM plants
    ORDER BY in_maxhigh DESC
```

If the result set (before TOP is applied) is equal to or less than *n*, all rows are returned. See SKIP on page B-7 for more information.

Creating subqueries and inline views

A subquery (i.e., a nested query) is a SELECT statement embedded within a SQL statement and enclosed in parentheses (except in INSERT statements). Like joins, subqueries enable you to query multiple tables, though there are other uses for subqueries. There are two basic types:

- A *scalar subquery* returns no more than one column from one row and can be used anywhere that a scalar value can be used in a WHERE, WHEN (for CASE), or FROM clause. For example, the subquery in the following returns order numbers for orders where the order price is greater than the average price for plants in the sample database:

  ```
  SELECT or_number FROM orders
      WHERE or_price > (SELECT AVG(in_price) FROM plants)
  ```

- A *table subquery* can return multiple rows and columns and can be used anywhere a table or view can be used in a FROM clause. A table subquery in a FROM clause is known as an *inline view*. A table subquery can also be used in a WHERE or WHEN clause, in certain conditions (discussed below), and it can be used in an INSERT statement (see INSERT on page B-47) and in a SET clause for UPDATE (see UPDATE on page B-48). For example, the following uses an inline view to create a combined list of zip codes for customers and vendors:

  ```
  SELECT cust_zip AS all_zips, cust_city AS all_city FROM (
      SELECT cust_zip, cust_city FROM customers UNION ALL
      SELECT vend_zip, vend_city FROM vendors)
  ```

Note the following:

- *xf*ODBC doesn't support SELECT ROW, so row subqueries (a type of scalar subquery that returns only one row) are not supported.
- ORDER BY clauses and aggregates in the select list are supported for inline views, but not for other subqueries.
- A table subquery can be used in a WHERE or WHEN clause only if it's preceded by IN or [NOT] EXISTS or a comparison operator used with SOME, ALL, or ANY. For example, the following statement uses a table subquery with IN to retrieve the name of every plant currently on order:

  ```
  SELECT plants.in_name FROM plants
      WHERE plants.in_itemid IN (SELECT or_item FROM orders)
  ```

  For an example that includes GROUP BY clauses in inline views that are part of a UNION clause, see GROUP BY on page B-22.

Example subqueries

The following retrieves the customers who currently have orders in the order table:

```
SELECT customers.cust_name FROM customers
    WHERE customers.cust_key IN (SELECT or_customer FROM orders)
```

The next example, however, does *not* work because it has a subquery in the select list (which isn't supported):

```
SELECT customer_id, customer_state, (SELECT SUM(quantity) FROM orders
    WHERE orders.customer_number=customers.customer_id)
        AS orders FROM customers
```

The next query is similar to the preceding query (although it returns rows with null in the second column), but in this case the subquery is in the FROM clause, which creates an inline view, so it is valid.

```
SELECT o.cust_key, o.cust_state, s.c
    FROM customers o, (SELECT or_customer, SUM(or_qty) c FROM orders
        GROUP BY or_customer) s
    WHERE o.cust_key = s.or_customer
```

The following example shows another use for subqueries. For this statement, the table subquery isn't used to query multiple tables, but it is necessary because aggregate functions (e.g., AVG) can't be used in WHEN clauses. (They can't be used in WHERE or ORDER BY clauses either.)

```
SELECT in_maxhigh,
    CASE
        WHEN in_maxhigh > (SELECT AVG(in_maxhigh) FROM plants)
            THEN 'Tall'
        END
    FROM plants
```

Joins

Joins are a way of returning records from two or more tables—in most cases, records that in some way match. For example, if you have a plant table and an order table, each with a field that stores vendor IDs, you can use a join to return records that have matching vendors. There are three types of join: inner, outer, and full.

Inner joins return only matching records. If a record in any table in the join doesn't have a match in the other tables, the record is ignored. See "SQL89 inner joins" and "SQL92 inner joins" on page B-11 for examples. *xf*ODBC supports SQL89 inner joins through the WHERE command and SQL92 inner joins through the INNER JOIN command.

Outer joins return all records from one table but only records with matches from the other table in the join. SQL89 syntax doesn't support outer joins (though some databases have extensions to SQL89 syntax that enable you to create these), but SQL92 does support these through the LEFT [OUTER] JOIN and RIGHT [OUTER] JOIN commands.

With SQL92 syntax, you determine which table will supply a full set of records by where you specify the table in a left outer join or a right outer join:

- Left outer joins return all of the records in the first table of the join and only matching records from the second table. See "SQL92 left outer joins" on page B-12 for examples.
- Right outer joins return all records in the second table specified in the join and only matching records from the first table. See "SQL92 right outer joins" on page B-13 for examples.

You can also create *full outer joins*, which return all records from both tables in the join. SQL89 syntax doesn't include any special keywords for this, but you can get similar results by using the UNION operator. SQL92 syntax, however, includes the FULL [OUTER] JOIN keyword.

`FULL` *[*`OUTER`*]* `JOIN` *table_name* `ON` *column1* `=` *column2*

where *column1* is a column in *table_name* and *column2* is a column in another FROM clause table. See "SQL92 full joins" on page B-13 for an example.

Note the following for joins:

- Don't nest joins. Nesting joins may reduce performance.
- We recommend using SQL92 syntax rather than SQL89 syntax, and we don't recommend combining the two forms (i.e., using both FROM and WHERE clauses to define the join and restriction criteria). This will generally reduce performance. See "Avoid mixing SQL92 and SQL89 syntax" on page 10-11.

- The join parser doesn't require SQL92 joins to be enclosed in the ODBC escape sequence. For example, both of the following are acceptable:

  ```
  {oj orders RIGHT JOIN vendors ON or_vendor = vend_key}
  ```

  ```
  orders RIGHT OUTER JOIN vendors ON or_vendor = vend_key
  ```

- If any field specified in a join is not part of an index segment, the *xf*ODBC driver may create a temporary index that includes all of the segments in the join. See "How xfODBC uses keys" on page 10-2 for more information.

For information on optimizing join performance, see "Creating Efficient SQL Statements" on page 10-9.

SQL89 inner joins

*xf*ODBC supports SQL89 inner joins through the WHERE command. Use a WHERE clause to specify which columns should match. The following example is a SQL89 inner join. It selects only those records from the m1 and m2 tables whose id and num columns match.

```
SELECT
    m1.id,
    m1.num,
    m1.alpha,
    m2.id,
    m2.num,
    m2.alpha
FROM
    multiop  m1,
    multiop2 m2
WHERE
    m1.id = 5
    AND  m1.num = 50
    AND  m1.id = m2.id
    AND  m1.num = m2.num
```

SQL92 inner joins

*xf*ODBC supports SQL92 inner joins through INNER JOIN clauses:

`INNER JOIN` *table_name* `ON` *column1* `=` *column2*

where *column1* is a column in *table_name* and *column2* is a column in another FROM clause table.

INNER JOIN returns only records that match records in the other table of the join. For example, the following query returns only those records from the m1 and m2 tables whose id and num columns match:

```
SELECT
    m1.id,
    m1.num,
    m1.alpha,
    m2.id,
    m2.num,
    m2.alpha
FROM
    multiop m1
      INNER JOIN multiop2 m2
        ON    m1.id = m2.id
        AND   m1.num = m2.num
```

Here's the same FROM clause using the ODBC escape sequence form:

```
FROM
    multiop m1
      {oj INNER JOIN multiop2 m2
        ON   m1.id = m2.id
        AND  m1.num = m2.num}
```

SQL92 left outer joins

*xf*ODBC supports SQL92 left outer joins through LEFT OUTER JOIN clauses:

`LEFT` *[*`OUTER`*]* `JOIN` *table_name* `ON` *column1* = *column2*

where *column1* is a column in *table_name* and *column2* is a column in another FROM clause table.

LEFT *[*OUTER*]* JOIN returns all of the records in the first table (the table that precedes “LEFT OUTER JOIN”) and matching records from the table on the right (the table that follows “LEFT OUTER JOIN”). For example, the following query returns the specified fields for all records in the vendors table and any records from the orders table that match the ON criteria.

```
SELECT
    orders.or_item,
    orders.or_number,
    vendors.vend_name
FROM
    {OJ vendors
      LEFT OUTER JOIN orders
        ON vendors.vend_key = orders.or_vendor}
```

The following example returns all orders in the orders table and each corresponding plant name from the plant table:

```
SELECT
    orders.or_item,
    orders.or_number,
    plants.in_name
FROM
    {OJ orders
      LEFT JOIN plants
        ON orders.or_item = plants.in_itemid}
```

SQL92 right outer joins

*xf*ODBC supports SQL92 right outer joins through RIGHT OUTER JOIN clauses:

`RIGHT` *[*`OUTER`*]* `JOIN` *table_name* `ON` *column1* = *column2*

where *column1* is a column in *table_name* and *column2* is a column in another FROM clause table.

RIGHT *[*OUTER*]* JOIN returns all of the records in the second table (the table that follows "RIGHT OUTER JOIN") and matching records from the first table (the table that precedes "RIGHT OUTER JOIN"). For example, the following query returns the specified fields for all records in the vendors table and any records in the orders table that match the ON criteria.

```
SELECT
    orders.or_item,
    orders.or_number,
    vendors.vend_name
FROM
    {OJ orders
      RIGHT JOIN vendors
        ON vendors.vend_key =  orders.or_vendor}
```

SQL92 full joins

*xf*ODBC supports SQL92 full joins through FULL OUTER JOIN clauses:

`FULL` *[*`OUTER`*]* `JOIN` *table_name* `ON` *column1* = *column2*

where *column1* is a column in *table_name* and *column2* is a column in another FROM clause table.

FULL *[*OUTER*]* JOIN returns all records from both the left and right table, whether or not there are matching values. For example, the following query returns the specified fields for all records in the vendors and orders tables:

```
SELECT
    o.or_item, o.or_number, v.vend_name
FROM
    orders o
      FULL JOIN vendors v
        ON o.or_vendor = v.vend_key
```

ON clauses in SQL92 outer joins

Each ON clause for a SQL92 outer join should be placed immediately after the table qualifier it modifies. If this is not the case, as in the following example, you will get an error.

```
SELECT
    o.or_terms,
    o.or_odate,
    o.or_qty,
    p.in_name,
    v.vend_name
FROM
    { oj orders o
      LEFT JOIN vendors v
      LEFT JOIN plants p
      ON o.or_vendor = v.vend_key
      ON o.or_item = p.in_itemid}
```

The next example is correctly constructed. Each ON clause follows its qualifier.

```
SELECT
    o.or_terms,
    o.or_odate,
    o.or_qty,
    p.in_name,
    v.vend_name
FROM
    { oj orders o
      LEFT JOIN vendors v
        ON o.or_vendor = v.vend_key
      LEFT JOIN plants p
        ON o.or_item = p.in_itemid}
```

FROM clauses in SQL92 outer joins

FROM clauses in SQL92 outer joins can include SELECT statements, which can greatly optimize performance. If you include a SELECT statement in a FROM clause, you must enclose the entire SELECT statement in parentheses (), and you must include a table alias for the derived table. For example:

```
SELECT
    o.or_number,
    p.in_itemid,
    p.in_name
FROM
    (SELECT * FROM orders WHERE orders.or_item < 7) o
      INNER JOIN plants p ON o.or_item = p.in_itemid
```

Notes on Clauses, Columns, and Aliases

This section discusses the *xf*ODBC driver's support for the following:

- WHERE on page B-16
- ORDER BY on page B-21
- GROUP BY on page B-22
- HAVING on page B-23
- FROM on page B-23
- FOR UPDATE OF on page B-24
- CASE on page B-24
- UNION on page B-26
- Computed columns on page B-27
- Text columns on page B-27
- Table aliases on page B-27
- Column aliases on page B-28

For information on optimizing clauses, see "Creating Efficient SQL Statements" on page 10-9.

For information on SKIP and TOP sub-clauses, see SKIP on page B-7 and TOP on page B-8.

WHERE

The WHERE clause enables you to specify selection criteria for an SQL command. You can use the following operators with the WHERE clause:

WHERE Clause Operators

Name	Operator	Examples
Equal to	=	`SELECT or_number FROM orders WHERE or_customer = 8`
	IS NULL	`SELECT * FROM orders WHERE or_sdate IS NULL`
Greater than	>	`SELECT or_number FROM orders WHERE or_customer > 8`
Greater than or equal to	>=	`SELECT or_number FROM orders WHERE or_customer >= 8`
Less than	<	`SELECT or_number FROM orders WHERE or_customer < 8`
Less than or equal to	<=	`SELECT or_number FROM orders WHERE or_customer <= 8`
Not equal to	<>	`SELECT * FROM plants WHERE in_shape <> 'tree'`
	IS NOT NULL	`SELECT * FROM orders WHERE or_sdate IS NOT NULL`

For more complex selection criteria, combine multiple WHERE clauses with AND or OR connectors. For example:

```
SELECT or_number, or_customer FROM orders WHERE or_customer = 8
    AND or_odate = '1993-03-07'

SELECT or_number, or_customer FROM orders
    WHERE (or_customer = 16 AND or_odate = '1995-03-03')
    OR (or_customer = 8 AND or_odate = '1993-03-07')
```

Note the following:

- A single quote (apostrophe) in column data must be preceded by another single quote. For example, the following is incorrect:

  ```
  SELECT cust_gift FROM customers
      WHERE cust_name = 'Troy's Trees'
  ```

  The next query, however, correctly handles this by adding a second single quote:

  ```
  SELECT cust_gift FROM customers
      WHERE cust_name = 'Troy''s Trees'
  ```

- You can use OR to connect up to 127 conditions:

  ```
  WHERE condition_1 OR condition_2 ... OR condition_127
  ```

- When comparing date and time columns, make sure you use the correct format. See "Formats for returned dates and times" on page 8-13 and "Masks for dates and times in SQL statements" on page 8-15.

Clauses in WHERE clauses

You can also use the following clauses, some of which use subqueries. Note that in the following *exp* is short for "expression," *compare_op* is short for "comparison operator," and *char_exp*, is short for "character expression."

ALL

ALL returns true if all values returned by the subquery cause the full clause to be true. Otherwise, it returns false. ALL has the following syntax:

exp compare_op `ALL` *subquery*

where *exp* is an expression, *compare_op* is a WHERE clause operator, and *subquery* is a subquery.

For example, the following returns true if every value produced by the subquery ("SELECT a FROM b") equals 'able':

```
...WHERE 'able' = ALL (SELECT a FROM b)
```

ANY

ANY returns true if any value returned by the subquery causes the full clause to be true. Otherwise, it returns false. ANY has the following syntax:

exp compare_op `ANY` *subquery*

where *exp* is an expression, *compare_op* is a WHERE clause operator, and *subquery* is a subquery.

For example, the following returns true if one of the values produced by the subquery (“SELECT a FROM b”) equals ‘able’:

```
...WHERE 'able' = ANY (SELECT a FROM b)
```

ANY is identical to SOME.

BETWEEN

BETWEEN returns true if a given value is in a given range. BETWEEN has the following syntax:

value_a [`NOT`*]* `BETWEEN` *value_b* `AND` *value_c*

where *value_a*, *value_b* and *value_c* are value expressions.

BETWEEN returns true if *value_a* >= *value_b* and *value_a* <= *value_c*. Otherwise it returns false.

For example, the following return true:

```
2 BETWEEN 1 AND 10
```

```
10 BETWEEN 1 AND 10
```

```
'c' BETWEEN 'a' AND 'm'
```

The following return false:

```
11 BETWEEN 1 AND 10
```

```
2 NOT BETWEEN 1 AND 10
```

```
'c' NOT BETWEEN 'a' AND 'm'
```

EXISTS

EXISTS returns true if the subquery produces rows. Otherwise it returns false. EXISTS has the following syntax:

[`NOT`*]* `EXISTS` *subquery*

For example, the following returns all rows specified by the main query (“SELECT * FROM staff”) if there is a deptno value that equals 10. If no deptno value equals 10, nothing is returned.

```
SELECT * FROM staff WHERE EXISTS
    (SELECT deptno FROM org WHERE deptno = 10)
```

IN

IN returns true if *search_value* is in *value_set*. Otherwise it returns false. IN has the following syntax:

search_value [`NOT`*]* `IN` *(value_set)*

were *value_set* can have up to 127 values separated by commas.

For example, the following return true:

```
2 IN (1,2,3)
```

```
'3' IN ('1', '2', '3')
```

The following return false:

```
4 IN (1,2,3)
```

```
2 NOT IN (1,2,3)
```

```
'3' NOT IN ('1', '2', '3')
```

LIKE

LIKE searches for a string (*search_string*) in a character string expression (*char_exp*). It has the following syntax:

char_exp [`NOT`*]* `LIKE` *search_string [*`{ESCAPE '`*c*`'}`*]*

Search_string can include the following:

%	A wildcard character that represents a string of zero or more characters. Note that null values are not strings, so the following returns rows whose *item_desc* is non-null: `WHERE item_desc LIKE '%'`
_ (underscore)	Any single character
\	The escape character for %, _, or \
{ESCAPE '*c*'}	A definition for an escape character, where *c* is the escape character that can be used in *search_string*. Note that the braces are optional. For example: `char_exp LIKE string ESCAPE '-'`

Note that escape characters in LIKE clauses must either be escaped by another escape character, or they must precede an escapable character. Otherwise the escape character is discarded in the search, causing the query results to be incorrect. (Most likely, no rows will be returned.) Note that no error is generated in this situation.

For example, both of the following clauses are incorrect. For the first example, the driver will discard the backslashes (\) and look for 1231995. For the second example, the driver will discard the hyphen (-) and look for 123ABC.

```
...WHERE order_date LIKE '12\3\1995'

...WHERE account_no LIKE '123-ABC' {ESCAPE '-'}
```

The following, however, are correct in this respect:

```
...WHERE order_date LIKE '12\\3\\1995'

...WHERE order_date LIKE '^%9_' {ESCAPE '^'}
```

The following example uses a WHERE clause operator with the string wildcard (%):

```
SELECT cust_key, cust_name FROM customers
    WHERE cust_name LIKE 'P%'
```

The following uses the backslash (\) escape character to prevent the percent sign (%) from being interpreted as an escape character:

```
SELECT item_no FROM history
    WHERE item_desc
    LIKE 'LESS 15\% DISCOUNT'
```

The following statement defines ^ as an escape character and uses this to prevent the percent sign (%) from being interpreted as an escape character:

```
SELECT item_no FROM history
    WHERE item_desc
    LIKE 'LESS 15^% discount'
      ESCAPE '^'
```

SOME

Identical to ANY. See ANY on page B-17.

Nulls

If an expression returns a null value for a row, *xf*ODBC will return that row. For example, the following returns all plant information where the color is not white, including rows where in_color is null:

```
SELECT * FROM plants WHERE in_color <> 'white'
```

Note that if >, >=, <=, or < is used to compare “null” to a literal, no rows will be returned.

For information on how *xf*ODBC interprets null values, see “Zeros, spaces, and null values” on page 3-17.

ORDER BY

The ORDER BY clause sorts the result set. Rows are sorted according to the columns listed in the ORDER BY clause: the first column listed is the primary sort criterion, the second column determines the order within duplicate values in the first, etc. You can specify ascending or descending order by including ASC or DESC. Ascending is the default. Here are some examples:

```
SELECT * FROM table ORDER BY col_1

SELECT * FROM table ORDER BY col_1 ASC

SELECT * FROM table ORDER BY col_1 DESC

SELECT * FROM table ORDER BY col_1, col_2

SELECT * FROM table ORDER BY col_1 ASC, col_2 DESC, col_3
```

Note the following:

- You can use integer column positions in an ORDER BY clause (1 for the first item in the column list, 2 for the second, and so forth). For example:

  ```
  SELECT or_customer, SUM(or_customer) FROM orders GROUP BY or_customer
      ORDER BY 1
  ```

- For compound queries (queries containing UNION or UNION ALL), ORDER BY clauses must use positions, rather than explicit expressions, and can appear only in the last query (though an ORDER BY will order all rows in the result set). For example:

  ```
  SELECT cust_name FROM customers UNION ALL
  SELECT vend_name FROM vendors ORDER BY 1
  ```

- See “Column aliases” on page B-28 for information on restrictions to aliases that can affect ORDER BY clauses.

GROUP BY

The GROUP BY clause enables you to collate rows with identical column values (which may be aggregated values) and return them as a single row. For example, the following statement returns five rows from the sample database even though there are eight rows of data in the table. (Four rows have the same or_price value, so these rows are combined into one row.)

```
SELECT or_price, SUM(or_qty) FROM orders GROUP BY or_price
    ORDER BY 2
```

The next example shows how inline views can include GROUP BY clauses:

```
SELECT t1.c1, t1.c2, t1.c3
  FROM (
    SELECT 1 AS c3, COUNT(cust_key) AS c1, cust_state AS c2
      FROM customers il1 GROUP BY cust_state
      ) t1
  UNION
  SELECT t2.c1, t2.c2, t2.c3
    FROM (
      SELECT 1 AS c3, COUNT(vend_key) AS c1, vend_state AS c2
        FROM vendors il2 GROUP BY vend_state) t2
```

The next example returns an average for in_price for each grouping created by the GROUP_BY clause. So when this is run against the sample database for Connectivity Series, it returns two averages, one for each in_type value (1 and 2).

```
SELECT in_type, AVG(in_price) FROM plants GROUP BY in_type
```

GROUP BY is also used to apply aggregate functions to groups of rows. For example, the result set for the following includes a row for each customer, a column for the customer's number, and a column with the sum of or_qty values for the customer.

```
SELECT or_customer, SUM(or_qty) FROM orders GROUP BY or_customer
```

(If the column lists contains only aggregate functions, rows in the result set are treated as a single group for the aggregate functions. For example, "SELECT SUM(or_qty) FROM orders" returns the sum of all or_qty values in each row of the result set.)

Note the following:

- *xf*ODBC supports up to eight GROUP BY columns.
- GROUP BY is not supported for UNION clauses, and it is not supported for SELECT * statements. For example, the following will cause an error ("Non aggregates require a GROUP BY expression"):

  ```
  SELECT * FROM orders GROUP BY or_odate
  ```

  Additionally, note that a subquery can't have a GROUP BY clause if an outer query uses *.

- All columns in the select list must be in the GROUP BY clause if there is one. (Columns derived from aggregate calculations are the exception. You don't need to include these in the GROUP BY clause. See the examples above.) Statements that don't adhere to this will cause "Column(#) is out of range" errors (where # is a meaningless number).
- If the column list includes an aggregate function, all non-aggregate items in the list must be specified in a GROUP BY clause. If all items in a statement's column list are aggregate functions, rows in the result set are treated as a single group for the aggregate functions.
- Note that a GROUP BY clause must contain an actual column name or number, not an expression. For example, the following will cause a syntax error:

  ```
  SELECT in_name FROM plants GROUP BY LCASE(in_name)
  ```

  To work around this, put the expression in a subquery that creates an alias for the expression, and then use the alias with GROUP BY:

  ```
  SELECT "Common Name"
      FROM (SELECT LCASE(in_name) "Common Name" FROM plants)
      GROUP BY "Common Name"
  ```

- See "Column aliases" on page B-28 for information on restrictions to aliases that can affect GROUP BY clauses.

HAVING

The HAVING clause enables you to place limitations on groups returned by a GROUP BY clause. (This is particularly useful for criteria that include an aggregate function because a WHERE clause cannot contain an aggregate function.) For example:

```
SELECT or_customer, AVG(or_qty) FROM orders
    GROUP BY or_customer HAVING AVG(or_qty) > 65
```

FROM

The FROM clause enables you to specify which tables a query will retrieve data from. You can specify views, base tables, and tables that result from operations that create tables: queries, subqueries (inline views), and so forth. For example, the following example selects all columns from two tables:

```
SELECT * FROM table1, table2
```

Note the following:

- You can optimize performance for SQL92 outer joins (full outer, left outer, and right outer) by using inline views, which are SELECT statements in FROM clauses.
- If you use a SELECT statement in a FROM clause (i.e., an inline view), you must enclose the entire SELECT statement in parentheses ().
- You can use table aliases to simplify table references. See "Table aliases" on page B-27.

FOR UPDATE OF

The FOR UPDATE OF clause enables you to select rows that match a statement's selection criteria. If the clause is part of a transaction, it locks selected rows. It has the following syntax:

sel_statement where_clause `FOR UPDATE OF` *[column_list]*

where *sel_statment* is a SELECT statement, *where_clause* is a WHERE clause, and *column_list* is an optional list of columns that will be updated. (Note that *column_list* is vestigial for *xf*ODBC and has no effect on the statement.) For example, the following statement locks rows for which in_itemid equals 20:

```
SELECT in_itemid, in_zone FROM plants WHERE in_itemid = 20
    FOR UPDATE OF
```

Note that a FOR UPDATE clause can only be at the end (the last clause) of a SELECT statement.

CASE

CASE evaluates a list of conditions and returns the result for the condition that is true. If no condition is true, the result specified in the ELSE clause is returned or, if there is no ELSE clause, null is returned. It has the following syntax:

```
CASE
    WHEN boolean_1 THEN result_1
    [WHEN boolean_2 THEN result_2
    . . .
    WHEN boolean_n THEN result_n]
    [ELSE else_result]
END
```

or

```
CASE case_exp
    WHEN value_1 THEN result_1
    [WHEN value_2 THEN result_2
    . . .
    WHEN value_n THEN result_n]
    [ELSE else_result]
END
```

where arguments are as follows:

*boolean_**

Boolean expressions. These can be constructed with any operator, clause, etc., that can be used in a WHERE clause (subqueries, IN clauses, BETWEEN clauses, and so forth). See WHERE on page B-16.

*result_**

Expressions whose results are the possible return values for the CASE statement.

case_exp

An expression that's compared to *value_** arguments to determine which result to return.

*value_**

Values that are compared for equality to *case_exp* to determine which result to return.

else_result

An expression that is returned if no *boolean_** argument is true or if no *value_** argument matches *case_exp*.

Note that for the second syntax form, the comparison is always a test for equality—i.e., if *case_exp* = *value_2*, CASE returns *result_2*.

The following example uses the first syntax form:

```
SELECT in_itemid, in_name,
    CASE
        WHEN in_color IS NULL THEN 'No color'
        WHEN CONCAT(in_color, in_shape) = 'blue vine' THEN 'Blue vine'
        WHEN in_size > 10 THEN 'Large'
        WHEN in_size BETWEEN 5 AND 10 THEN 'Medium'
        WHEN in_size IN (1,2,3,4,5) THEN 'Small'
        ELSE '0'
    END AS mycol
    FROM plants
```

The next example uses the second syntax form:

```
SELECT in_itemid, in_name,
     CASE in_size
       WHEN 10 THEN 'Large'
       WHEN 5  THEN 'Medium'
       WHEN 1  THEN 'Small'
       ELSE 'Other'
     END AS mycol
     FROM plants
```

The next example uses a subquery. The subquery is necessary because aggregate functions (AVG in this case) can't be used in WHEN clauses. (They can't be used in WHERE or ORDER BY clauses either.)

```
SELECT in_maxhigh,
  CASE
    WHEN in_maxhigh > (SELECT AVG(in_maxhigh) FROM plants)
      THEN 'Tall'
    END
  FROM plants
```

Note the following:

- All results, including *else_result* must have the same data type.
- Conditions are evaluated in order. *Boolean_1* is evaluated before *boolean_2*, *value_1* is evaluated before *value_2*, and so forth.

UNION

The UNION operator combines the results of multiple SELECT statements into one result set. It has the following syntax:

sel_1 `UNION` [`ALL`] *sel_2* [...`UNION` [`ALL`] *sel_n*]

where *sel_1* through *sel_n* are SELECT statements.

If you include ALL, duplicate rows are included in the result set. If you omit ALL, duplicate rows are omitted and results are sorted by the first column.

The following returns the cities for all customers and all vendors in alphabetical order. Note, however, that if there's a duplicate city, only one occurrence of the city will be returned:

```
SELECT cust_city FROM customers UNION
SELECT vend_city FROM vendors
```

If you add ALL to the same query, all customer and vendor cities will be returned, even duplicates.

```
SELECT cust_city FROM customers UNION ALL
SELECT vend_city FROM vendors
```

Note the following:

- Columns for all queries must be identical.
- There must be the same number of columns for each query.
- The data types must be compatible.
- Only one ORDER BY clause is allowed with a UNION, and this must follow the final SELECT statement. An ORDER BY clause in a UNION applies to the entire result set.
- GROUP BY clauses are not supported for UNION clauses.
- Using UNION without ALL is more efficient than using SELECT DISTINCT with UNION ALL.

See "Creating subqueries and inline views" on page B-8 for an example of a UNION clause in an inline view.

Computed columns

An expression can be used within a SELECT statement's column list. For example:

```
SELECT or_price + or_price * .1
    FROM orders
```

```
SELECT or_price, '*', or_qty, '=', or_price * or_qty
    FROM orders
```

```
SELECT or_price + TO_NUMBER(or_item)
    FROM orders
```

Text columns

One or more text columns (text strings enclosed in single quotes) can be added to a SELECT statement's list. For example:

```
SELECT or_price, '*', or_qty, '=', or_price * or_qty
    FROM orders
```

Table aliases

To make table references simpler, you can assign an alias to a table. (This is also know as a "correlation name" or a "range variable.") Table aliases last for the duration of a statement.

To create a table alias, add the alias after the table name in a SELECT statement:

`SELECT` *column_list* `FROM` *table_name* [`AS`] *alias*

You can enclose the alias in double quotes ("") or square brackets ([]) if you want to protect the alias from change—for example, if you want to preserve the case of characters in the alias. If the alias has a space, you must use double quotes or square brackets.

In the following example, cust is the alias for the customer table, and ord is the alias for the orders table.

```
  SELECT cust.cust_key, ord.or_number
    FROM public.customers cust, public.orders ord
    WHERE cust.cust_key = ord.or_customer
```

This could also be written with AS:

```
SELECT cust.cust_key, ord.or_number
  FROM public.customers AS cust, public.orders AS ord
  WHERE cust.cust_key = ord.or_customer
```

Note the following:

- Table aliases are optional for base tables and views, but are required for tables produced by subqueries.
- Table aliases last only for the duration of the statement's processing.
- Table aliases defined in inline views can be used only in the inline view.

Column aliases

To make column references simpler, you can assign an alias (correlation name) to a column. Column aliases last for the duration of a statement and are used as the column headings in the result set.

To create a column alias, add the alias after the column name. You can enclose the alias in double quotes ("") or square brackets ([]) if you want to protect the alias from change (for example, if you want to preserve the case of characters in the alias). If the alias has a space, you must use double quotes or square brackets.

For example:

```
SELECT customers.cust_zip zipcode, customers.cust_tcode taxcode
  FROM customers

SELECT customers.cust_zip AS zipcode, customers.cust_tcode
  AS taxcode
    FROM customers

SELECT customers.cust_zip "zipcode", customers.cust_tcode
  AS "taxcode"
    FROM customers

SELECT customers.cust_zip [zipcode], customers.cust_tcode [taxcode]
    FROM customers
```

Note the following:

- Do not enclose an alias in single quotes.
- If an alias is identical to a column name that is specified in a scalar function, including that alias in a GROUP BY or ORDER BY clause may cause the result set to be sorted in an unexpected way. (The alias or a literal used in the function, rather than the column, may be used as the ORDER BY or GROUP BY criterion.)
- Column aliases from inline views can be used only in the outermost SELECT statement. (They can't be used in the inline view.)

Aggregate Functions

An aggregate function is a function that derives a single value from a set of values from a column. Aggregate functions must be used with SELECT or HAVING clauses. For example:

```
SELECT in_color, COUNT(in_color) FROM plants GROUP BY in_color
```

```
SELECT or_customer, AVG(or_qty) FROM orders
    GROUP BY or_customer HAVING AVG(or_qty) > 65
```

The following, however, cause errors because an aggregate function cannot be in a GROUP BY clause (unless it's contained in a HAVING clause), an ORDER BY clause, or a WHERE clause:

```
SELECT or_customer, AVG(or_qty) FROM orders
    GROUP BY AVG(or_qty)
```

```
SELECT or_customer, AVG(or_qty) FROM orders
    GROUP BY or_customer ORDER BY AVG(or_qty)
```

```
SELECT or_customer, AVG(or_qty) FROM orders
    WHERE AVG(or_qty) > 65 GROUP BY or_customer
```

If a function is listed in the SELECT statement's column list, you can reference it by its ordinal position in the column list or in a WHERE or ORDER BY clause. (Note that the SQL syntax parser does not currently allow aggregate functions, other than select list functions, to be referenced in ORDER BY clauses.)

Also note that aggregate functions cannot be used with individual columns in a SELECT statement's column list unless accompanied by a GROUP BY command that groups by individual column. For example:

```
SELECT or_customer, SUM(or_qty) FROM orders GROUP BY or_customer
```

```
SELECT or_customer, COUNT(or_customer), SUM(or_qty) FROM orders
    GROUP BY or_customer
```

See GROUP BY on page B-22 for information about GROUP BY.

AVG

This function returns the average of the values in the specified column. It has the following syntax:

AVG(*col*)

For example, the following returns the average or_qty value for rows in the orders table:

```
SELECT AVG(or_qty) FROM orders
```

The next example returns the or_qty averages for each customer (or_customer):

```
SELECT or_customer, AVG(or_qty) FROM orders GROUP BY or_customer
```

COUNT

This function returns the number of rows that don't have null values in the specified column. It has the following syntax:

```
COUNT(col)
```

COUNT(*) returns a count of all of the rows in a table that meet WHERE clause criteria (or a count of all rows if there is no WHERE clause).

The following statement returns a count of rows whose in_size value is greater than 5 and whose in_color value is not equal to null:

```
select count(in_color) from plants where in_size > 5
```

MAX

This function returns the largest number in the specified column. It has the following syntax:

```
MAX(col)
```

For example, the following returns the largest or_qty value in the orders table:

```
SELECT MAX(in_itemid) FROM plants
```

The next example returns the largest or_qty value for each group of rows grouped by the or_customer code:

```
SELECT or_customer, MAX(or_qty) FROM orders GROUP BY or_customer
```

Note that MAX is not automatically optimized, so there is no performance advantage if *col* is a primary key. For better performance when getting the maximum value of a key column, use TOP (which is optimized) and an ORDER BY clause with DESC. For example:

```
SELECT TOP 1 in_itemid FROM plants ORDER BY in_itemid DESC
```

MIN

This function returns the smallest number in the specified column. It has the following syntax:

```
MIN(col)
```

For example, the following returns the smallest or_qty value in the orders table:

```
SELECT MIN(in_itemid) FROM plants
```

The next example returns the smallest or_qty value for each group of rows grouped by the or_customer code:

```
SELECT or_customer, MIN(or_qty) FROM orders GROUP BY or_customer
```

Note that MIN is not automatically optimized, so there is no performance advantage if *col* is a primary key. For better performance when getting the minimum value of a key column, use TOP (which is optimized) and an ORDER BY clause. For example:

```
SELECT TOP 1 in_itemid FROM plants ORDER BY in_itemid
```

SUM

This function returns the sum of the numbers in *col*, a column that contains numeric values. Null rows are ignored. It has the following syntax:

SUM(*col*)

For example, the following returns the sum of all or_qty values for the orders table:

```
SELECT SUM(or_qty) FROM orders
```

The next example returns the sum of or_qty values for each group of rows grouped by the or_customer code:

```
SELECT or_customer, SUM(or_qty) FROM orders GROUP BY or_customer
```

Scalar Functions

The following scalar functions are supported by *xf*ODBC. Note that parameters with "exp" represent expressions or the results of expressions (e.g., *str_exp* represents a string expression or the string produced by an expression). Also note that scalar functions are not designed to evaluate null values, so it's best to include an AND IS NOT NULL clause when using a scalar function.

ABS

This function returns the absolute value of a numeric expression, and has the following syntax:

`ABS(`*num_exp*`)`

For example, if *num_exp* produces the value -5, the ABS function will return 5.

ASCII

This function returns the ASCII code (an integer value) for the leftmost character in *str_exp*, and has the following syntax:

`ASCII(`*str_exp*`)`

For example, if *str_exp* produces the string "Main Street Plants", ASCII will return 77, the ASCII decimal code for M.

CAST

This function converts an expression (*exp*) or null to the specified data type, and has the following syntax:

`CAST(`*exp*`|NULL AS` *datatype*`)`

where *datatype* is one of the following:

SQL_BIGINT	SQL_DOUBLE	SQL_TIME
SQL_BIT	SQL_FLOAT	SQL_TIMESTAMP
SQL_CHAR	SQL_INTEGER	SQL_TINYINT
SQL_DATE	SQL_NUMERIC	SQL_VARCHAR
SQL_DECIMAL	SQL_SMALLINT	

CAST also supports the database data types listed in CREATE TABLE on page B-53—for example, integer and number(*n*). Note, however, that no truncation occurs and that *n* in the following is ignored: numeric(*n*), char(*n*), and varchar(*n*).

If the specified data type cannot store the entire result, the data will either be truncated or you will get a "data truncated" warning, and the field that caused the warning will have an undetermined value. For alphanumeric data types, the data will be truncated. For numeric data types, the data will be truncated only if truncating removes the fractional portion of the numeric data.

The following example returns term codes as SQL_NUMERIC values.

```
SELECT CAST(or_terms AS SQL_NUMERIC)
  FROM orders WHERE or_odate < '1995-01-01'
```

The next example returns null values as integer values. (By default, null values are returned as char values.) Without the CAST statement, the CASE statement wouldn't work because all of its return values, including the return value for ELSE, must have the same data type.

```
SELECT CASE  in_shape
  WHEN 'tree' THEN 1
  ELSE CAST(NULL AS SQL_INTEGER)
  END
FROM plants
```

CHAR_LENGTH or CHARACTER_LENGTH

This function returns the number of characters or bytes in a string and has the following syntax:

`CHAR_LENGTH(`*exp*`)`

or

`CHARACTER_LENGTH(`*exp*`)`

If the expression (*exp*) results in a character data type, this function returns the number of characters in the resulting string. Otherwise, this function returns the number of bytes in the resulting string. This number is the smallest integer that's greater than or equal to the number of bits divided by 8.

CHR

This function returns the ASCII character equivalent of *numeric_exp*. It has the following syntax:

`CHR(`*numeric_exp*`)`

CONCAT

This function returns a string that's the concatenation of *str_exp1* and *str_exp2*. It has the following syntax:

`CONCAT(`*str_exp1, str_exp2*`)`

CONVERT

This function converts an expression (*exp*) to a specified data type (*datatype*) and has the following syntax:

`CONVERT(`*exp, datatype*`)`

where *datatype* is one of the keywords listed for the CAST scalar function (see CAST on page B-32).

Note the following:

- You may need to use the ODBC escape sequence ("{fn" and an ending brace "}") with this scalar function:

  ```
  SELECT or_number, {fn CONVERT(or_number, SQL_BIGINT)},
                    {fn CONVERT(or_number, SQL_SMALLINT)}
    FROM orders
  ```

- Data may be truncated if the specified data type cannot store the entire result of the expression. For example, if you convert a bigint to a decimal, you may lose precision.

CURDATE

This function returns the current date. It has no argument:

```
CURDATE()
```

CURTIME

This function returns the current time. It has no argument:

```
CURTIME()
```

DATABASE

This function returns the name of the connected database. It has no argument:

```
DATABASE()
```

DAYNAME

This function returns the name of the day of the week for the date specified by *date_exp*. It has the following syntax:

DAYNAME(*date_exp*)

For example, if the value of or_edate in the sample database is 2000-10-01, the following will return "Sunday".

```
SELECT DAYNAME(or_edate) FROM orders WHERE or_vendor=44
```

DECODE

This function returns different values depending on the value of an expression (*col*). It has the following syntax:

DECODE(*col, exp1, return1[, exp2, return2]...[, default]*)

If *col* matches an *exp** value (*exp1*, *exp2*, etc.), DECODE returns the corresponding *return** value. If *col* doesn't match an *exp** value, DECODE returns the default value or NULL if *default* isn't passed.

Note that all return expressions must be the same data type.

The following example returns ‘RED ZONE’ if the value for *in_zone* is 2. If the value for *in_zone* is 4, the statement returns ‘BLUE ZONE’. If *in_zone* is neither 2 nor 4, the statement returns ‘BLACK ZONE’, the default.

```
SELECT DECODE(in_zone, 2, 'RED ZONE', 4, 'BLUE ZONE', 'BLACK ZONE')
    FROM plants
```

GREATEST

This function returns the greatest of a specified set of values (*exp*, *exp2*, etc.). It has the following syntax:

`GREATEST(`*exp, [exp2, ...]*`)`

For example, when used with the sample database distributed with Connectivity Series, the following query returns 2000-10-01, which is the value for or_odate. (For order number 7, or_odate is 2000-10-1 and or_edate is 1993-03-07.)

```
SELECT GREATEST(or_odate, or_edate) FROM orders
    WHERE or_number=7
```

HOUR

This function returns the hour portion of the return value for a time expression (*time_exp*). It has the following syntax:

`HOUR(`*time_exp*`)`

The hour is returned as an integer in the range of 0-23.

IFNULL

Depending on whether a specified expression is null, this function either returns the expression itself or a different value. It has the following syntax:

`IFNULL(`*exp, return_if_null*`)`

If *exp* is null, IFNULL returns *return_if_null*. If *exp* is not null, *exp* is returned. *Return_if_null* must have a data type that’s compatible with the data type of *exp*. For example, the following will not work:

```
SELECT IFNULL(or_sdate, '1999-04-05') FROM orders
```

The next example, however, is correct:

```
SELECT IFNULL(or_sdate, {d '1999-04-05'}) FROM orders
```

Note that *xf*ODBC interprets zero-length strings as nulls, so clauses like the following are unnecessary: IFNULL(strng, ‘’).

INSTR

This function returns the position of the first character of a string (*str_exp2*) within another string (*str_exp1*). It has the following syntax:

`INSTR(`*str_exp1, str_exp2[, n[, m]]*`)`

If the *n* argument is not specified, INSTR searches *str_exp1* and returns the position of the first character in the first occurrence of *str_exp2*. If the *n* argument is specified, INSTR begins the search *n* characters into *str_exp1*. If the *n* argument is negative, the search begins *n* characters from the end of *str_exp1*. If *m* is specified, the position of the *m*th occurrence of *str_exp2* in *str_exp1* is returned. If *str_exp2* does not exist in *str_exp1*, this function returns 0.

LCASE

This function converts any uppercase characters in a string (*str_exp*) to lowercase characters, and returns the resulting string. It has the following syntax:

`LCASE(`*str_exp*`)`

LEAST

This function returns the least of the specified values (*exp*, *exp2*, etc.). It has the following syntax:

`LEAST(`*exp, [exp2, ...]*`)`

For example, when used with the sample database distributed with Connectivity Series, the following query returns 1993-03-07, which is the value for or_edate. (For order number 7, or_odate is 2000-10-1 and or_edate is 1993-03-07.)

```
SELECT LEAST(or_odate, or_edate) FROM orders WHERE or_number=7
```

LEFT

This function returns the first *n* characters of a string (*str_exp*). It has the following syntax:

`LEFT(`*str_exp, n*`)`

For example, if *str_exp* produces the string “ablebaker” and the value of n is 4, LEFT returns “able”.

Note that you must use the ODBC escape sequence ("{fn" and an ending brace "}") with this scalar function. For example:

```
SELECT {fn LEFT(cust_name, 4)} FROM customers
```

LENGTH

This function returns the number of characters in a string (*str_exp*) minus any trailing blanks. It has the following syntax:

`LENGTH(`*str_exp*`)`

LOCATE

This function returns the position of the first character of a string (*str_exp1*) in a string (*str_exp2*). It has the following syntax:

`LOCATE(`*str_exp1, str_exp2[, n[, m]]*`)`

If the *n* argument is not specified, LOCATE searches *str_exp2* and returns the position of the first character in the first occurrence of *str_exp1*. If the *n* argument is specified, LOCATE begins the search *n* characters into *str_exp2*. If the *n* argument is negative, the search begins *n* characters from the end of *str_exp2*. If *m* is specified, the position of the *m*th occurrence of *str_exp1* in *str_exp2* is returned. If *str_exp1* does not exist in *str_exp2*, this function returns 0.

LTRIM

This function returns a string (*str_exp)* with leading blanks removed. It has the following syntax:

`LTRIM(`*str_exp*`)`

NOW

This function returns the current date and time. It has no argument:

`NOW()`

NVL

For a given expression (*exp1*), this function either returns the same expression (if it’s not null) or a different expression (*exp2*) if it is null. (*Exp2* must have the same data type as *exp1*.) It has the following syntax:

`NVL(`*exp1, exp2*`)`

This function enables you to create queries like the following that substitute a string or value for null.

```
SELECT in_name, NVL(in_color, 'n/a')
    FROM plants
```

POSITION

This function returns the position of a character expression (*char_exp1)* in another character expression (*char_exp2)*. It has the following syntax:

`POSITION(`*char_exp1* `IN` *char_exp2*`)`

If *char_exp1* does not exist in *char_exp2*, This function returns 0.

REPLACE

This function searches for a string (*str_exp2*) in a string (*str_exp1*) and replaces occurrences of the found string with another string (*str_exp3*). It has the following syntax:

`REPLACE(`*str_exp1, str_exp2, str_exp3*`)`

For example, the following returns “baker st.”:

```
REPLACE('baker street', 'street', 'st.')
```

REVERSE

This function reverses the order of the characters returned for a string expression. It has the following syntax:

`REVERSE(`*string_exp*`)`

For example, the following returns “cba”:

```
REVERSE('abc')
```

RIGHT

This function returns a given number of characters (*n*) from the end of a string (*str_exp*). It has the following syntax:

`RIGHT(`*str_exp, n*`)`

For example, if the value of *str_exp* is “Border Imports”, and *n* is set to 7, RIGHT returns “Imports”.

Note that you must use the ODBC escape sequence (“{fn” and an ending brace “}”) with this scalar function. For example:

```
SELECT {fn RIGHT(cust_name, 4)} FROM customers
```

ROUND

This function rounds a numeric expression (*num_exp*). It has the following syntax:

`ROUND(`*num_exp[, int_exp]*`)`

If *int_exp* is not specified, ROUND returns *num_exp* rounded to a whole number. If *int_exp* is passed and is positive, ROUND rounds *num_exp* to *int_exp* places to the right of the decimal point. If *int_exp* is negative, *num_exp* is rounded to *int_exp* places left of the decimal point.

RTRIM

This function removes trailing blanks from a string (*str_exp*), and returns the resulting string. It has the following syntax:

`RTRIM(`*str_exp*`)`

SQRT

This function returns the square root of a numeric expression (*numeric_exp*). It has the following syntax:

`SQRT(`*numeric_exp*`)`

This function supports only non-negative real numbers. A negative number will cause an error during SQLFetch. To avoid such errors, we suggest you add a WHERE clause that eliminates rows that have negative values for the column passed to SQRT.

SUBSTR or SUBSTRING

This function returns the substring of a string expression (*str_exp*) that begins at a given position (*n*) and has a given length (*m* characters long). It has the following syntax:

`SUBSTR(`*str_exp, n[, m]*`)`

or

`SUBSTRING(`*str_exp, n[, m]*`)`

If *n* equals 0, the entire string is returned. If *n* is a negative number, this function begins *n* characters (or spaces) from the end of the string and returns *m* characters. For example, if the database has “Main Street Plants” as a customer name, the following statement returns “Street” for that customer.

```
SELECT SUBSTR(cust_name, -13, 6) FROM customers
```

If you don’t specify *m*, SUBSTR returns all characters from the character in the *n* position to the end of the string.

SYSDATE

This function returns the current system date and time for the client system. It has no argument:

```
SYSDATE
```

TO_CHAR

This function converts date or numeric column data into a character string. It has the following syntax:

`TO_CHAR(`*col[, format]*`)`

If *col* is a date column and no format argument is passed, the default format, YYYY-MM-DD, is used. If *col* is a numeric column, you can use the following to specify the format:

0

To add a leading or trailing zero to the result, add a zero to the format string. For example:

Field value	Mask	Return value
`1.49`	`'$000,000.00'`	`'$000,001.49'`

other numerals

To specify a placeholder for a digit in the field value, add any other number besides 0. If there is no corresponding digit in the field, the numeral will be replaced by a space in the result. For example:

Field value	Mask	Return value
`1.49`	`'$999,999.99'`	`'      $1.49'`

symbols and punctuation marks

To include a symbol or punctuation mark in the result, add the symbol or punctuation mark to the format string. Note that if any symbol other than a dollar sign ($) or decimal point will be replaced by a blank space if it is in the first position. (In other words, if no digits precede the symbol.) A dollar sign or decimal point will be included in the resulting string even if it is in the first position. For example:

Field value	Mask	Return value
`65000`	`'*99999'`	`' 65000'`
`65000`	`'99999*'`	`'65000*'`
`65000`	`'$99999'`	`'$65000'`
`.65000`	`'.99999'`	`'.65000'`
`.65000`	`'000.00%'`	`'000.65%'`

Also note that if the result contains a period with no digits preceding it, a zero will precede the period. For example:

Field value	Mask	Return value
`.35`	`'$99999.99'`	`'    $0.35'`
`.35`	`'99999.99'`	`'    0.35'`

B

To instruct the driver to replace a zero with a blank space if the zero is a leading zero, put a capital B in the position of the zero. If a B is in any other position, a zero in that position will remain. For example:

Field value	Mask	Return value
`0035`	`'BBB.BBB'`	`' 35.000'`

other letters

To include a letter in the result (other than capital B), add the letter to the format string. Note if the letter is not preceded by any digits, the letter will be replaced by a blank space.

If *col* is a date/time column, you can use the following format masks:

Date Masks	
Mask	**Description**
AM, PM, am, pm	Two-digit meridian indicator (AM, PM, am, or pm). Case of first character determines case of indicator.
D[a]	Single-digit day of the week (1-7). Weeks start on Sunday.
DAY[a]	Full name of day in uppercase characters
Day[a]	Full name of day with initial character capitalized
day[a]	Full name of day in lowercase characters
DD, dd	Two-digit day of the month (01-31)
DDD	Three-digit day of the year (001-356)
DY[a]	Uppercase three character day
Dy[a]	Three character day with initial character capitalized
dy[a]	Lowercase three character day
HH, HH12	Two-digit hour (00-11)
HH24	Two-digit hour (00-23)
J	Julian day (does not support BC dates)
MI	Two-digit minute (00-59)
MM, mm	Two-digit month (01-12)
MON	Uppercase three character month
Mon	Three character month with initial character capitalized
mon	Lowercase three character month
MONTH[a]	Full name of the month in uppercase characters
Month[a]	Full name of the month with initial character capitalized
month[a]	Full name of the month in lowercase characters
Q[a]	Single-digit quarter (1-4)
RR	Two-digit year from another century (a sliding window format based on 20)

Date Masks (Continued)	
Mask	**Description**
SS	Two-digit second (00-59)
SSSSS	Number of seconds past midnight (00000-86399)
UUUUUU	Microsecond
W[a]	Single-digit week of month (1-4)
WW[a]	Two-digit week of year (01-52)
YY	Two-digit year
YYYY	Four-digit year

a. Not supported by TO_DATE.

To add the letters (th, rd, and so forth) needed to create ordinal numbers, such as “5th” or “3rd”, add “th” to any uppercase digit mask. For example, if or_odate is 1993-03-01, the following will return 2ND because 1993-03-01 was a Monday.

```
SELECT TO_CHAR(or_odate, 'Dth') FROM orders WHERE or_number = 3
```

Given the same date, the next example returns “060th day of 1993”:

```
SELECT to_char(or_odate, 'dddth "day" of YYYY') FROM orders
    WHERE or_number = 3
```

Note that “day” is in quotation marks in the last example. If you want the resulting string to include mask characters, enclose the character(s) in quotation marks.

The following example retrieves or_odate values and formats them in month, day, four-digit year order (MMDDYYY):

```
SELECT to_char (or_odate, 'MMDDYYYY') FROM orders
```

TO_DATE

This function converts a string expression (*str_exp*) to a date or datetime data type. It has the following syntax:

TO_DATE(*str_exp[, format]*)

If no format is specified, the default date format is used (YYYY-MM-DD). For example:

```
SELECT TO_DATE(or_odate) FROM orders
```

For information on formats, see the date masks in TO_CHAR on page B-39 and note the following:

- TO_DATE supports all of the masks listed *except* MONTH, Month, month, D, DAY, Day, day, DY, Dy, dy, Q, W, and WW.
- The AM, PM, am, and pm masks work only if the TO_DATE clause is cast as SQL_TIMESTAMP or SQL_TIME. For example, the following returns “1990-06-10 22:10:02.000000”. (Note that the returned hour would be 10 if the value had no AM/PM indicator and the mask didn’t include an AM, PM, am, or pm mask.)

  ```
  SELECT CAST(TO_DATE('10-06-1990 10:10:02 PM',
      'DD-MM-YYYY HH:MI:SS AM') AS SQL_TIMESTAMP)
      FROM dual
  ```

TO_NUMBER

This function converts the results of a string or character expression (*exp*) into a numeric value. It has the following syntax:

TO_NUMBER(*exp*)

Note that even though *exp* can be a character or a string, it must contain data. For example:

```
SELECT or_number, {fn TO_NUMBER(or_item)} FROM orders
```

TRANSLATE

This function replaces characters in a string. It has the following syntax:

TRANSLATE(*str_exp, str_exp_from, str_exp_to*)

TRANSLATE returns *str_exp* after replacing each character in *str_exp_from* with the character(s) in the corresponding position in *str_exp_to*. For example, the following TRANSLATE clause results in the string “1bcd23”:

```
TRANSLATE('AbcdEF', 'AEF', '123')
```

If *str_exp_to* is empty, all characters in *str_exp_from* are removed. The following, for example, results in the string “def”:

```
TRANSLATE('abcdef', 'abc', '')
```

Note that *str_exp_from* cannot be larger than *str_exp_to*. (If it is, you will get an error: “Illegal parameters for function TRANSLATE”.) *Str_exp_from*, however, can be smaller than *str_exp_to*, as long as the number of characters in *str_exp_from* is a multiple of the number of characters in *str_exp_to*. For example, the following will work (if desc is large enough to hold the result). Every left angle bracket (<) will be replaced with &lt;. Every right angle bracket (>) will be replaced with &gt;.

```
TRANSLATE(desc,'<>','&lt;&gt;')
```

Note that if you want to change the entire value for a column (rather than just selected characters), use DECODE.

TRUNC

This function removes the fractional portion of a number (*num_exp*) or returns a date/time value (*datetime_exp*) with the time portion set to zeros. It has the following syntax:

TRUNC(*num_exp*|*datetime_exp*)

For example, if the value for a numeric field in a given record is 1.05, applying the TRUNC function to the field will return the value 1, as in the following:

```
SELECT TRUNC(or_price) FROM orders WHERE or_number=3
```

The next series of SQL statements results in “1993-03-01 00:00:00”:

```
CREATE TABLE date_table(datetime_field datetime)

INSERT INTO date_table (datetime_field)
  VALUES ('1993-03-01 17:02:20')

SELECT TRUNC(datetime_field) FROM date_table
```

UCASE

This function converts lowercase characters in a string expression (*str_exp*) to uppercase characters and returns the resulting string. It has the following syntax:

UCASE(*str_exp*)

USER

This function returns the name of the user for the data source, which may be different than the name used to log in to the data source. It has no argument:

```
USER()
```

Bitwise Functions

The following is a list of bitwise functions supported by *xf*ODBC. Note that “exp” represents an expression that results in an numeric value. *xf*ODBC converts non-integer numeric values into integers before performing a bitwise operation.

BITAND

This function returns the result of a bitwise AND operation performed on two numeric values. It has the following syntax:

`BITAND(`*num_exp1*, *num_exp2*`)`

For example, the following query returns 1 (0001 AND 1111 = 0001).

```
SELECT BITAND(1, 15) FROM dual
```

BITOR

This function returns the result of a bitwise OR operation performed on two numeric values. It has the following syntax:

`BITOR(`*num_exp1*, *num_exp2*`)`

For example, the following query returns 15 (0010 OR 1101 = 1111).

```
SELECT BITOR(2, 13) FROM dual
```

BITXOR

This function returns the result of a bitwise exclusive OR operation performed on two numeric values. It has the following syntax:

`BITXOR(`*num_exp1*, *num_exp2*`)`

For example, the following query returns 14 (0001 XOR 1111 = 1110).

```
SELECT BITXOR(1, 15) FROM dual
```

Statements that Modify Data

*xf*ODBC supports the following SQL statements that modify data:

- DELETE on page B-47
- INSERT on page B-47
- UPDATE on page B-48

Note the following:

- For updating Synergy databases, we strongly recommend using a Synergy application that's designed to efficiently maintain database integrity. If you use an ODBC-enabled application to update a Synergy database, you may run into record-locking issues.
- We strongly recommend that you prevent applications that use the Microsoft Jet database engine (including Microsoft Access and Query) from updating Synergy databases. Failure to do so may result in unsupportable situations. These applications often have record locking issues (they may lock more than just the record that's being updated), and there are often no referential checks or triggers to ensure database integrity. In addition, these applications may allow users to make bulk changes without your control. (Different versions of Jet will give you different results, and though we recommend at least Jet 4 service pack 8, Synergy/DE installations do not update Jet.)

DELETE

This command deletes a row, a group of rows, or all rows in a table. It has the following syntax:

`DELETE FROM` *table_name [table_alias] [*`WHERE` *search_condition]*

where *table_name* is the name of a table or view, *table_alias* is an alias, and *search_condition* is the selection criteria for the rows.

For example, the first statement below deletes all rows in mytable. The second deletes only rows that meet the WHERE clause criteria.

```
DELETE FROM mytable
```

```
DELETE FROM mytable WHERE col_1 > 2 OR col_2 < 4
```

Note that DELETE cannot be used with a view. Use it with the base table instead.

INSERT

This command inserts one or more rows into a table. It has the following syntax;

```
INSERT INTO table_name [(col_1[, col_2, ...])]
    VALUES (value_1[, value_2...])
```

or

```
INSERT INTO table_name [(col_1[, col_2, ...])]
    subquery
```

where

table_name is the name of the table that the rows will be added to.

col_# are columns in *table_name* that values will be specified for.

value_# are values to be inserted.

subquery is a SELECT statement whose results will be inserted.

The first form of INSERT syntax inserts a single row into a table. The second form inserts as many rows as are returned by the subquery.

If you don’t list columns (i.e., if you omit *col_1, col_2, ...*), you must supply a value for each column in the table (*table_name*). Either include a *value_#* for each column, or make sure the select list for *subquery* includes an item for each column. The order of values (*value_#*) or *subquery* select list items must match the order of the columns in *table_name*. For example, the following includes a *value_#* for each column in the ORDERS table for the sample database:

```
INSERT INTO orders
    VALUES (11, 42, 1, 24, 5, 2.55, '01', 1993-04-18, null,
            2000-10-01, 586455)
```

If you do list columns (*col_1, col_2, ...*), the specified values (*value_#*) or the items in the select list for *subquery* must correspond to the specified columns. For example:

```
INSERT INTO mytable (col_1, col_2) VALUES (1, 1)

INSERT INTO orders (or_number, or _item, or_price)
    SELECT 12, in_itemid, in_price FROM plants
        WHERE in_name = 'Wedelia'
```

Note that if you list columns, columns you leave out of the list are set to null, so if a column can't accept null values, you must include it in the column list. Additionally, if *table_name* includes an overlay column that is not read-only and is not set by the INSERT, the overlay column is set to null, which sets all of the fields included in the overlay to null. To prevent this, set overlay columns to read-only by setting their corresponding repository fields to read-only. Then regenerate the system catalog.

Note the following for both syntax forms:

- You must use single quotes to specify string values.
- INSERT cannot be used with a view. Use it with the base table instead.

UPDATE

This command changes a single row, groups of rows, or all of the rows in a table. You can specify the rows you want to change and a new value. The new value can be a constant or an expression. UPDATE has the following syntax:

`UPDATE` *table_name [table_alias]* `SET` *column* = *value|subquery [[, column* = *value|subquery]...]*
    [`WHERE` *condition]*

where

table_name is the is the name of the table or a view for the table that will be updated. (If *table_name* is a view, the underlying table will be updated.)

table_alias is an alias for the table or view.

column is a column in *table_name*.

value is the value (or an expression that results in a value) that *column* will be set to.

subquery is a subquery whose results will be used to update *column*. Note that subqueries in SET clauses must use SELECT DISTINCT to ensure only one value is returned for the column.

condition is the criteria used to determine if a row will be updated.

For example:

```
UPDATE mytable SET col_3 = 'Fine'

UPDATE mytable SET col_3 = 'Fine' WHERE col_1 > 2

UPDATE mytable SET col_2 = 3, col_3 = 'Fine' WHERE col_2 > 2

UPDATE orders SET
    or_vendor = (SELECT DISTINCT vend_key FROM vendors
        WHERE vend_name = 'Border Imports'),
    (or_item, or_price) = (SELECT DISTINCT in_itemid, in_price
        FROM plants
        WHERE in_name = 'Wedelia')
    WHERE or_number = 3
```

Note the following:

- UPDATE cannot be used with a view. Use it with the base table instead.
- For Synergy ISAM files, you can't use UPDATE to change a value in a non-modifiable key column. To change a value in a key column, you must delete the row and then insert a new row that contains the change.

Statements that Define the Schema (DDL)

*xf*ODBC supports the following SQL statements that define the schema:

- CREATE INDEX on page B-51
- CREATE SYNONYM on page B-52
- CREATE TABLE on page B-534
- CREATE VIEW on page B-55
- DROP SYNONYM on page B-55
- DROP TABLE on page B-56
- DROP VIEW on page B-56

CREATE INDEX

The CREATE INDEX command creates an index for a specified table. It has the following syntax:

```
CREATE [UNIQUE] INDEX index_name ON table_name
    (column_name [ASC|DESC][, column_name [ASC|DESC]]...)
```

where

index_name is the name of the index that will be created. UNIQUE specifies that no two rows of the index can have the same value.

table_name is the name of the table that the index will be created for.

column_name is the name of the column to create the index on. ASC|DESC specifies the sort direction for the column (ascending or descending).

For example:

```
CREATE INDEX my_key1 ON public.orders (or_vendor DESC)
```

```
CREATE INDEX my_key1 ON public.orders (or_vendor, or_item)
```

Note the following:

- CREATE INDEX works only with ISAM files and is supported only after the initial CREATE TABLE and before the first INSERT.
- If you specify more than one *column_name*, the index key is built using the columns in the order that they are listed in the statement.
- For existing tables, if you use CREATE INDEX with an existing table, the file for the table is opened (if it isn't already open) and a temporary index is created. This index will last for the life of the connection.
- For new tables, see the notes for CREATE TABLE on page B-53 and note that you must execute the CREATE INDEX statement before any SQL statement on the new table you create. If you don't, the default key on the first column is used as a primary key with duplicates allowed. Once the file is created and you execute the CREATE INDEX statement, you must reorganize the data file manually.
- For both new and existing tables on OpenVMS, each column must have the same sort direction (ascending or descending).

CREATE SYNONYM

The CREATE SYNONYM command creates a synonym, which is an alternate name for a table or view. It has the following syntax:

```
CREATE SYNONYM [owner_name.]synonym_name FOR
      [owner_name.]object_name
```

where

owner_name is the name of the schema that will contain the synonym. If you don't specify *owner_name*, the synonym is created in your default schema.

synonym_name is the name of the synonym you are creating.

object_name is the name of the object (table or view) that the synonym will be created for.

Note the following:

- For every table in a system catalog, *xf*ODBC creates a default synonym that consists only of the table name—no owner name. (For example, for the public.orders table in the sample database, *xf*ODBC creates the synonym "orders".) When you create a synonym for a table, your synonym overwrites the default synonym, so you won't be able to use it anymore. For example, if you issue the following commands,

  ```
  SELECT * FROM orders
  ```

  ```
  CREATE SYNONYM newname FOR public.orders
  ```

  this statement will result in an error:

  ```
  SELECT * FROM orders
  ```

- If you drop a synonym you've created for a table, *xf*ODBC re-creates the default synonym.

CREATE TABLE

The CREATE TABLE command creates a table and its columns. It creates ISAM files for the table and adds table information to the system catalog. CREATE TABLE has the following syntax:

```
CREATE TABLE [owner.]table_name
      (column_definition [, column_definition]...)
```

where *table_name* is the name of the table to be created, and *column_definition* is the following:

column_name data_type [`NOT NULL`*]*

Data_type must be one of the following. Note that these SQL data types are not related to Synergy DBL types; instead they are the SQL ODBC data types we support within ODBC only. (The only data types that can be mapped directly in a Synergy application are char/varchar type to alpha, smallint to **i2**, and int to **i4**. Other than these, none of the data types can be directly used in non-SQL Connection Synergy applications.)

CREATE TABLE Data Types

Data_type	Size,	Described as...
char*[*(*n*)*]*	*n* (default is 1, maximum is 4000)	SQL_VARCHAR
date	10 (YYYY-MM-DD)	SQL_TYPE_TIMESTAMP
datetime	19 (YYYY-MM-DD HH:MI:SS)	SQL_TYPE_TIMESTAMP
decimal*[*(*p[,s]*)*]*	*p* is precision (default is 10, maximum is 28) *s* is scale (default is 0, maximum is 28)	SQL_DECIMAL
double	16 (equivalent to decimal(16,6))	SQL_FLOAT
integer	10	SQL_INTEGER
number*[*(*p[,s]*)*]*	*p* is precision (default is 10, maximum is 28) *s* is scale (default is 0, maximum is 28)	SQL_DECIMAL
numeric*[*(*p[,s]*)*]*	*p* is precision (default is 10, maximum is 28) *s* is scale (default is 0, maximum is 28)	SQL_DECIMAL
real	8 (equivalent to decimal(8,6))	SQL_DECIMAL
smallint	5	SQL_SMALLINT
time	8 (HH:MI:SS)	SQL_TYPE_TIMESTAMP
timestamp	19 (YYYY-MM-DD HH:MI:SS)	SQL_TYPE_TIMESTAMP
varchar*[*(*n*)*]*	*n* (default is 1, maximum is 4000)	SQL_VARCHAR

NOT NULL prevents a column from being updated with null values and values that *xf*ODBC considers null. See "Preventing null updates and interpreting spaces, zeros, and null values" on page 3-27.

For example:

```
CREATE TABLE mytable (col_1 integer NOT NULL, col_2 char(10),
    col_3 decimal(4), col_4 decimal(5,2))
```

Note the following:

- A default key will be created on the first column. (Duplicates are allowed, but no modification of key values are allowed.)
- Files are created with the first SELECT or INSERT to a table, not with the CREATE TABLE statement.
- Filenames consist of the owner name, the percent sign (%), the table name, and .ISM and .IS1 extensions. For example, if you login as public and create a table named ORG, the PUBLIC%ORG.ISM and PUBLIC%ORG.IS1 files are created in the first datasource path directory.
- Filenames are in all uppercase characters. If you use DBLCASE with 'l' option, filenames are converted to lowercase, and the Synergy driver will not open the new table files. For information on this environment variable, see DBLCASE in the "Environment Variables" chapter of *Environment Variables &s System Options*.
- If you use NOT NULL for a column, that column must be included in every INSERT statement for the table.
- If you overwrite the system catalog (with the **-c dbcreate** option or the "Clear and re-create catalog" DBA option), you won't be able to use *xf*ODBC to access a table created with CREATE TABLE unless you added the table information to the repository before regenerating. The CREATE TABLE command does not add table information to the repository.

CREATE VIEW

The CREATE VIEW command creates a logical view of one or more tables or one or more views. (You can use joins to include multiple views or tables.) Views contain data from tables, have columns, and otherwise appear as tables, but they're not the actual database tables. You can use views to present table information in different ways and to enable users to view data without having access to the actual database tables.

CREATE VIEW has the following syntax:

```
CREATE VIEW [user_name.]view_name (view_col [, ...])
      AS sel_stmnt
```

where

user_name is the table owner name.

view_name is the name of the resulting view.

view_col is the name of the column in the view. Column names are generally optional, but they are required if more than one column in the resulting view has the same name (usually because of a join) or if a column is derived from an arithmetic expression, function, or constant value. Column names may also be assigned in the SELECT statement by assigning correlation names to the columns. Note that if you do name columns, you must name them all, and they must all have different names.

sel_stmnt is a SELECT statement.

For example:

```
CREATE VIEW contacts (Company, Contact, Phone)
    AS SELECT cust_name, cust_contact, cust_phone
    FROM customers
```

The following uses a join:

```
CREATE VIEW cust_orders AS
    SELECT orders.or_item, orders.or_number, customers.cust_name
    FROM {OJ public.orders LEFT OUTER JOIN public.customers
    ON orders.or_customer = customers.cust_key}
```

Note that views cannot be used to update, insert, or delete rows.

DROP SYNONYM

The DROP SYNONYM command deletes a synonym. It has the following syntax:

```
DROP SYNONYM [owner_name.]synonym_name
```

where *owner_name* is the name of the schema that contains the synonym, and *synonym_name* is the name of the synonym you want to delete.

DROP TABLE

The DROP TABLE command removes a table. It has the following syntax.

`DROP TABLE` *[owner_name.]table_name*

where *owner_name* is the name of the schema that contains the table, and *table_name* is the name of the table or view you want to delete.

DROP VIEW

The DROP VIEW command deletes a view. It has the following syntax:

`DROP VIEW` *[owner_name.]view_name*

where *owner_name* is the name of the schema that contains the view, and *view_name* is the name of the view you want to delete.

Note the DROP TABLE command can also drop views.

Statements that Set Options

The SET OPTION command enables you to set SQL options. Note that you can either set these by including them in a SQL statement run from the third-party application that accesses Synergy data or by including them in a query processing options file, which is a text file that the GENESIS_INITSQL environment variable is set to. (There is an exception: *xf*ODBC ignores TMPINDEX settings in query processing options files.) For more information, see "Creating a file for query processing options" on page 8-17.

SET OPTION

The SET OPTION command sets SQL options. It has the following syntax:

`SET OPTION` *option_type param1*

or

`SET OPTION` *option_type param1 param2*

where *option_type*, *param1*, and *param2* are the following:

SQL Options			
Option_type	**Param1**	**Param2**	**Description**
COMPSORT	ON|OFF		Sets sort page compression. By default COMPSORT is ON.
DATETIME	*[n] format_str*		Enables you to modify the conversion masks used to interpret dates and times that are part of SQL statements. See "Notes on DATETIME" on page B-60.
ERROR	ON|OFF		Records internal error information for use by Synergy/DE Developer Support. To use this, LOGFILE must also be set. By default ERROR is OFF.
EXPR	ON|OFF		Records internal expression information for use by Synergy/DE Developer Support. To use this option, LOGFILE must also be set. By default EXPR is OFF.
HASH	ON|OFF		Records internal hash information for use by Synergy/DE Developer Support. To use this, LOGFILE must also be set. By default HASH is OFF.

SQL Options (Continued)			
Option_type	**Param1**	**Param2**	**Description**
HEAPBLOCKSIZE	*bytes*		Sets the minimum heap block size (in bytes) used to allocate memory. (Larger blocks may be allocated.) This is set to 32768 by default. We don't recommend changing this, but you can set it to any value from 0 to 1000000 (inclusive). Larger sizes require less CPU overhead but may result in excessive memory use. If you set this to zero, heap blocks are allocated in the exact sizes needed.
LOGFILE	'*filename*'		Sets the name and location of the debugging log file. Generally this file is used in conjunction with PLAN to record information on the index(es) used for a query. The other logging options (ERROR, EXPR, HASH, TRACE, and TREE) are for use by Synergy/DE Developer Support.
MAXOPTLOOP	*max_combos*		Limits the number of ways the query optimizer will try to optimize a query that has a multi-table join. Setting this option may enable you to reduce the time it takes to process the query. By default, *max_combos* is set to 100, which limits the query optimizer to trying 100 configurations. You can, however, set it to any positive number, or you can set it to 0, which turns off this feature, allowing the optimizer to try any number of combinations.
MERGESIZE	*max_rows*		(Windows only) Optimizes SELECT statements that have one or more OR clauses and one or more AND clauses. See "Notes on MERGESIZE" on page B-61.
OPTIMIZE	ON|OFF		Enables you to control whether or not optimization is used. By default OPTIMIZE is ON.
PLAN	ON|OFF		Records indexes used for a query. To use this option, LOGFILE must also be set. By default PLAN is OFF. See "Notes on PLAN" on page B-61.

SQL Options (Continued)

Option_type	Param1	Param2	Description

PREOPT	ON|OFF		Optimizes SELECT statements that have an IN clause or an OR clause that is part of an AND clause. For example:
			SELECT myfield FROM mytable
			  WHERE id=1
			  AND num IN (48,49,50)
			or
			SELECT myfield FROM mytable
			  WHERE id=1
			  AND (num=48 OR num=49
			    OR num=50)
			When set to ON (the default), the fields on both sides of the AND clause (id and num in the example) are included in the key to optimize the statement. When set to OFF, the field for the IN or OR side of the AND clause is omitted from the key.

SORTPAGES	*totalpages*	*mempages*	Sets the amount of disk and memory storage used for sort operations for subsequently opened cursors. See “Notes on SORTPAGES” on page B-63.
TMPINDEX	ON|OFF		Enables or disables temporary indexes for inner joins. (Temporary indexes are used only for inner joins.) By default, TMPINDEX is on. Note that this option will be ignored if it’s in a query processing options file (a file specified by the GENESIS_INITSQL environment variable).
TRACE	ON|OFF		Records internal trace information for use by Synergy/DE Developer Support. To use this, LOGFILE must also be set. By default TRACE is OFF.
TREE	ON|OFF		Records internal tree information for use by Synergy/DE Developer Support. To use this, LOGFILE must also be set. By default TREE is OFF.

The following are some example settings:

```
SET OPTION DATETIME 0 'DD-MM-YYYY HH:MI'
SET OPTION DATETIME 1 'DD-MM-YYYY'
SET OPTION DATETIME 2 'HH:MI'
SET OPTION LOGFILE 'mylogfile'
SET OPTION MAXOPTLOOP 100
SET OPTION MERGESIZE 0
SET OPTION TRACE ON
SET OPTION PLAN ON
SET OPTION SORTPAGES 8000 2000
SET OPTION TMPINDEX ON
```

Note that if you use SDMS logging as you run SET OPTION commands, you'll see begintx/endtx pairs in the log file. This is the correct and expected behavior.

Notes on DATETIME

This option enables you to modify the conversion masks used to interpret dates and times that are part of SQL statements. Dates and times in SQL statements are strings and must be converted to the *xf*ODBC driver's internal format. This command works with SQL Connection when using the Synergy Database driver (VTX4), and it works with *xf*ODBC. (This option is ignored, however, if you put it in a query processing options file specified by the GENESIS_INITSQL environment variable.)

There are four masks. By default they are set to the following:

0	YYYY-MM-DD HH:MI:SS
1	YYYY-MM-DD
2	HH:MI:SS
3	YYYY-MM-DD HH:MI:SS.UUUUUU

The *xf*ODBC driver attempts to use the 0 mask first. If this doesn't match the data, it tries the 1 mask. If that doesn't match, it tries the 2 mask, and finally it tries the 3 mask.

To modify one of these masks, use the *n* parameter to specify which mask to set. If you don't specify the *n* parameter, the 0 mask is set by default.

For information on the characters you can include in the *format_str* string, see the date masks listed in TO_CHAR on page B-39.

For information on setting the *xf*ODBC display format for dates, see chapter 8, "Configuring Data Access."

Notes on MERGESIZE

This optimizes SELECT statements that have one or more OR clauses by evaluating each side of each OR clause as a separate SELECT statement and then combining the results (in a multimerge operation). This works only when keys are available to optimize each side of each OR clause. And note that in some cases, this feature could impair performance. The *max_rows* argument specifies the maximum number of rows that can be returned when a statement is optimized with multimerge.

Max_rows must be either 0 (which turns this feature off) or a positive numeric value from 100 to 999999. (Anything from 1 to 99 will cause a runtime error.)

- If the result set of an optimized statement is larger than *max_rows*, an error is generated.
- If the statement is not optimized with multimerge, the *max_rows* limit doesn't apply.

By default MERGESIZE is enabled (the default setting is 10000) because some applications automatically generate the kind of statements that MERGSIZE optimizes. For example, if you use Microsoft Access to issue a query that selects all of the columns in a table, Access generates a SELECT statement with a series of OR clauses that repeatedly specify key segments.

Note the following:

- With each row allowed by the *max_rows* value, *xf*ODBC uses six bytes of memory, so setting *max_rows* to a large value doesn't generally affect performance.
- You can also set this option for a DSN by setting the "Max number of rows" field in the xfODBC Setup dialog box, the dialog box that enables you to add and configure *xf*ODBC DSNs. For more information, see "Adding a user or system DSN" on page 8-5.

Notes on PLAN

To find out which indexes are used for a query, use SET OPTION PLAN in conjunction with SET OPTION LOGFILE. This generates a log file that includes a "Pushed key#" line that lists the index used to optimize the query, and it includes a table that lists index information. For example, if you create a query for the sample database (distributed with Connectivity Series) so that it selects rows from the VENDORS table where VEND_RTYPE = 1 and VEND_KEY < 44, your log file will contain something like the following:

```
PUBLIC.VENDORS has 4 keys, Table/Buffer: 0/0
Key NKC Unq Nul Col/Dty/Dir/Name
--- --- --- --- -----------------------------------------
0   1   Y   N  0/2/A/VEND_KEY
1   1   Y   N  0/2/A/VEND_KEY
2   2   Y   N  1/2/A/VEND_RTYPE 0/2/A/VEND_KEY
3   1   Y   N  11/98/D/ROWID
```

```
Execution plan
--------------
Tables ........... 1
Keys used ........ 1
Columns pushed ... 2
ANDs not pushed .. 0
Plan chosen ...... 3 of 4

Fetch node for table: 'PUBLIC.VENDORS'
 Cursor# ......... 2
 Buffer# ......... 0
 Table# .......... 0
 Pushed key# ..... 2, Col/Dty/Dir/Name: 1/2/A/VEND_RTYPE 0/2/A/VEND_KEY
  KeySeg 0 oper .. =
   Constant ...... dty: 2, flg: 0, len: 2, 1
  KeySeg 1 oper .. <
   Constant ...... dty: 2, flg: 0, len: 2, 44
```

In this case, the index used for the query (listed on the “Pushed key#” line) is 2. To find out what this is, look for “2” in the Key column of the table. The KeySeg lines under “Pushed key#” list segment information for the index and correspond to the segments listed on the “Pushed key#” line. The first segment listed on this line is considered segment 0, the second is considered segment 1, and so on. So, for the example above, KeySeg 0 refers to VEND_RTYPE, and KeySeg 1 refers to VEND_KEY.

Note that you can also use this information to get key of reference (KRF) information. To do this, you’ll need to run the following query:

```
SELECT a.i_table, a.i_name, a.i_type, a.columns, a.keynum,
       b.x_name, b.x_position
  FROM genesis_indexes a, genesis_xcolumns b
  WHERE (a.i_name=b.x_index)
    AND (a.i_table=b.x_table)
    AND (a.i_owner=b.x_owner)
    AND (a.i_database=b.x_database)
    AND (a.i_table='table_name')
  ORDER BY 5, 2, 7
```

where *table_name* is the case-sensitive name of the table used in the query. Then count down the rows in the result set (starting with 0 for the first row) until you get to the number listed in as the Pushed key#. Then, for the corresponding key of reference in the Synergy ISAM file, look at the KEYNUM value for the row.

For our example query, the index number is 2, but what's the KRF? To find out, run the query documented above, replacing *table_name* with VENDORS (use all capital letters). This should return the following:

```
I_TABLE I_NAME              I_TYPE COLUMNS KEYNUM X_NAME     X_POSITION
VENDORS KEY0                U      1       0      VEND_KEY   0
VENDORS $_VTX_TAG_VIX_0001 U      1       1      VEND_KEY   0
VENDORS TAG_KEY             U      2       1      VEND_RTYPE 0
VENDORS TAG_KEY             U      2       1      VEND_KEY   1
```

Now count down the rows, starting from 0 and ending with the number listed as Pushed key#, which is 2. For our example, count

- the KEY0 row as 0.
- the $_VTX_TAG_VIX_0001 row as 1.
- the first TAG_KEY row as 2 (there are two returned rows for TAG_KEY, one for each segment).

Look at the KEYNUM value for TAG_KEY (which is 1). This value is the key of reference in the Synergy ISAM file.

Notes on SORTPAGES

SORTPAGES sets the amount of disk and memory storage used for sort operations for subsequently opened cursors. This overrides the "Total" and "In memory" DSN settings (see "Adding a user or system DSN" on page 8-5).

The *totalpages* argument sets the number of pages to use, and *mempages* is the number of these pages kept in memory. (Pages are 4,096-byte blocks.) *Totalpages* must be greater than or equal to *mempages*. The default value for *totalpages* is 10000, and the default value for *mempages* is 1000. Note that memory for SORTPAGES is allocated for every subsequently opened cursor even if no sort is performed for a cursor.

You can set SORTPAGES any time after connecting, but once it is set, the specified memory is allocated until SORTPAGES is reset or a cursor is closed. An application uses the sum of the memory specified for all concurrently open cursors.

On Windows, **vtxnetd** uses the sum of memory specified by SORTPAGES for every open cursor for every connected application. Though heap memory is freed for reuse when a cursor closes, the memory used for the **vtxnetd** process (reported as private bytes in Task Manger) does not decrease while **vtxnetd** is running.

Restrictions

The following are some of the restrictions to *xf*ODBC's support of SQL. For more, see the Connectivity Series release notes (**REL_CONN.TXT**) and the Synergy/DE restrictions file (**RESTRICT.TXT**).

- The ORDER BY column must be referenced by SELECT column position when accompanied by a GROUP BY clause. See ORDER BY on page B-21.
- GROUP BY is not supported for UNION clauses.
- Columns specified in a GROUP BY clause must be included in the select list and columns specified in the select list must be included in the GROUP BY clause. See GROUP BY on page B-22 for information.
- Only one ORDER BY clause is allowed with a UNION, and this must follow the final SELECT statement. An ORDER BY clause in a UNION applies to the entire result set.

Additionally, *xf*ODBC does *not* support the following:

- Field-level access controls. (Instead, *xf*ODBC enables you to set access levels for tables and groups, which determine the access levels for users. See "Setting Security Levels" on page 8-2.)
- ALTER TABLE
- CREATE INDEX (supported after the initial CREATE TABLE and before the first INSERT, but at no other time)
- DROP INDEX
- Expressions in ORDER BY column
- "FROM" in the UPDATE *[*FROM*]* command
- Multi-table updates
- ROW_NUMBER and the OVER clause

ODBC Reserved Words

The following words are reserved for use in ODBC function calls. To ensure compatibility with drivers that support the core SQL grammar, you should avoid using any of these keywords in column or table names. If you do use a reserved word, you must enclose it in double quotes.

ADD	DELETE	LENGTH	SET
ALL	DESC	LIKE	SKIP
ALTER	DISTINCT	LOCATE	SMALLINT
AND	DOUBLE	LTRIM	SQRT
ANY	DROP	MAX	SUBSTR
AS	DUMP	MIN	SUBSTRING
ASC	ELSE	NCHAR	SUM
ASCII	END	NOT	SYNONYM
AVG	ESCAPE	NOW	SYSDATE
BETWEEN	EXISTS	NULL	TABLE
BY	FLOAT	NUMBER	THEN
CASE	FOR	NUMERIC	TIME
CAST	FROM	NVARCHAR	TIMESTAMP
CHAR	FULL	NVL	TO
CHAR_LENGTH	GRANT	OF	TO_CHAR
CHARACTER	GREATEST	OFF	TO_DATE
CHARACTER_LENGTH	GROUP	ON	TO_NUMBER
CHECK	HAVING	OPTION	TOP
CHR	HOUR	OR	TRANSLATE
COLUMN	IDENTIFIED	ORDER	TRUNC
COMMIT	IFNULL	OUTER	TRUNCATE
CONCAT	IN	PASSWORD	UCASE
CONNECT	INDEX	POSITION	UNION
CONVERT	INITCAP	PRECISION	UNIQUE
COUNT	INNER	PRIMARY	UPDATE
CREATE	INSERT	PRIVILEGES	USER
CURDATE	INSTR	PROCEDURE	VALUES
CURTIME	INT	REAL	VARCHAR
DATABASE	INTEGER	RENAME	VARYING
DATE	INTO	RESOURCE	VIEW
DATETIME	IS	REVOKE	WHEN
DAYNAME	JOIN	RIGHT	WHERE
DBA	KEY	ROLLBACK	WITH
DEC	LCASE	ROUND	WORK
DECIMAL	LEAST	RTRIM	
DECODE	LEFT	SELECT	

Glossary

access key	A “true” key in an ISAM data file. See foreign key.
access level	A number from 0 to 255 that determines who can access a table and how. Each table has an access level, as does each group. A user may view only those tables whose access level is equal to or lower than access level of the group to which the user belongs. Additionally, access levels determine whether data can be modified.
association	An object that represents a relationship between entities in an EDM. For example, if a table has a relation, an EDM will show that relation as an association.
attributes	Characteristics of a repository structure that describe fields, keys, relations, tags, and redisplay formats.
column	Used interchangeably with *field*. For the purposes of *xf*ODBC, a system catalog column is equivalent to a repository field.
connect file	A text file containing datasource (data file location) and dictsource (system catalog path) definitions. This file is used by DBA when opening a system catalog and *xf*ODBC when accessing a Synergy database.
conversion setup file	A text file containing table access levels and data file location, generated by DBA.
data provider	A set of classes that provide ADO.NET access to a data source, as well as data services. For example, the .NET Framework (except version 1.0) includes the .NET Framework Data Provider for ODBC, which provides ADO.NET access to ODBC data sources. See also Synergy/DE Data Provider for .NET.
database	A set of related files created and managed by a database management system (DBMS). Database and file structures are determined by the software application in which it is generated. Although the term *database* is sometimes used to refer to the combination of a software application and a data source, this manual considers data source and database to be analogous.

Database Administrator (DBA)	An *xf*ODBC component that enables you to generate, maintain, customize, and verify the system catalog; create user access; and generate a conversion setup file.
dbcreate	The *xf*ODBC utility used to generate a system catalog and initialize user and group access.
EDM	See entity data model (EDM).
EDMX file	An XML file that defines the conceptual model for an EDM, describes the schema for the database (the storage model), and defines the mapping between the conceptual model and the database.
entity data model (EDM)	A model that represents data as a set of entities and relationships that map to the data source (e.g., a Synergy database).
Entity Framework	A set of technologies that work with .NET Framework to enable object-to-relational mappings. The Entity Framework enables you to generate a model (an EDM) that abstracts data and presents it as a set of objects. You can then manipulate the model as necessary in Visual Studio and write code to work with the objects in the model. This will, in turn, result in queries to the data source (e.g., a Synergy database).
Entity Model Designer	A Visual Studio component that consists of a set of visual tools for creating and editing an EDM.
Entity SQL	A SQL dialect for writing queries against an EDM. The Synergy/DE Data Provider for .NET translates Entity SQL into SQL that's used to directly query the Synergy database.
entity type	A class that represents an entity in the conceptual model for an EDM.
environment setup file	A text file you write to define the data environment variables that are used by *xf*ODBC when locating Synergy data files. The environment setup file is typically used for setting environment variables that are used in the Open filename field of a repository file definition.
field	A record component that contains an individual data element. Used interchangeably with *column*. For the purposes of *xf*ODBC, a repository field is equivalent to a system catalog column.
foreign key	A key used to specify relationships between files but which is not a true key in the data file. (Foreign keys are defined in the repository.)

group (DBA)	One or more users with the same database access level defined in the group catalog.
group (repository)	A structure within a structure, as defined by the Synergy DBL GROUP statement. Fields or other group definitions can be members of a group.
group catalog	The **SODBC_GROUPS.*** files that contain group ID, group name, and user access information.
index	A common method for keeping track of information in a Synergy Database file or a database by storing key information. See key.
ISAM	Indexed Sequential Access Method. A file access method that stores data sequentially while maintaining an index of key fields.
key	A value or field used to identify and locate records in a data file and to define a sequential order in which to process that file. For practical purposes, *key* and *index* are used interchangeably.
key of reference	A key that defines which ISAM file index is used in sequential operations.
LINQ	Language Integrated Query (LINQ) is a .NET Framework component that adds native data querying functionality that's similar to SQL to .NET languages that support it.
LINQ to Entities	A set of operators and a syntax that enables you to query over objects in an EDM. LINQ to Entities is a subset of LINQ to ADO.NET and consists of extensions to .NET languages that support LINQ (Visual C#, Visual Basic, and so forth). The Synergy/DE Data Provider for .NET translates LINQ to Entities queries into SQL that's used to directly query the Synergy database.
literal	A specific, constant value, as opposed to a variable. Both numbers and text can be literal values.
.NET Framework Data Provider for ODBC	An ADO.NET data provider included with the .NET Framework (except version 1) that enables access from .NET applications to ODBC data sources. The Synergy/DE Data Provider for .NET wraps this data provider (and adds support for the Entity Framework and Visual Studio integration).
ODBC	Open Database Connectivity. A standard API for accessing databases in a Windows environment.

ODBC Driver Manager	A dynamic-link library (DLL) provided by Microsoft. The Driver Manager opens and closes ODBC drivers as directed by an ODBC-enabled application.
ODBC-enabled application	A front-end or client application running on Windows that uses an ODBC API to access various types of databases.
record	A data area containing one or more consecutive fields on a related subject, such as a customer record or a file layout.
relation	An attribute of structures that enables you to link the keys of one structure with the keys of other structures.
relative file	A file that consists of a series of fixed-length records referenced by relative position in the file.
repository	The files generated by S/DE Repository. These files describe your actual data files.
Repository	A Synergy application used to define files, structures, tags, fields, and keys for a database; **dbcreate** translates these repository definitions into system catalogs.
runtime	See Synergy Runtime.
scalar property	An EDM entity property that maps to a field in the data source. For example, if you create an EDM for the sample database, the CUSTOMERS table will include a scalar property for each field in the table (CUST_CITY, CUST_CONTACT, and so forth).
segment	A column that is a part or section of an index or key.
sequence	An ADO.NET term for an object that implements the IEnumerable or IQueryable interface.
structure	A record definition or the collection of field and key characteristics for a particular file or files.
Synergy database	A database comprising data files and repository files; the two sets of files together constitute a Synergy database.
Synergy Runtime	The Synergy component required for Synergy applications, such as DBA, to run.

Synergy/DE Data Provider for .NET	An ADO.NET data provider that enables access from .NET applications to Synergy databases. The Synergy/DE Data Provider for .NET includes all of the functionality of the .NET Framework data provider for ODBC (which it wraps), along with support for the Entity Framework, enabling you to create an EDM and then modify and query the EDM (and, by extension, the Synergy database) using LINQ to Entities and Entity SQL. The Synergy/DE Data Provider for .NET also includes a Visual Studio plug-in that enables you to create Visual Studio data connections for Synergy databases.
system catalog	A set of files generated by **dbcreate** and consisting of the individual table, user, group, column, index, segment, tag, and relation catalogs. The system catalog contains a "translation" of the Synergy database in a form that ODBC-enabled applications can understand.
system wide	Used to refer to the level at which some environment variables must be set. A system-wide environment variable is available to all applications running on the system.
table catalog	The **GENESIS_TABLES.*** files that contain file and structure information as well as data table access information.
tag	A set of characters, usually a field or part of a field, that is used for identifying or grouping records in a data structure.
user catalog	The **sodbc_users.*** files that contain user name, password, and group ID information.
USR_DD_FILNAM	A routine that enables you to customize and generalize the data filenames specified in a system catalog.
Visual Studio plug-in	For *xf*ODBC, this refers to a component of the Synergy/DE Data Provider for .NET that enables integration with Visual Studio. For example, it enables you to create Visual Studio data connections to Synergy databases.
***xf*ODBC**	A package of components that enables you to make your Synergy data accessible to third-party applications. *xf*ODBC includes the *xf*ODBC driver and two utilities, the *xf*ODBC Database Administrator (DBA) program and **dbcreate**, used to create system catalogs from your Synergy repository definitions.

Index