

INFORMIX[®]- Data Director[™] for Visual Basic

Data Director Programmer's Guide

Version 3.0
January 1998
Part No. 000-4185

Published by INFORMIX® Press

Informix Software, Inc.
4100 Bohannon Drive
Menlo Park, CA 94025-1032

Copyright © 1981-1998 by Informix Software, Inc. or their subsidiaries, provided that portions may be copyrighted by third parties, as set forth in documentation. All rights reserved.

The following are worldwide trademarks of Informix Software, Inc., or its subsidiaries, provided that portions may be copyrighted by third parties, registered in the United States of America as indicated by “®,” and in numerous other countries worldwide:

INFORMIX®, INFORMIX®-OnLine Dynamic Server™; Data Director™; DataLink™; DataDriver™.

All other marks or symbols are registered trademarks or trademarks of their respective owners.

ACKNOWLEDGMENTS

Documentation Team: Andrea Ames, Caroline Arakelian, Adam Barnett, Kaye Bonney, Lisa Braz, Neal Kaplan

Contributors: Aun Faizullahoy, Farzad Farahbod, Edie Hovermale, Sucheta Joshi, Yonghong Pan, Soumitro Tagore, Kevin Taylor, Denny Wong

To the extent that this software allows the user to store, display, and otherwise manipulate various forms of data, including, without limitation, multimedia content such as photographs, movies, music and other binary large objects (blobs), use of any single blob may potentially infringe upon numerous different third-party intellectual and/or proprietary rights. It is the user's responsibility to avoid infringements of any such third-party rights.

RESTRICTED RIGHTS/SPECIAL LICENSE RIGHTS

Software and documentation acquired with US Government funds are provided with rights as follows: (1) if for civilian agency use, with Restricted Rights as defined in FAR 52.227-19; (2) if for Dept. of Defense use, with rights as restricted by vendor's standard license, unless superseded by negotiated vendor license as prescribed in DFAR 227.7202. Any whole or partial reproduction of software or documentation marked with this legend must reproduce the legend.

Table of Contents

Introduction

About This Manual	3
Organization of This Manual	3
Types of Users	4
Software Dependencies	5
Demonstration Database	5
Documentation Conventions	5
Typographical Conventions	6
Icon Conventions	7
Screen-Illustration Conventions	7
The Data Director Documentation Set	8
Informix Welcomes Your Comments	9

Chapter 1

Getting Started

System Requirements	1-3
Installing Data Director	1-4
Configuring Your Database Connection	1-5
Using the Data Director Connect Utility	1-8
Installing the Apex True Grid	1-9
Installing Acrobat Reader	1-9
Starting Data Director	1-10
Starting the Model Viewer	1-10
The Model Viewer User Interface	1-10
Starting the DataLink Manager	1-13
The DataLink Manager User Interface	1-13
Using the Data Director Tutorials	1-19
Tutorial Objectives	1-19
A Summary of the Lessons	1-20
Starting the Tutorials	1-21
Getting Help	1-22
Accessing On-Line Help	1-22
Accessing Context-Sensitive Help	1-22
Accessing Books On-Line	1-23

Chapter 2	Introducing Data Director	
	Overview of Data Director	2-3
	The Client/Server Challenge.	2-3
	Introducing the Data Director Components.	2-5
	Designing an Application	2-6
	Creating Models	2-7
	Designing the User Interface.	2-8
	Creating DataGroups	2-8
	Creating DataLinks	2-9
	Creating a Navigation Control	2-10
	Adding Application Logic	2-11
	Testing and Debugging.	2-12
	Distributing an Application	2-12
	Running an Application	2-12
	Master-Detail Coordination	2-14
	Cached Data	2-15
	Data Value Synchronization	2-15
	Lookup Data	2-16
	Data Loss Alerts	2-16
 Chapter 3	 Customizing Your Environment	
	Setting DataLink Manager Options	3-3
	Setting Auto Size Options.	3-4
	Setting Status Bar Options	3-6
	Setting Display Options	3-8
	Setting Confirmation Options	3-10
	Setting Source Options.	3-11
	Setting Project Options	3-12
	Setting Trace File Options.	3-12
	Setting Model Options	3-13
	Setting Display Options	3-14
	Setting Filtering Options	3-16
	Setting DataGroup Filtering Options	3-16
	Setting DataLink Filtering Options.	3-18
 Chapter 4	 Working with Models	
	Model Concepts	4-3
	Text and Binary Model Files	4-4
	Reflecting Changes in Your Database	4-5
	Capturing Your Database Structure	4-6
	Planning Which Tables to Import	4-7
	Launching the Model Import Wizard	4-8

Setting Import Options	4-9
Selecting a Database Driver and Data Source	4-12
Logging On to Your Database	4-13
Selecting Import Items	4-14
Updating Columns and Column Attributes	4-15
Verifying Primary Key Columns	4-16
Verifying Inferred Relationships	4-17
Working with Imported Models	4-18
Viewing a Model	4-19
Setting a DBMS	4-20
Connecting to a Database	4-21
Disconnecting from a Database	4-21
Setting Primary Keys	4-21
Refreshing and Saving a Model	4-22
Setting Model Properties	4-23
Setting General Properties	4-24
Working with Tables	4-26
Creating Tables	4-26
Creating Virtual Tables	4-27
Modifying Tables	4-32
Working with Columns	4-32
Creating Columns	4-33
Creating Calculated Columns	4-39
Modifying Columns	4-41
Working with Relationships	4-41
Specifying Master-Detail Records and Lookup Fields.	4-41
Understanding Relationship Types	4-42
Creating Relationships	4-45
Using Multiple-Column Relationships	4-47
Modifying the Key Order	4-48
Modifying Relationships	4-49

Chapter 5 Working with DataGroups

Using DataGroups	5-3
DataGroup Concepts	5-3
Concurrency Control Concepts	5-7
Read Consistency Concepts	5-9
Using Optimistic Concurrency Control.	5-9
Using Pessimistic Concurrency Control	5-14
Creating DataGroups	5-14
Deleting DataGroups	5-16
Recovering Deleted DataGroups	5-18

Setting DataGroup Properties	5-19
Setting Query Properties	5-20
Setting Error-Handling Properties	5-22
Setting Database Properties	5-24
Setting Connection Properties	5-25
Setting Concurrency Properties	5-26
Setting Blob Support Properties.	5-29
Setting Join Properties	5-29
Setting DataGroup Table Properties	5-31
Setting Conditions	5-31
Setting Data Sorting.	5-34
Setting DataGroup Column Properties	5-37
Specifying a Custom Default Value	5-38
Using Multiple DataGroups	5-39

Chapter 6 **Working with DataLinks**

DataLink Concepts	6-3
DataLinks, DataGroups, and Models	6-4
Standard Visual Basic Control Support	6-5
Using Bound Controls with Data Director	6-6
Creating DataLinks	6-6
Creating Standard DataLinks	6-8
Creating DataLinks to Option Buttons	6-10
Creating DataLinks to Control Arrays	6-10
Creating Reference Links	6-11
Replacing DataLinks	6-12
Deleting DataLinks	6-12
Working with Data Paths	6-13
Multiple Data Paths	6-13
Selecting a Data Path	6-15
Modifying Data Paths	6-16
Viewing Data Path Syntax.	6-19
Setting DataLink Properties	6-19
Setting Text Box DataLink Properties	6-19
Setting Option Button DataLink Properties	6-20
Setting Check Box DataLink Properties	6-21
Setting Command Button DataLink Properties	6-23

Chapter 7

Using the Navigation Control

Overview of the Navigation Control	7-5
Adding the Navigation Control to a Project	7-6
Adding the Navigation Control to a Form	7-7
Customizing the Navigation Control	7-7
Setting General Properties	7-7
Setting Control Selection Properties	7-9
Setting Control Appearance Properties.	7-15
Setting Font Properties	7-16
Setting Color Properties	7-17
Initializing an Application	7-18
Using Multiple Navigation Controls	7-19
Writing Event Procedures	7-19
Navigation Control Events	7-20
Intercepting Events	7-21
AfterDataChanged	7-21
AfterDelete	7-22
AfterEnterQBE	7-22
AfterExitQBE	7-23
AfterFetchBlob	7-23
AfterGotoRecord	7-24
AfterLogon	7-24
AfterNewRecord	7-25
AfterQuery	7-26
AfterSave	7-26
BeforeDataChanged	7-27
BeforeDelete	7-28
BeforeEnterQBE	7-28
BeforeExitQBE	7-29
BeforeFetchBlob	7-30
BeforeGotoRecord	7-31
BeforeLogon	7-32
BeforeNewRecord	7-33
BeforeQuery	7-33
BeforeSave	7-34
CustomEvent	7-35
Error	7-36
OptimisticDetectChanged	7-37
Navigation Control Events Quick Reference	7-38
Navigation Control Properties	7-40
BookmarkButtonsVisible Property	7-40
ButtonStyle Property	7-41
Cancel Property.	7-42

DataGroup Property	7-43
DefaultButtonSize Property	7-44
FrameStyle Property	7-45
LayoutMode Property	7-46
ModelFile Property	7-47
NavigationButtonsVisible Property	7-47
OptimisticDetect Property	7-48
PreferSingleGroupPerLine Property	7-49
SpacingHorizontal Property	7-50
SpacingVertical Property	7-50
StatusBackColor Property.	7-51
StatusCaption Property	7-51
StatusForeColor Property	7-53
StatusVisible Property	7-53
Table Property.	7-54
Navigation Control Properties Quick Reference	7-55
Navigation Control Objects	7-57
CustomButton1 Object	7-57
DeleteButton Object	7-58
NewButton Object	7-59
QBEButton Object	7-61
QueryButton Object	7-62
SaveButton Object	7-63
Navigation Control Objects Quick Reference	7-65

Chapter 8 **Multiple-Developer and Multiuser Support**

Building Multiple-Developer Projects	8-3
Using a Source Code Control System	8-4
Checking Out and Checking In Files	8-6
Working with Project Files	8-8
Working with Form Files	8-9
Working with Model Files.	8-10
Working with DataGroup Files	8-11
Working with Data Director Project Files	8-11
Applying Changes to Your Project	8-12
Compiling Your Project	8-12
Building Multiuser Applications	8-13
Multiple-Developer File Status Indicators	8-14

Chapter 9	Running and Testing a Data Director Application	
	Starting a Data Director Application	9-3
	Logging On to Your Database	9-4
	Enabling Smart Logon for a DataGroup	9-4
	Running Your Application	9-5
	Navigating Through a Result Set	9-5
	Setting Record Bookmarks	9-6
	Performing Data Operations	9-6
	Master-Detail Coordination	9-10
	Using Lookup Data	9-10
	Data Loss Alerts	9-12
	Using Error Handling	9-12
	Error Handling in Data Director	9-12
	Standard Integration with Visual Basic.	9-14
	Customizing Runtime Error Messages	9-14
	Using the Trace File	9-15
	Locating and Opening a Trace File	9-15
	Trace File Format	9-16
	Reusing SQL Statements.	9-16
Chapter 10	Distributing an Application	
	Deploying Your Application	10-3
	Including Data Director and Other Required Files	10-3
	Using the Visual Basic Application SetupWizard	10-5
	Installing the Application	10-7
Appendix A	ODBC DataDriver Support	
	Glossary	
	Index	

Introduction

About This Manual	3
Organization of This Manual	3
Types of Users	4
Software Dependencies	5
Demonstration Database	5
Documentation Conventions	5
Typographical Conventions	6
Icon Conventions	7
Screen-Illustration Conventions	7
The Data Director Documentation Set	8
Informix Welcomes Your Comments	9

Read this introduction to the *Data Director Programmer's Guide* for an overview of the information provided in this manual and for an understanding of the conventions used throughout this manual.

About This Manual

This manual provides you with conceptual, task, and reference information to help you maximize your productivity as you use INFORMIX-Data Director for Visual Basic (Data Director).

Organization of This Manual

This guide describes how to use each Data Director feature. This guide includes the following chapters:

- [Chapter 1, “Getting Started,”](#) describes how to install and start Data Director in your Visual Basic environment. This chapter also describes how to use on-line help and how to use the Data Director tutorials to help you get started using Data Director right away.
- [Chapter 2, “Introducing Data Director,”](#) introduces the Data Director components, provides an overview of the Data Director development process, and describes Data Director runtime features.
- [Chapter 3, “Customizing Your Environment,”](#) describes how to set various Data Director options to customize your development environment.
- [Chapter 4, “Working with Models,”](#) introduces the concept of a Model and describes how to capture your database structure. This chapter also provides information about working with Model elements.

- [Chapter 5, “Working with DataGroups,”](#) introduces the concept of a DataGroup and describes how to create and manage DataGroups.
- [Chapter 6, “Working with DataLinks,”](#) introduces the concepts of DataLinks and data paths and describes how to link database columns to Visual Basic controls.
- [Chapter 7, “Using the Navigation Control,”](#) introduces the Data Director Navigation Control and describes how to customize it. This chapter also describes the properties, objects, and events available to the Navigation Control.
- [Chapter 8, “Multiple-Developer and Multiuser Support,”](#) describes how to work with Data Director in a multiple-developer environment and how to build multiuser applications.
- [Chapter 9, “Running and Testing a Data Director Application,”](#) describes the runtime features of a Data Director application. This chapter also describes error handling and using the trace file to assist you in developing your applications.
- [Chapter 10, “Distributing an Application,”](#) describes how to distribute a Data Director application and includes a list of Data Director files required for distribution.
- [Appendix A, “ODBC DataDriver Support,”](#) describes the Data Director ODBC DataDriver and includes information about compliance levels and additional software required to use the ODBC DataDriver.

Types of Users

This manual is written for Visual Basic and C++ application developers who want to use Data Director to write client/server applications in Visual Basic. Before using Data Director, you should have the following information about your software environment:

- A basic understanding of Microsoft Windows 95 or Windows NT Version 3.51 or higher
- A working knowledge of Visual Basic 4.0 or 5.0 (though Data Director is designed to enable even relatively inexperienced developers to produce fully functional database applications)
- A basic understanding of the database against which your application runs

Software Dependencies

To use Data Director as described in this manual, you must:

- be running Windows 95 or Windows NT Version 3.51 or higher.
- have Microsoft's Visual Basic 4.0a or 5.0 installed.
- have access to a database server that supports ODBC.

Although Data Director supports non-Informix databases, INFORMIX-OnLine databases are considered the default. You must contact your ODBC or database vendor for ODBC drivers to connect to non-Informix databases.

Demonstration Database

The Data Director product set provides some demonstration applications along with a small example database named **books** (an INFORMIX-OnLine database) that contains information about a fictitious book distributor. The Data Director Tutorial provides instructions for setting up the demonstration database (see [“The Data Director Documentation Set” on page 8](#)).

All the examples in the Data Director documentation set are based on the **books** database.

Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other volumes in the documentation set.

The following conventions are covered:

- Typographical conventions
- Icon conventions
- Screen-illustration conventions

Typographical Conventions

This manual uses the following standard set of conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.




Convention	Meaning
KEYWORD	All keywords appear in uppercase letters in a serif font.
<i>italics</i>	Within text, new terms and emphasized words appear in italics.
boldface	Identifiers (names of classes, objects, constants, events, functions, program variables, forms, labels, and reports), environment variables, database names, filenames, table names, column names, icons, menu items, command names, and other similar terms appear in boldface.
<code>monospace</code>	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
◆	This symbol indicates the end of product- or platform-specific information.



***Tip:** When you are instructed to “enter” characters or to “execute” a command, immediately press RETURN after the entry. When you are instructed to “type” the text or to “press” other keys, no RETURN is required.*

Icon Conventions

Throughout the documentation, you will find text that is identified by comment icons. Comment icons identify warnings, important notes, or tips. This information is always displayed in *italics*.

Icon	Description
	The <i>warning</i> icon identifies vital instructions, cautions, or critical information.
	The <i>important</i> icon identifies significant information about the feature or operation that is being described.
	The <i>tip</i> icon identifies additional details or shortcuts for the functionality that is being described.

Screen-Illustration Conventions

The illustrations in this manual represent a generic rendition of various windowing environments. The details of dialog boxes, controls, and windows have been deleted or redesigned to provide this generic look. Therefore, the illustrations in this manual depict Data Director a little differently than the way it appears on your screen.

The Data Director Documentation Set

Data Director documentation is provided in a variety of formats:

- **Manuals.** Two on-line manuals are provided in PDF format:
 - The *Data Director Programmer's Guide* provides you with a high-level overview of Data Director and explains how to use each Data Director component. Concepts and procedures for each stage of the design process are included, as well as information about using the runtime environment and distributing a Data Director application.
 - The *Data Director Objects Guide* provides you with a complete reference to the Data Director Objects, including detailed information about the methods and properties available to each object.

To access the on-line manuals, choose **Help→Data Director Books Online** from the Visual Basic menu bar, and then select the book that you want to view. You can also access the on-line manuals from Answers OnLine, on Informix's Web page (www.informix.com).

You can also order printed versions of these manuals.

- **On-line help.** This facility includes overview, task, and reference information about Data Director, as well as reference information about the Data Director Objects and the Data Director Navigation Control. To access on-line help, choose **Help→Data Director** from the Visual Basic menu bar.
- **On-line examples.** The on-line examples illustrate using third-party grid controls and using Data Director Objects in a C++ application. The examples are located in the **Examples** directory within the directory where you installed Data Director. There are three subdirectories within the **Examples** directory, one for each example. See the accompanying **readme.txt** file in each of the **Examples** subdirectories for information about how to use that example.
- **Data Director tutorials.** The tutorials take you through the key steps of developing a client/server application using Data Director. To access the tutorials, choose **Help→Learning Informix Data Director** from the Visual Basic menu bar.

Informix Welcomes Your Comments

Please tell us what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about any corrections or clarifications that you would find useful. Please include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Write to us at the following address:

Informix Software, Inc.
TETC Technical Publications Department
4100 Bohannon Drive
Menlo Park, CA 94025

If you prefer to send electronic mail, our address is:

`doc@informix.com`

We appreciate your feedback.

Getting Started

System Requirements	1-3
Installing Data Director	1-4
Configuring Your Database Connection.	1-5
Using the Data Director Connect Utility	1-8
Installing the Apex True Grid	1-9
Installing Acrobat Reader	1-9
Starting Data Director	1-10
Starting the Model Viewer	1-10
The Model Viewer User Interface	1-10
Model View Pane.	1-11
Table Relationships Pane	1-11
Model Viewer Toolbar	1-12
Starting the DataLink Manager	1-13
The DataLink Manager User Interface	1-13
DataGroup Combo Box	1-15
Model Pane.	1-15
DataGroup Pane	1-15
DataLink Pane.	1-16
DataLink Manager Toolbar	1-16
Model Pane Toolbar Buttons	1-16
DataGroup Pane Toolbar Buttons	1-17
DataLink Pane Toolbar Buttons	1-17
DataLink Manager Status Bar	1-18
Using the Data Director Tutorials	1-19
Tutorial Objectives.	1-19
A Summary of the Lessons	1-20
The Data Director Tutorial.	1-20

The Data Director Objects Tutorial	1-21
Starting the Tutorials	1-21
Getting Help	1-22
Accessing On-Line Help	1-22
Accessing Context-Sensitive Help	1-22
Accessing Books On-Line	1-23

This chapter explains how to install and start Data Director in your Microsoft Windows and Visual Basic environment. It also introduces you to the Data Director user interface and includes information about using the Data Director tutorials.

System Requirements

You can run Data Director on any personal computer with the configurations listed in the following table.

System Component	Requirements
System software	<ul style="list-style-type: none">■ Microsoft Windows 95 or Windows NT Version 3.51 or higher.■ Microsoft Visual Basic 4.0a or 5.0.
Hardware	<ul style="list-style-type: none">■ Personal computer capable of running Microsoft Windows 95 or Windows NT Version 3.51 or higher and Visual Basic 4.0a or 5.0. Informix recommends a fast processor (486/25 or faster) for best performance and maximum developer productivity.■ VGA or higher-resolution monitor (1024 x 768 display resolution recommended).
Memory	<ul style="list-style-type: none">■ 16 MB of RAM (32 MB RAM recommended).
Disk space	<ul style="list-style-type: none">■ Depending on the installation options that you specify, Data Director requires approximately 20-25 MB of available disk space.

You may also need to install a Microsoft Service Pack, depending upon the version of your operating system and Visual Basic software. See the following table for details.

If You Have	You Also Need to Install
Windows 95	Windows 95 Service Pack 1
Windows NT 3.51	Windows NT Service Pack 5
Windows NT 4.0	Windows NT Service Pack 3
Visual Basic 5.0	Visual Basic Service Pack 3

See the Microsoft Web site to download these Service Packs.

Installing Data Director

You can install Data Director on a stand-alone computer or a network server. This section explains the installation procedure.

Important: For the latest installation information, please see *install.txt* located at the root level of your Data Director CD.

Before you install Data Director, you must have Visual Basic installed on your computer. You must also have the Data Director serial number and key to begin the installation procedure.

Important: If you are installing on a Windows NT machine, you must have administrator privileges to install Data Director and your database's client-side connection software and to configure your database connection and set up data sources.

To install Data Director

1. Start Microsoft Windows.
2. Insert the Data Director CD into a CD-ROM drive.
3. In Windows 95 and Windows NT 4.0, choose **Run** from the **Start** menu; in Windows NT 3.51, choose **Run** from the Program Manager **File** menu.



4. Type `D:\Disk1\setup`, where **D** corresponds to the drive from which you are installing.
5. Click **OK** to start the **setup** program.
The Initialization screen appears for a moment. When initialization is complete, the Welcome dialog box appears.
6. To proceed with the installation, click **Next** and follow the instructions on the screen.

If **setup** fails, correct any specified errors and reinstall Data Director.

Configuring Your Database Connection

To use Data Director, you must be able to access a database server. To configure your PC to enable Data Director to connect to your database, you must first install the INFORMIX-Connect, Version 2.0, client-side connection software that ships with your database. The Connect software includes **Setnet32**, which is necessary to complete your connection configuration.

Important: For the latest configuration information, please see ***install.txt*** located at the root level of your Data Director CD.

To determine whether you have the appropriate software installed

1. From the Windows **Start** menu, choose **Find→Files or Folders**.
2. Enter `setnet32` in the **Named** text box, and click **Find Now**.

If you do not find **Setnet32**, you do not have Connect installed.

If you do not have Connect installed, run **setup.exe** from the **Connect** directory located at the root level of the Data Director CD.

After installing Connect, you must perform the following steps:

1. Set up a connection to the server.
2. Select an ODBC driver and create a data source.

Details of these procedures follow in this section. You can find details about Connect and **Setnet32** settings in the *Informix Client Products Installation and Configuration Guide for Microsoft Windows Environments*.





The example values within the procedures assume that you have created the sample **books** database using an INFORMIX-OnLine server and that you are creating a data source for it. Please see your database administrator or documentation for the details of your database setup.

Important: *If you are installing on a Windows NT machine, you must have administrator privileges to install Data Director and your database's client-side connection software and to configure your database connection and set up data sources.*

To set up your database connection

1. Start **Setnet32**.

You can find the **Setnet32** software in %INFORMIXDIR%\bin. It may also be accessed via the **Informix Setnet32** icon in an **Informix** icon group on your **Start** menu.

2. On the **Server Information** page, enter the following information:

- Informix server: *your_server*
- Host name: *your_host*
- Protocol name: *your_protocol*
- Service name: *your_service*

Important: *The protocol and service are dependent upon the server. See your database documentation for details.*

3. Click **Make Default Server**.

4. On the **Host Information** page, enter the following:

- Current host: *your_host*
- User name: *a_username*
- Password option: *password*
- Password: *a_password*

5. Click **Apply**.

6. Click **OK** to define a new host.

7. Click **OK** to complete the task.



Windows 95

Windows NT



8. Edit your PC's **services** file to describe the connection to the server:

- a. Open the **services** file with a text editor.

The **services** file is located in the **\Windows** directory. ♦

The **services** file is located in the **\WINNT\system32\drivers\etc** directory. ♦

- b. Add an entry like the following (which assumes you specified the **turbo2** service) to the **services** file, and save the file:

```
turbo2 1527/tcp # added for ifmx-ODBC
```

If you already have a service name specified for 1527, comment out the existing entry, and add the new one.

***Tip:** If you are connecting to an Informix database, your client-side connection software includes a utility called **ILogin**. You can use **ILogin** to test your database connection and ensure that the information you entered in **Setnet32** is valid.*

To select an ODBC driver and create a data source

1. Choose **Start→Settings→Control Panel**.
2. Open the ODBC control panel.
3. Click **Add** to add a data source.
4. Choose the **INTERSOLV/Informix Informix** driver, and click **Finish**.
5. Enter the following information on the **General** page of the ODBC Informix Driver Setup dialog box:
 - Data source name: *books_source*
 - Database: *books*
6. Enter the following information on the **Connection** page of the ODBC Informix Driver Setup dialog box:
 - Default user name: *a_username*
 - Host name: *your_host*
 - Service name: *your_service*
 - Server name: *your_server*
 - Protocol type: *your_protocol*

***Important:** The user name is the same user name you specified in step 4 of the previous procedure, **"To set up your database connection."** The server, protocol, and service are the same as those you specified in step 2 of that procedure.*





7. Click **OK** to complete the task.

You will now see the *books_source* data source in your data source list.

Tip: You can test your connection using the Data Director Connect utility.

Using the Data Director Connect Utility

The Data Director Connect utility enables you to verify that you can connect to an ODBC data source. This utility also provides version and compatibility information that can be used to verify the integrity of your client connectivity software configuration. You can find this utility in the **Informix Data Director** program group or in the **Informix Data Director 3.0** menu option on the **Start** menu.

To test your ODBC connection with Data Director Connect

1. Choose **Informix Data Director→Data Director Connect** from the **Start** menu, or run **ddconn.exe** from the **Utility** directory where Data Director is installed.

For example, if you use the default directory in Windows 95, the file is in **C:\Program Files\Informix\Data Director\Utility**.

2. Click **Connect**.

The ODBC SQL Data Sources dialog box appears.

3. Double-click the data source to which you want to connect.
4. Log in to the database.

A dialog box will indicate if a successful connection is made. You can now browse through the detailed driver information that is provided.

Tip: If you have connection problems, try uninstalling Data Director and reinstalling using the custom installation and choosing to install the *INTERSOLV* ODBC drivers.



Installing the Apex True Grid

The Data Director software includes the Apex True Grid, which enables you to easily format your database columns into a table-like structure on your Visual Basic form.

To install the Apex True Grid

1. Double-click the **tdgbs32.exe** file in the Apex directory on the Data Director CD-ROM.

The installation program prompts you through the installation process.

Installing Acrobat Reader

The Data Director software includes on-line versions of the printed documentation. To view the on-line documentation, you must have the Adobe Acrobat Reader software installed on your computer.

To install Adobe Acrobat Reader

1. Double-click the **ar32e30.exe** file in the **Acrobat** directory where Data Director is installed.

For example, if you use the default directory in Windows 95, the file is in **C:\Program Files\Informix\Data Director\Acrobat**.

The install program prompts you to create a new directory for the Acrobat reader and guides you through the installation process.

2. Delete the **ar32e30.exe** file contained in the Acrobat subdirectory of the Data Director directory to save space on your hard disk.



Model Viewer
icon

Starting Data Director

The central design components of Data Director are the DataLink Manager and the Model Viewer. This section provides you with basic information about how to start each Data Director component and briefly describes the user interface of each component.

Starting the Model Viewer

You can launch the Model Viewer from the Visual Basic **View** menu or from the DataLink Manager.

To launch the Model Viewer from Visual Basic

1. Choose **View→Model Viewer** from the Visual Basic menu bar.

To launch the Model Viewer from the DataLink Manager

1. Click the **Model Viewer** icon in the lower-left corner of the DataLink Manager window status bar.

The Model Viewer User Interface

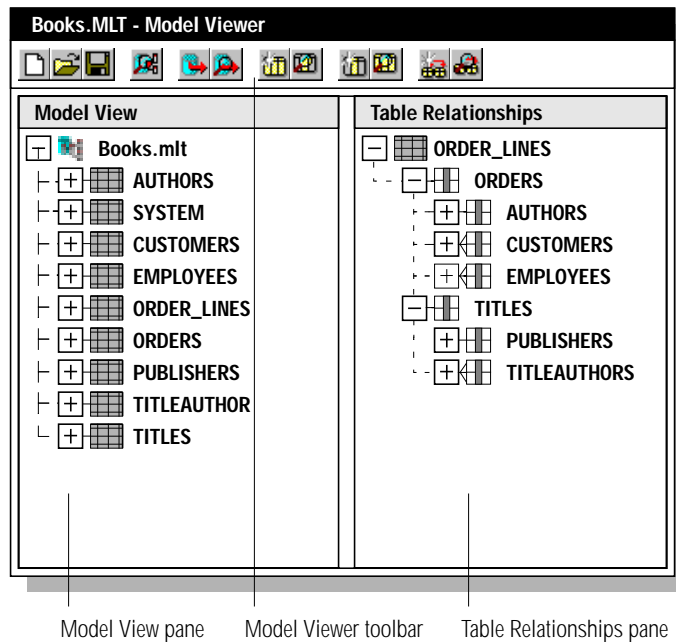
The Model Viewer enables you to work with tables and columns in your database while you are designing an application. You can create new items or modify existing items using the Model Viewer's popup menus.

To access the Model Viewer's popup menus

1. Choose an item in the Model Viewer.
2. Right-click the item.
3. Choose an item from the popup menu.

Figure 1-1 shows the Model Viewer window with each user interface element labeled.

Figure 1-1
Model Viewer
Window



Each Model Viewer user interface element is described in the sections that follow.

Model View Pane

The Model View pane shows all of the database elements in the open Model, including tables, views, routines, table owners, and columns.













Table Relationships Pane

The Table Relationships pane shows all of the relationships between the tables in the open Model.

Model Viewer Toolbar

The Model Viewer toolbar provides easy access to frequently used Model Viewer commands and enables you to easily create, modify, and customize Model elements. All of the commands in the Model Viewer toolbar are also accessible either through the Visual Basic menus or through the popup menus available when Model items are selected.

Each toolbar button is described briefly in the following table.

Toolbar Button	Action	Toolbar Button	Action
	Creates a new Model		Creates a new table
	Opens an existing Model		Sets table properties
	Saves a Model		Creates a new column
	Launches the Model Properties dialog box		Sets column properties
	Launches the Model Import Wizard		Creates a new relationship
	Sets import options		Sets relationship properties

Starting the DataLink Manager

If you selected **Launch Data Director with Visual Basic** during Data Director installation, launching Visual Basic automatically starts the DataLink Manager. If the DataLink Manager does not start, open the Visual Basic Add-In Manager, and ensure that the **Data Director for Visual Basic** option is selected.

Data Director integrates into Visual Basic seamlessly. When you first start Visual Basic, you see all the standard Visual Basic elements with additional windows and menu items for Data Director.

The **Data Director** menu appears in the Visual Basic menu bar. You will also find additional Data Director menu items in many of the Visual Basic menus. In addition to all the familiar Visual Basic components, the DataLink Manager appears below the Visual Basic controls.

The DataLink Manager User Interface

The DataLink Manager enables you to link database columns to Visual Basic controls in an application window. You can create new items or modify existing items using the DataLink Manager's popup menus. You can also float and dock the DataGroup or DataLink panes.

Important: *The DataLink Manager always displays the Model pane, and it cannot be floated or docked.*



To access the DataLink Manager's popup menus

1. Choose an item in the DataLink Manager.
2. Right-click the item.
3. Choose an item from the popup menu.

To float a pane outside the DataLink Manager

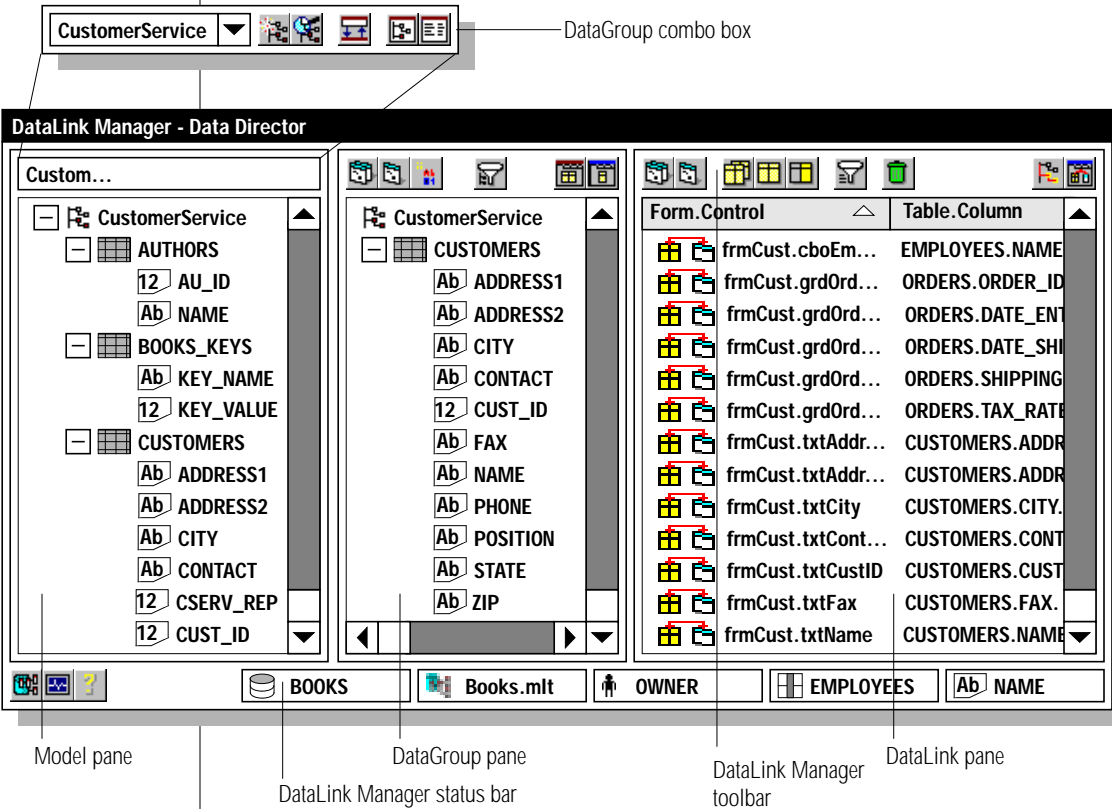
1. Click inside the toolbar area of the pane.
2. Drag and drop the pane anywhere outside the DataLink Manager.

To dock a pane within the DataLink Manager

- 1. Click inside the toolbar area of the pane.
- 2. Drag and drop the pane within the Model pane.

The DataLink Manager window with each user interface element labeled is shown in Figure 1-2.

Figure 1-2
DataLink Manager Window



Each DataLink Manager user interface element is described in the sections that follow.

DataGroup Combo Box

The **DataGroup** combo box lists all of the existing DataGroups for the open project. The first DataGroup in the list is automatically selected when you open an existing project.

To select a different DataGroup, click the down arrow in the combo box, and then select the DataGroup that you want to use to create DataLinks.

Model Pane

The Model pane displays the tables and columns in the Model associated with the current DataGroup. Use the Model pane to create DataLinks from tables and columns in your database to Visual Basic controls.

From the Model pane, you can create DataGroups, open and close the DataGroup or DataLink pane, and modify DataGroup properties.

DataGroup Pane

The DataGroup pane displays all of the tables referenced by DataLinks in the open DataGroup. As you create DataLinks, Data Director displays the table and column associated with the DataLink in the DataGroup pane. You can easily identify the data path of each DataLink using the information displayed in the DataGroup pane.

This information is organized in the DataGroup pane according to the data path that Data Director takes from the master table to the table and column in the DataLink when querying data.

When expanded, the DataGroup pane indicates the type of relationship between tables as you create DataLinks.

From the DataGroup pane, you can set DataGroup table and column properties and filter your view of tables using the filtering options in the toolbar or in the custom filter dialog box.

DataLink Pane

The DataLink pane dynamically displays DataLinks as you create them and identifies each DataLink according to the Visual Basic control and database column to which it is linked. The **Form.Control** column lists the full name of the control in the DataLink and the form on which the control resides. The **Table.Column** column shows which database table and column are linked to the corresponding control.






From the DataLink pane, you can set DataLink properties, delete DataLinks, modify the path of DataLinks, and filter your view of DataLinks using the filtering options in the toolbar or in the custom filter dialog box.

DataLink Manager Toolbar

The DataLink Manager toolbar provides easy access to frequently used DataLink Manager commands and enables you to easily create, modify, and customize DataLink Manager elements. All of the commands in the DataLink Manager toolbar are also accessible either through the Visual Basic menus or through popup menus for selected DataLink Manager items.







Model Pane Toolbar Buttons

Each toolbar button located on the Model pane is described briefly in the following table.

Toolbar Button	Action	Toolbar Button	Action
	Creates a new DataGroup		Shows or hides the DataGroup pane
	Sets DataGroup properties		Shows or hides the DataLink pane
	Enables or disables auto size options for the DataLink Manager		





DataGroup Pane Toolbar Buttons

Each toolbar button located on the DataGroup pane is described briefly in the following table.






Toolbar Button	Action	Toolbar Button	Action
	Shows DataLinks for all forms		Launches DataGroup Tree Filter Options dialog box
	Shows DataLinks for the current form		Shows or hides properties for a table in the DataGroup
	Enables or disables the auto column display		Shows or hides properties for a column in the DataGroup

DataLink Pane Toolbar Buttons

Each toolbar button located on the DataLink pane is described briefly in the following table.

Toolbar Button	Action	Toolbar Button	Action
	Shows DataLinks for all forms		Sets filtering options for the DataLink pane
	Shows DataLinks for the current form		Deletes the selected DataLinks

(1 of 2)




Toolbar Button	Action	Toolbar Button	Action
	Shows DataLinks for all tables		Views or modifies the data path of the selected DataLink
	Shows DataLinks for the current table		Shows or hides DataLink properties
	Shows DataLinks for the current column		

(2 of 2)

DataLink Manager Status Bar

The DataLink Manager status bar provides easy access to frequently used Data Director components and includes additional status information about the items in the DataLink Manager window.

Each status bar button is described briefly in the following table.

Button	Action
	Opens the Model Viewer
	Opens the trace file for the project
	Launches Data Director on-line help

Using the Data Director Tutorials

The Data Director tutorials help you learn Data Director by teaching you the basic concepts and tasks you need to understand to design and run your own application.

To complete these tutorials, you should have a working knowledge of Visual Basic. Specifically, you should be able to:

- open, close, and save files.
- work in a project's Code and Properties windows.
- create all types of controls.

If you need to review some or all of these procedures, see your Microsoft Visual Basic documentation.

Tutorial Objectives

After you complete the tutorials, you will be able to:

- create a Model and work with tables, columns, and relationships in your Model.
- create DataGroups.
- create DataLinks to all types of controls.
- create and use a Navigation Control.
- run your application.
- program with Data Director using the Data Director Objects.

A Summary of the Lessons

There are two tutorials: the Data Director Tutorial and the Data Director Objects Tutorial. The Data Director Tutorial consists of five lessons, and the Data Director Objects Tutorial consists of two lessons. Each lesson describes several tasks and provides step-by-step instructions to help you complete each task.

The tutorial project user interface is designed to allow you to focus on learning about Data Director–specific features. If you would like more information about any pre-coded Visual Basic feature, study the Code windows and refer to your Microsoft Visual Basic documentation.

The Data Director Tutorial

An overview of each lesson follows.

Lesson 1: Creating a Model

In this lesson, you use the Model Import Wizard to import a database to a new Model. You also use the Model Viewer to browse through your new Model and learn how to create a virtual table and a column alias.

Lesson 2: Creating a DataGroup

In this lesson, you use the DataLink Manager to create a DataGroup. You also learn how to set DataGroup properties.

Lesson 3: Creating DataLinks

In this lesson, you use the DataLink Manager to create DataLinks for your application. You also learn how to set DataLink properties and filter your view of DataLinks.

Lesson 4: Creating a Navigation Control

In this lesson, you create a Navigation Control and set Navigation Control properties.

Lesson 5: Running a Data Director Application

In this lesson, you learn how to use several runtime features of Data Director, including Query-by-Example (QBE) and record bookmarks. You also learn about data synchronization and lookup data.

The Data Director Objects Tutorial

An overview of each lesson follows.

Lesson 1: Basics

In this lesson, you learn how to instantiate Data Director Objects, and use them to log on and off a database, query and save data, and navigate through a result set.

Lesson 2: Beyond Basics

In this lesson, you learn how to implement many Data Director features programmatically, such as master-detail coordination, concurrency control, custom SQL statements, and much more.

Starting the Tutorials

The tutorials are on-line help files that guide you through each lesson.

To start the tutorial

1. Choose **Help→Learning INFORMIX-Data Director** from the Visual Basic menu bar.

In the Data Director Tutorial, you need to complete Lesson 1 before you can work on additional lessons—Lessons 2 through 5 are based on a Model that you create in Lesson 1.

To run a lesson, select the lesson number in the Data Director Tutorial and follow the instructions on the screen.

Getting Help

On-line help is provided for Data Director tasks, overview materials, reference materials, programmatic elements, and user interface elements.

You can also view the *Data Director Programmer's Guide* and [Data Director Objects Guide](#) on-line using Adobe Acrobat Reader.

Accessing On-Line Help

You can obtain help on Data Director tasks, reference materials, and programmatic elements by using the Data Director and Data Director Objects on-line help.

To access on-line help

1. Choose **Help**→**Data Director** from the Visual Basic menu bar.
2. Select the help file that you want to view.
3. Click a topic from the help system to view help about that topic.

Accessing Context-Sensitive Help

You can obtain help on user interface elements by:

- using the **Help** button at the top of a dialog box.
- pressing F1 in a dialog box or window.

To use the Help button

1. Click the **Help** button at the top of a dialog box.
Your cursor changes to a question mark.
2. Click an item in a dialog box to see a brief description of the item.



Help button

Accessing Books On-Line

You can view on-line versions of the *Data Director Programmer's Guide* and the *Data Director Objects Guide* using Adobe Acrobat Reader. These books were automatically installed during the Data Director installation, unless you unchecked the **Data Director Online Books** check box.

To access books on-line

1. Choose **Help→Data Director Books Online** from the Visual Basic menu bar.
2. Select the on-line book that you want to view.

You must have Adobe Acrobat installed before you can view the on-line books. For installation instructions, see [“Installing Acrobat Reader” on page 1-9](#).

Introducing Data Director

Overview of Data Director	2-3
The Client/Server Challenge	2-3
Accessing Data with Data Director.	2-4
Providing Data Management with Data Director	2-4
Introducing the Data Director Components	2-5
Designing an Application	2-6
Creating Models	2-7
Designing the User Interface	2-8
Creating DataGroups.	2-8
Creating DataLinks	2-9
Custom Control Support	2-10
Creating a Navigation Control	2-10
Adding Application Logic	2-11
Testing and Debugging	2-12
Distributing an Application	2-12
Running an Application	2-12
Master-Detail Coordination	2-14
Cached Data.	2-15
Data Value Synchronization	2-15
Lookup Data.	2-16
Data Loss Alerts	2-16

Data Director is a powerful extension to Visual Basic that enables client/server applications to access data stored in back-end databases and provides applications with key data management features.

This chapter introduces concepts such as data access and data management, introduces the Data Director components, and provides an overview of the Data Director design-time and runtime environment.

Overview of Data Director

Data Director is a client/server tool set that enhances the Visual Basic development environment with *data access* and *data management* functionality. Data Director provides you with a robust environment that helps you tackle the design and implementation challenges you face as a client/server application developer.

The Client/Server Challenge

From a user's perspective, a good client/server application should provide easy and intuitive ways to find data, modify it, delete it, and save it. From a developer's perspective, a good application development tool should automate the most code-intensive aspects of client/server functionality and enable the developer to focus on user-centric features such as ease-of-use and flexibility.

One of the biggest challenges involved in designing and implementing a client/server application is extracting information from a database—whether that database is a local database or a corporate-wide SQL database—and putting it into a front-end client. Known as *data access*, this interchange between front-end client and back-end server typically requires a lot of your development time.

Almost as big a challenge as data access is data management. Once data is displayed in your client, application users must have a way to manipulate and navigate that data. Known as *data management*, the features that enable users to view, modify, and save data are not often easy to incorporate into your application.

Data Director takes care of data access and data management for you so that you can provide users with easy-to-use, robust applications.

Accessing Data with Data Director

Data Director provides data access features without the need for extensive programming, including:

- managing database connections.
- managing database cursors.
- providing logon management and logon security.
- providing data sorting and data query conditions.
- querying data from a database.

Data Director generates SQL statements and provides your front-end client with back-end independence. You do not need to understand the complexities of SQL or proprietary SQL languages of different databases because Data Director manages all communication with your back-end server.

Providing Data Management with Data Director

Data management is the functionality in an application that enables the coordination and management of data. Data Director provides data management features without the need for extensive programming, including:

- data operations, such as saving and updating data.
- master-detail synchronization and data value synchronization across multiple application windows.
- record navigation and the ability to set bookmarks.
- transaction management control and concurrency control.
- automated data caching.

Data Director automates data management without increasing the code overhead of your application and without causing any degradation of application speed or performance. Any application that you develop with Data Director can include all the data management features mentioned in the preceding list, and more—but Data Director requires no additional code.

Introducing the Data Director Components

The Data Director components help you create client/server applications using Visual Basic with a minimum of development effort.

This section briefly describes each of the Data Director components:

- **Model Import Wizard.** The Model Import Wizard enables you to import the structure of your database to a Model. You can import database information such as tables, views, routines, and primary keys.
- **Model Viewer.** The Model Viewer enables you to view imported Model information, create new Model elements, and extend or modify the functionality of the information imported from a database. For example, you can create virtual tables to pass custom SQL statements directly through Data Director to your database. You can also create calculated columns and extend imported column information to include default values for columns.
- **DataLink Manager.** The DataLink Manager is a visual, drag-and-drop interface for associating Visual Basic controls with database columns at design time. The DataLink Manager also enables you to organize the data in your application into meaningful groups of data for different business processes.
- **Navigation Control.** The Navigation Control is a custom control that enables application users to navigate through a result set, insert and browse bookmarks, query data, and insert, delete, and modify records through a visual interface. You can also use a Navigation Control to initialize an application and trap data events that occur in a running application.

- **Data Director Objects.** Data Director provides a complete collection of objects, methods, and properties that enable developers to access and manipulate data returned by Data Director. The Data Director Objects feature set provides programming power and flexibility to developers.
- **ODBC DataDriver.** The Data Director Open Database Connectivity (ODBC) DataDriver enables Data Director to communicate with your database. The ODBC DataDriver provides database access and automates connectivity to many commercially available databases.
- **Trace file.** The trace file enables you to view all SQL statements, database transactions, Data Director error messages, and database messages that occur in a Data Director application.

Designing an Application

As a developer, you are often called on to organize a company's data, present it in an easy-to-use application that is not hindered by data access limitations, and provide users with runtime features that make data easy to view and to keep up-to-date. Data Director aids you in each of these areas of client/server application development.

With Data Director, the primary design-time tasks are centered around building your user interface, deciding which data your application will query and where the data should be displayed, and customizing your application.

This section explains how you use Data Director to build an application. To introduce how all the pieces of the Data Director tool set work together, this section lists the tasks you would typically complete during the application development process and identifies which Data Director component to use for each task.

Creating Models

The first step in creating a client/server application with Data Director is to create a Model. Data Director provides you with a Model Import Wizard that enables you to easily capture your database structure and create a Model for each database that your application will access. You can import any existing database to a Model.

Models are the starting point for any Data Director application, containing all the database structure information used by Data Director to implement data access and data management. Specifically, a Model stores information about the following components of a database:

- Table names
- Column names
- Relationships
- Table owners
- Column sizes
- Column data types
- Primary key attributes
- Foreign keys

Figure 2-1 shows the relationship between a database and a Model.

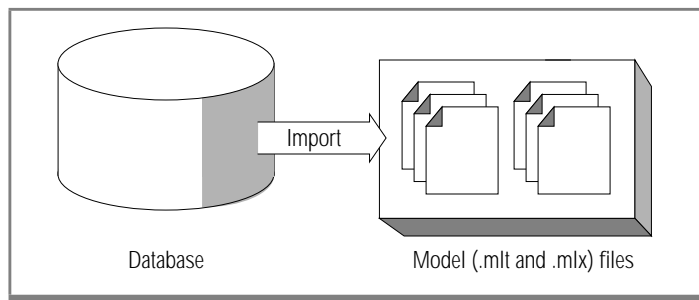


Figure 2-1
*Importing a
Database to a
Model*

At design time, Data Director reads the text (**.mlt**) Model file that provides information about your database tables and columns.

At runtime, Data Director reads the binary (**.mlx**) Model file to access your database and query the appropriate tables.

Once you import a database to a Model, you can access database tables and columns in the Visual Basic development environment using the DataLink Manager. Models are discussed in greater detail in [Chapter 4, “Working with Models.”](#)

Designing the User Interface

When you design your user interface, you implement the main business process that your application will manage. Business processes are essentially determined by the specific data you are interested in working with. The following business processes are typical examples:

- Order entry
- Customer information
- Human resources

Creating DataGroups

Once you create Models, you can create DataGroups that group and organize the data in your Models into different business processes. For example, you can create one DataGroup to access data in a **CUSTOMERS** table and another to access information in an **ORDERS** table. Each DataGroup provides you with a different view into your application’s data and also coordinates data across multiple application windows.

You create DataGroups using the DataLink Manager. When you create a DataGroup, you assign a Model and a master table from the Model to the DataGroup. The master table tells Data Director which table in your database to start queries from when querying data.

DataGroups also contain additional information about your application, such as table-sorting and query conditions, concurrency control settings, error-handling options, and much more.

[Figure 2-2](#) shows the Create DataGroup dialog box.

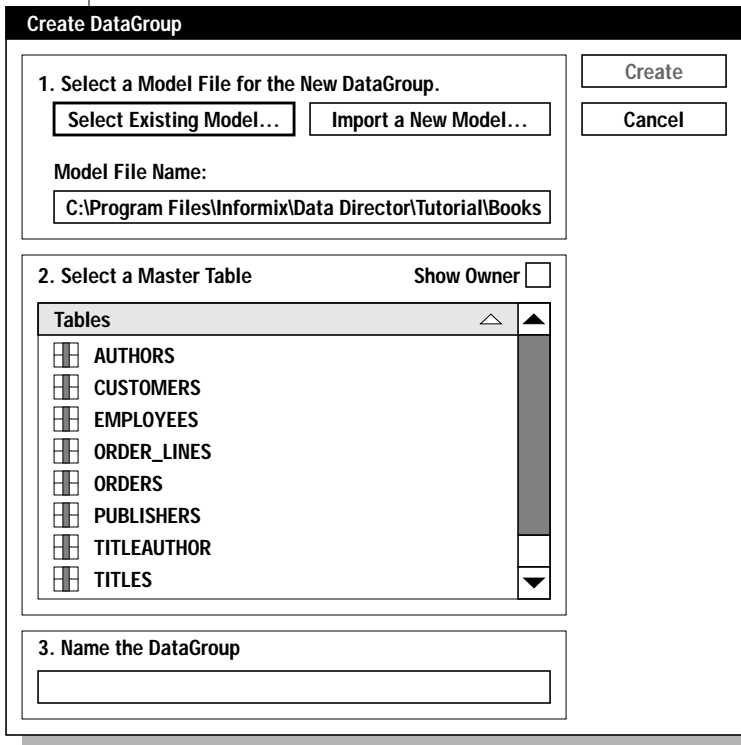


Figure 2-2
Creating a DataGroup

DataGroups are described in greater detail in [Chapter 5, “Working with DataGroups.”](#)

Creating DataLinks

After you create DataGroups, you create DataLinks. DataLinks are the connections you create between database columns and Visual Basic controls. A DataLink coordinates data between a database and your application and tells Data Director where data should be displayed in your application.

The DataLink Manager enables you to easily create DataLinks with an intuitive drag-and-drop interface. Simply drag and drop database columns to Visual Basic controls to create DataLinks.

At runtime, Data Director displays data in the appropriate control in your application, based on the information in each DataLink. For example, if you link the **NAME** column from your **CUSTOMERS** table to the **Customer Name** text box in the Customer Service form, your application automatically displays the customer name in the **Customer Name** field at runtime.

DataLinks also contain special properties for certain types of Visual Basic controls. For example, DataLink properties for an option button group allow you to specify a data value for each button in the group. When a user clicks an option button and saves the changes, Data Director saves to the database the value of the option button based on the properties of the DataLink.

DataLinks are discussed in greater detail in [Chapter 6, “Working with DataLinks.”](#)

Custom Control Support

Data Director provides support for the standard Visual Basic-bound controls. Simply create DataLinks to the bound controls—Data Director displays data in the control without any additional coding on your part.

Data Director also provides support for many popular third-party controls, letting you preserve your investment in the third-party controls that you currently use.

Creating a Navigation Control

The Navigation Control is a custom control that enables users to navigate through a result set, insert and browse bookmarks, query data, and insert, delete, and modify records through a visual interface.

To create a Navigation Control, simply double-click the **Navigation Control** button in the Visual Basic Toolbox.

[Figure 2-3](#) shows an application window with a Navigation Control.



Navigation Control
icon

Customer Service

Customer Information

Name: Cust ID:

Address: Phone:

FAX:

Contact:

CS Rep: Position:

Navigation Control

Order Information

Figure 2-3
Customer Service
Form with a
Navigation Control

The Navigation Control features numerous properties and events that enable you to customize the user interface of the Navigation Control and trap data events in a running application. Navigation Controls are described in greater detail in [Chapter 7, “Using the Navigation Control.”](#)

Adding Application Logic

After you design your application, you can add any required application logic to enhance the functionality of your application. For example, you might want to add application logic to handle errors that occur in a Data Director application, trap data events, or use the Data Director Objects to perform other tasks in your application programmatically. Error handling is discussed in greater detail in [“Using Error Handling” on page 9-12](#). For information about the Data Director Objects programming interface, see the [Data Director Objects Guide](#).

Testing and Debugging

After you have finished building your application, you can compile and run it. An application that uses Data Director compiles and runs the same way as an application developed with Visual Basic alone. Simply enter run mode to run the application.

You can use the Data Director trace file to view the SQL statements generated for your application, as well as track any database or Data Director error messages that occur. You can also copy SQL statements from the trace file for use in other applications. Detailed instructions are provided in [Chapter 9, “Running and Testing a Data Director Application.”](#)

Distributing an Application

Data Director’s runtime services are provided by Data Director’s dynamic link libraries. If you decide to distribute your application in an executable form, simply distribute the appropriate Data Director files with your application. You can easily distribute a Data Director application using the Visual Basic Setup Wizard. For more information, see [Chapter 10, “Distributing an Application.”](#)

Running an Application

You will quickly realize the benefits of using Data Director to develop your application when you run an application developed using Data Director.

When you run an application, Data Director determines which SQL statements to generate and which data to display in an application window, based on the information in your project files.

Data Director generates all necessary SQL statements while your back-end server executes each SQL statement and returns the appropriate data to the front-end client. Data Director automatically caches data that cannot be immediately displayed.

The client application displays the result set and provides users with features that enable them to navigate data and search for specific records. If your application includes a Navigation Control, users can query data, browse through a result set, enter QBE search criteria, set bookmarks on records, create new records, and modify or delete existing records automatically.

Data Director uses information about the Models and DataLinks in your project to connect to your database, coordinate runtime behavior, and handle user events, such as querying data, in the following way (shown in [Figure 2-4](#)):

1. When you start an application, Data Director first reads the associated project files.
2. After Data Director connects to your database, it generates the appropriate SQL statements and then retrieves the requested data from your database.
3. Data Director constructs SQL statements and sends them to your database in response to user events such as modifying and inserting data.
4. Data Director fetches data on a need-to-display basis. When a user navigates to a specific record, the appropriate data is queried and cached in Data Director's internal buffers.
5. Data Director makes sure that queried data is displayed in the correct fields, provides data value synchronization, and maintains master-detail coordination across your application.

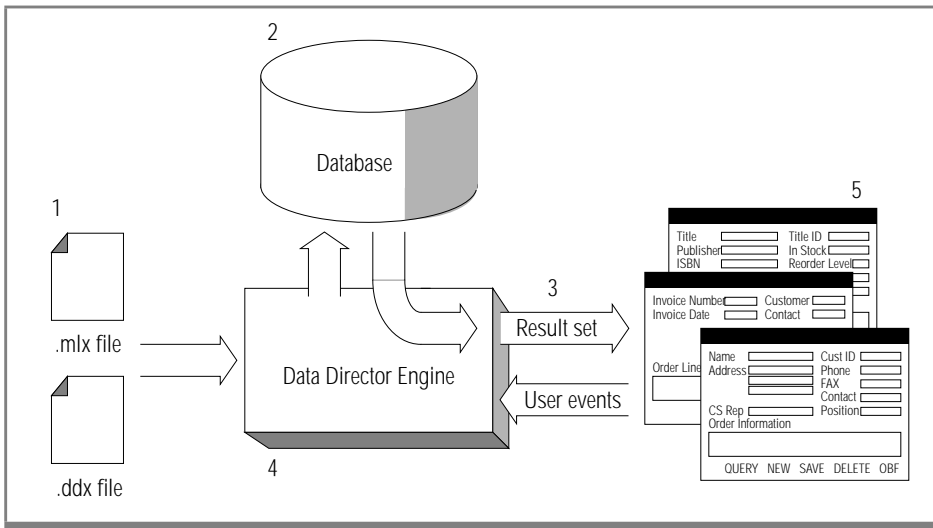


Figure 2-4
*Data Director
Automates Data
Access and Data
Management*

Many data access and data management features are in place as soon as you create DataLinks and run your application. In fact, there are a few features that can be observed about Data Director, based on the data shown in a running application in the development environment.

The following sections describe the runtime features of a Data Director application—specifically, the Customer Service portion—built against the **books** demonstration database that you received with Data Director. The application accesses customer and order information from the **books** database.

Master-Detail Coordination

When an end user queries data for tables that are related to each other through a one-to-many relationship, Data Director automatically queries all detail records for the master table.

The Visual Basic **DBGRID** displays all the existing orders for the current customer. When you navigate to a different customer, the orders for that customer appear. Data Director first queries the **CUSTOMERS** table. Based on the value of the **CUST_ID** column, Data Director queries records in the **ORDERS** table for the current customer.

Based on the project files in use, Data Director automatically knows that **CUSTOMERS** is the master query table and that a one-to-many relationship exists from the master **CUSTOMERS** table to the detail **ORDERS** table. Data Director recognizes this relationship through the **CUST_ID** column: This is the primary key in the **CUSTOMERS** table and a foreign key in the **ORDERS** table.

Cached Data

Data Director caches data until a user makes a request to view or modify the data. For example, clicking **Review Order** in the Customers window displays the Review Orders window. The information in the Review Orders window is coordinated and synchronized with the information in the Customers window.

Data Director enables you to display the same data in multiple controls and queries the data only once—whether that data is displayed in one or in several application windows—and holds all queried data in a cache distinct from any of your application windows.

Data Value Synchronization

When you modify data that is displayed in multiple controls, Data Director synchronizes the data in all controls across application windows. For example, if you modify a customer name, this change occurs in all controls that display that customer's name.

Lookup Data

The **CS Rep** list box (shown in [Figure 2-3 on page 2-11](#)) displays a list of employee names that users can choose from.

Data Director automatically knows to execute a blind query on the **EMPLOYEES** table based on the many-to-one relationship from the **CUSTOMERS** table to the **EMPLOYEES** table when you run the application.

Data Director also automatically maps each **EMP_ID** value from the **EMPLOYEES** table to the appropriate employee name, based on the value of the **Cserve_REP** column in the **CUSTOMERS** table.

Data Loss Alerts

If you make changes to the current order and then navigate to another customer before saving your changes, Data Director displays an alert dialog box asking if you would like to save the data before proceeding.

Customizing Your Environment

Setting DataLink Manager Options	3-3
Setting Auto Size Options	3-4
Setting Status Bar Options	3-6
Setting Display Options	3-8
Setting Confirmation Options	3-10
Setting Source Options	3-11
Setting Project Options	3-12
Setting Trace File Options	3-12
Setting Model Options	3-13
Setting Display Options	3-14
Setting Filtering Options	3-16
Setting DataGroup Filtering Options	3-16
Setting DataLink Filtering Options	3-18

Y

ou can customize your Data Director environment by setting DataLink Manager options, Model options, filtering options, and runtime options. This chapter describes the various options that you can set to customize your Data Director environment.

Setting DataLink Manager Options

There are several characteristics of the DataLink Manager environment that you can customize. For example, you can specify how you want to display items in the DataLink Manager.

If you modify the default settings for the DataLink Manager, your changes take effect when you close the DataLink Manager Options dialog box, and they persist until the next time you modify them.

To view the DataLink Manager Options dialog box

1. Choose **Data Director→Options→DataLink Manager** from the Visual Basic menu bar.

Setting Auto Size Options

Auto size options enable you to specify when Data Director automatically resizes the DataLink Manager. Auto size options are useful when you need to maximize your available screen space when designing an application.

The **Auto Size** page is shown in [Figure 3-1](#).

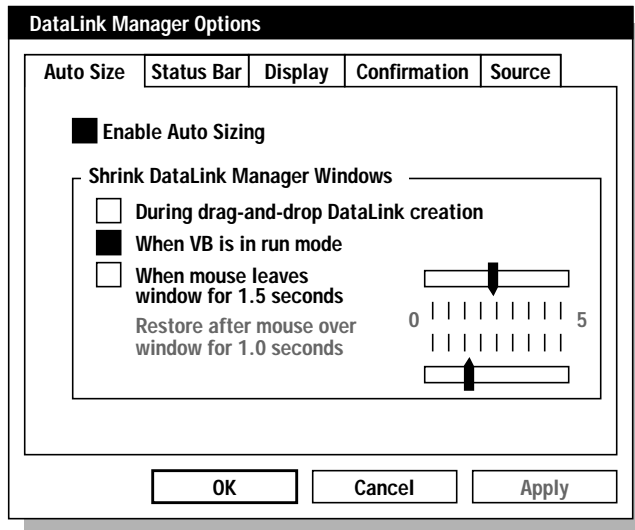


Figure 3-1
*Setting Auto
Size Options*

The following table describes the auto size options that you can set.

Option	Description
Enable Auto Sizing	This setting indicates whether to automatically size the DataLink Manager windows during design time or runtime.
Shrink DataLink Manager Windows: During drag-and-drop DataLink creation	Use this option to shrink the DataLink Manager when creating DataLinks at design time. As soon as you create a DataLink, the DataLink Manager resumes its original position.
Shrink DataLink Manager Windows: When VB is in run mode	Use this option to shrink the DataLink Manager when Visual Basic is in run mode. As soon as you exit run mode, the DataLink Manager resumes its original position.
Shrink DataLink Manager Windows: When mouse leaves window for x seconds	Use this option to shrink the DataLink Manager when the mouse leaves the window for a set amount of time. Use the slider to set the delay before the DataLink Manager resizes.
Restore after mouse over window for x seconds	Use this option to restore the DataLink Manager after a set amount of time when the mouse is over any DataLink Manager window. Use the slider to set the delay before the DataLink Manager resumes its original position.

Setting Status Bar Options

Status bar options enable you to display a status bar in the DataLink Manager window and specify which items to display in the status bar.

The **Status Bar** page is shown in [Figure 3-2](#).

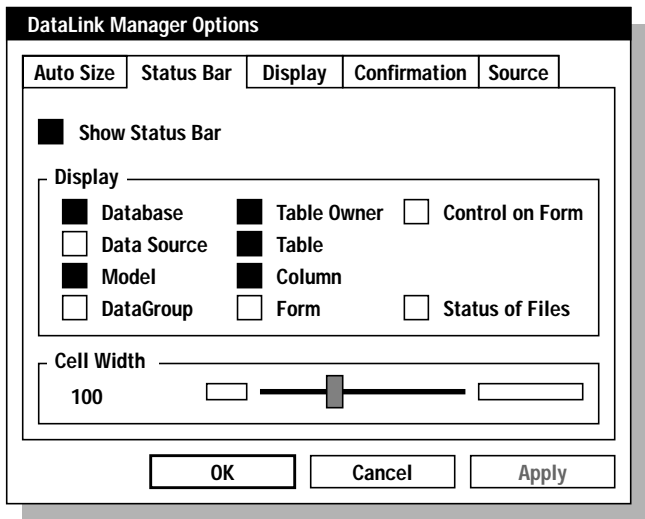


Figure 3-2
Setting Status Bar Options

The following table describes the status bar options that you can set.

Option	Description
Show Status Bar	Use this option to display the status bar at the bottom of the DataLink Manager window. The status bar includes information about the items in the DataLink Manager window, as well as additional buttons that enable you to launch the Model Viewer, open the trace file for a project, or launch Data Director help.
Display: Database	Use this option to display the database name associated with the current DataGroup.
Display: Data Source	Use this option to display the data source associated with the current DataGroup.

(1 of 2)

Option	Description
Display: Model	Use this option to display the Model associated with the current DataGroup.
Display: DataGroup	Use this option to display information about the DataGroup that is currently open.
Display: Table Owner	Use this option to display the table owner for the currently selected table.
Display: Table	Use this option to display information about the currently selected table.
Display: Column	Use this option to display information about the currently selected column.
Display: Form	Use this option to display information about the form that is currently open.
Display: Control on Form	Use this option to display information about the control that is currently selected on the form.
Display: Status of Files	Use this option to display information about the status of the current Visual Basic project file, form, and DataGroup. For example, if a project file is checked out, a closed lock icon appears next to the VBP project file indicator.
Cell Width	Use the slider bar to decrease or increase the pane size of items in the status bar.

(2 of 2)

Setting Display Options

Display options enable you to configure the information that Data Director displays in the DataLink Manager.

The **Display** page is shown in [Figure 3-3](#).

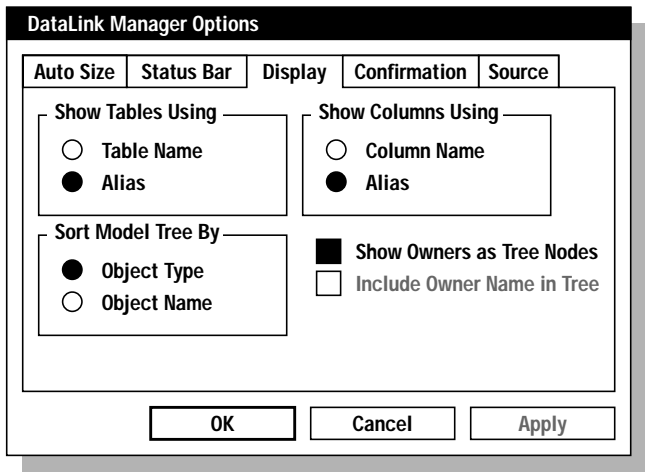


Figure 3-3
Setting Display Options

The following table describes the display options that you can set.

Option	Description
Show Tables Using: Table Name	Use this option to display table names in the DataLink Manager by the name of the table.
Show Tables Using: Alias	Use this option to display table names in the DataLink Manager by the table alias.
Show Columns Using: Column Name	Use this option to display column names in the DataLink Manager by the name of the column.
Show Columns Using: Alias	Use this option to display column names in the DataLink Manager by the column alias.
Sort Tree By: Object Type	Use this option to sort items in the DataLink Manager by the object type. For example, sort all objects by tables, then views, then stored procedures.

(1 of 2)

Option	Description
Sort Tree By: Object Name	Use this option to sort items in the DataLink Manager by the object name. For example, sort all tables, views, and stored procedures by the name of the object.
Show Owners as Tree Nodes	This option specifies whether you want to display table names and table owner names in the DataLink Manager. If you use this option, Data Director organizes objects by owner. If you do not use this option but do use Include Owner Name in Tree , Data Director includes the owner name for a table. If you do not select either option, you will see table names only.
Include Owner Name in Tree	Use this option if you want Data Director to include the owner name for a table in the DataLink Manager. This option works with the Show Owners as Tree Nodes check box.

(2 of 2)

Setting Confirmation Options

Confirmation options enable you to confirm the replacement or deletion of a DataLink during design time.

The **Confirmation** page is shown in [Figure 3-4](#).

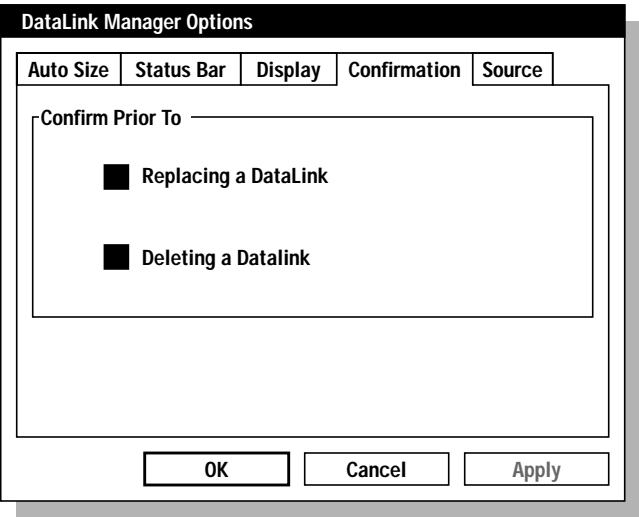


Figure 3-4
Setting Confirmation Options

The following table describes the confirmation options that you can set.

Option	Description
Confirm Prior To: Replacing a DataLink	Use this option if you want Data Director to prompt you to confirm the replacement of a DataLink.
Confirm Prior To: Deleting a DataLink	Use this option if you want Data Director to prompt you to confirm the deletion of a DataLink.

Setting Source Options

Source options enable you to configure how Data Director works with your source code control system.

The **Source** page is shown in [Figure 3-5](#).

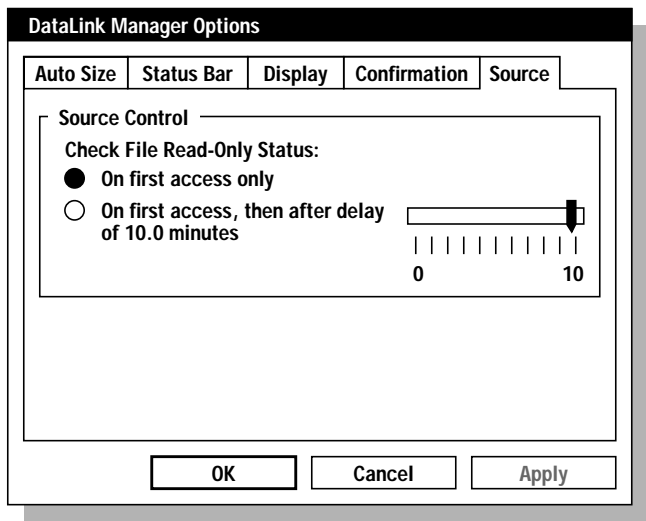


Figure 3-5
Setting Source Options

The following table describes the source options that you can set.

Option	Description
Source Control: Check File Read-Only Status	The buttons in this group enable you to specify how frequently Data Director checks the read-only status of project files.
On first access only	This option specifies that Data Director checks the read-only status of a file the first time you access it.
On first access, then after a set delay	This option specifies that Data Director checks the read-only status of a file the first time you access it and then checks it again if you access it after the set delay time.

Setting Project Options

Project options enable you to configure your project environment and the runtime environment of Data Director.

If you modify the default settings for a project, your changes take effect upon closing the Project Options dialog box and persist until the next time you modify them.

To view the Project Options dialog box

1. Choose **Data Director→Options→Project** from the Visual Basic menu bar.

Setting Trace File Options

Trace file options allow you to enable the trace file functionality in Data Director and specify additional information for a trace file (see [“Using the Trace File” on page 9-15](#)).

The Trace page is shown in [Figure 3-6](#).

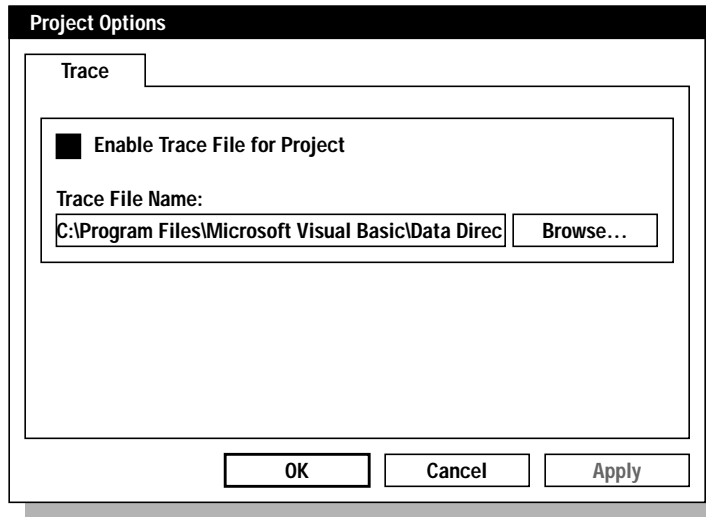


Figure 3-6
*Setting Trace
File Options*

The following table describes the trace file options that you can set.

Option	Description
Enable Trace File for Project	This option enables the trace file functionality for a project.
Trace File Name	<p>Specify a filename (including path) for the trace file. If you do not specify a filename, Data Director provides a default name based on the corresponding project name.</p> <p>For example, a project called books.vbp has an associated trace file named books.trc by default.</p> <p>If you do not specify a file location, Data Director saves the trace file in your temp directory as specified by your TEMP environment variable specified in your autoexec.bat file.</p>



Warning: When specifying the name of the trace file and its location, consider that if a project and its log file are stored in the same directory, multiple users might unintentionally overwrite one another's trace file entries.

Setting Model Options

There are several characteristics of Models that you can customize. For example, you can specify whether you want to be prompted to confirm the deletion of Model elements.

If you modify the default settings for a Model, your changes take effect upon closing the Setting Model Options dialog box and persist until the next time you modify them.

To view the Model Options dialog box

1. Choose **Data Director→Options→Model** from the Visual Basic menu bar.

Important: A Model must be open in the Model Viewer. If it is not, you cannot choose the **Model** menu item.





Tip: For information about setting import options before capturing a database structure, see [“Setting Import Options” on page 4-9](#).

Setting Display Options

Display options enable you to configure the information that Data Director displays in the Model Viewer window.

The **Display** page is shown in [Figure 3-7](#).

The screenshot shows a dialog box titled "Model Options - Books.mlt" with two tabs: "Import" and "Display". The "Display" tab is selected. It contains several settings:

- Show Tables Using:** Radio buttons for "Table Name" (unselected) and "Alias" (selected).
- Show Columns Using:** Radio buttons for "Column Name" (unselected) and "Alias" (selected).
- Sort Tree By:** Radio buttons for "Object Type" (selected) and "Object Name" (unselected).
- Show Owners as Tree Nodes:** A checked checkbox.
- Include Owner Name in Tree:** An unchecked checkbox.
- Confirm Delete:** A checked checkbox.
- Set Current Settings as Default:** An unchecked checkbox.

At the bottom are three buttons: "OK", "Cancel", and "Apply".

Figure 3-7
Setting Display Options

The following table describes the display options that you can view or set.

Option	Description
Show Tables Using: Table Name	Use this option to display table names in the Model Viewer by the name of the table.
Show Tables Using: Alias	Use this option to display table names in the Model Viewer by the table alias.
Show Columns Using: Column Name	Use this option to display column names in the Model Viewer by the name of the column.
Show Columns Using: Alias	Use this option to display column names in the Model Viewer by the column alias.
Sort Tree By: Object Type	Use this option to sort items in the Model Viewer by the object type. For example, sort all objects by tables, then views, then routines.
Sort Tree By: Object Name	Use this option to sort items in the Model Viewer window by the object name. For example, sort all tables, views, and routines by the name of the object.
Show Owners as Tree Nodes	This option specifies whether you want to display table names and table owner names in the Model Viewer. If you use this option, Data Director organizes objects by owner. If you do not use this option but do use Include Owner Name in Tree , Data Director includes the owner name for a table. If you do not select either option, you will see table names only.
Include Owner Name in Tree	Use this option if you want Data Director to include the owner name for a table in the Model Viewer. This option works with the Show Owners as Tree Nodes check box.
Confirm Delete	Use this option if you want Data Director to prompt you to confirm the deletion of Model elements.
Set Current Settings as Default	Use this option to save the current Model options for future Models.

Setting Filtering Options

Filtering options enable you to customize your view of the information that appears in the DataGroup pane and DataLink pane in the DataLink Manager. Setting filtering options enables you to easily locate specific information in your project. For example, if you filter DataLink information by a specific form, you can quickly locate the DataLinks to controls on that form.

Setting DataGroup Filtering Options

DataGroup filtering options enable you to customize your view of the information that appears in the DataGroup pane of the DataLink Manager. For example, you can display only tables that are linked to controls on the current form.

To set DataGroup filtering options

1. Click the **Filtering Options** button in the DataGroup pane.
2. Click the filtering options you want to use.

The DataGroup Tree Filter Options dialog box is shown in [Figure 3-8](#).



Filtering Options
button

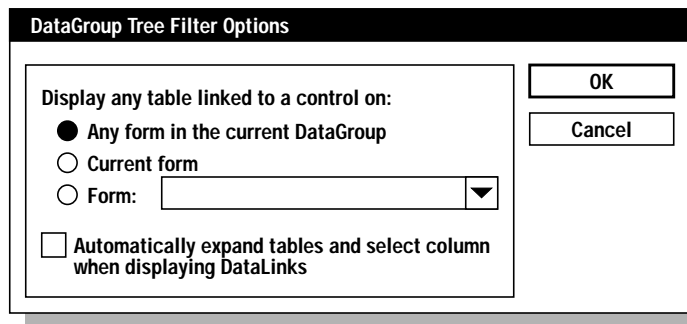


Figure 3-8
*Setting DataGroup
Filtering Options*

The following table describes the DataGroup filtering options you can set.

Option	Description
Any form in the current DataGroup	Use this option to display tables containing DataLinks that are linked to any form in the current DataGroup in the DataGroup pane.
Current form	Use this option to display tables with DataLinks that are linked to the currently selected form in the DataGroup pane.
Form	<p>Use this option to display tables containing DataLinks that are linked to a specific form in the DataGroup pane.</p> <p>Selecting this option might cause the DataGroup pane to be empty if the currently selected form does not contain any DataLinks.</p>
Automatically expand tables and select column when displaying DataLinks	<p>Use this option to automatically expand the tables in the DataGroup pane and select the column specified in the DataLink.</p> <p>This option is useful to quickly sift through large amounts of information in the DataGroup pane.</p>

Setting DataLink Filtering Options

DataLink filtering options enable you to customize your view of the information that appears in the DataLink pane of the DataLink Manager. For example, you can display only DataLinks that are linked from a specific table to the current form.

To set DataLink filtering options

1. Click the **Filtering** button in the DataLink pane.
2. Click the filtering options you want to use.

The DataLink List Filter Options dialog box is shown in [Figure 3-9](#).



Filtering button

DataLink List Filter Options

☐ Display all DataLinks for the current DataGroup

☒ Display any DataLink in the current DataGroup that links a control from:

☒ Any form

☐ Current form

☐ Form:

to a column in:

☒ Any table in the current DataGroup

☐ Selected table

☐ Table:

☐ Display only DataLinks for the current Table.Column

OK

Cancel

Figure 3-9
*Setting DataLink
Filtering Options*

The following table describes the DataGroup filtering options that you can set.

Option	Description
Display all DataLinks for the current DataGroup	Use this option to display all DataLinks in the current DataGroup.
Display any DataLink in the current DataGroup that links a control from:	The buttons in this group enable you to specify filtering options based on the control contained in the DataLink.
Any form	Use this option to display DataLinks that are linked to controls on any form in the project in the DataLink pane.
Current form	Use this option to display DataLinks that are linked to controls on the currently selected form in the DataLink pane. This option is useful if you want to quickly determine which forms in your project contain DataLinks. Selecting this option might cause the DataLink pane to be empty if the currently selected form does not contain any DataLinks.
Form	Use this option to display DataLinks that are linked to controls on a specific form in the DataLink pane. You can select a form in the project only if it contains DataLinks. Selecting this option might cause the DataLink pane to be empty if the specified form does not contain any DataLinks.
to a column in:	The buttons in this group enable you to specify filtering options based on the table and column specified in the DataLink.
Any table in the current DataGroup	Use this option to display DataLinks to any table and column in the DataGroup in the DataLink pane.
Selected table	Use this option to display DataLinks to the currently selected table in the DataGroup in the DataLink pane. Selecting this option might cause the DataLink pane to be empty if the currently selected table does not contain any DataLinks.
Table	Use this option to display DataLinks to a specific table in the DataGroup in the DataLink pane. Selecting this option might cause the DataLink pane to be empty if the specified table does not contain any DataLinks.
Display only DataLinks for the current Table.Column	Use this option to display only DataLinks that are linked to the currently selected table and column in the DataLink pane. This option is useful if two DataLinks to the same Visual Basic control have paths to two different tables. Using this option, you can quickly determine the different paths of the two DataLinks. The currently selected table and column is the table and column selected in the DataGroup pane.

Working with Models

Model Concepts	4-3
Text and Binary Model Files	4-4
Reading a Model at Design Time	4-4
Reading a Model at Runtime	4-4
Reflecting Changes in Your Database	4-5
Sharing Models Between Development Teams.	4-5
Using Multiple Models.	4-6
Capturing Your Database Structure	4-6
Planning Which Tables to Import	4-7
Using a Model to Develop a Single Application	4-7
Sharing a Model Across Applications.	4-8
Launching the Model Import Wizard	4-8
Setting Import Options	4-9
Selecting a Database Driver and Data Source	4-12
Logging On to Your Database.	4-13
Selecting Import Items	4-14
Updating Columns and Column Attributes	4-15
Verifying Primary Key Columns	4-16
Verifying Inferred Relationships	4-17
Working with Imported Models	4-18
Viewing a Model	4-19
Setting a DBMS.	4-20
Connecting to a Database	4-21
Disconnecting from a Database	4-21
Setting Primary Keys.	4-21
Refreshing and Saving a Model	4-22
Setting Model Properties	4-23
Setting General Properties	4-24

Working with Tables	4-26
Creating Tables	4-26
Specifying a Table Alias.	4-27
Creating Virtual Tables	4-27
How Virtual Tables Work	4-28
Specifying SQL Statements	4-29
Using Data from a Physical Table	4-30
Checking Your Syntax	4-30
Using a Routine	4-31
Modifying Tables	4-32
Working with Columns	4-32
Creating Columns	4-33
Specifying a Column Alias	4-34
Specifying a Data Type	4-35
Specifying a Numeric Column Size.	4-35
Setting Default Values	4-35
Selecting a Primary Key Column	4-37
Setting a Mandatory Column.	4-37
Setting a System-Generated Key.	4-37
Selecting a Changed Row Indicator Column	4-38
Specifying Intervals	4-38
Creating Calculated Columns	4-39
Aggregate Columns	4-39
Non-Aggregate Columns	4-39
Types of Calculation Functions	4-40
Entering SQL Text	4-40
Modifying Columns	4-41
Working with Relationships	4-41
Specifying Master-Detail Records and Lookup Fields.	4-41
Origin and Destination Columns	4-42
Primary Keys and Foreign Keys	4-42
Understanding Relationship Types	4-42
One-to-Many Relationships	4-43
Many-to-One Relationships	4-43
Implied Relationships	4-44
Creating Relationships	4-45
Specifying the Origin Column	4-46
Specifying Destination Information	4-46
Default Relationship Types	4-47
Using Multiple-Column Relationships	4-47
Modifying the Key Order	4-48
Modifying Relationships.	4-49

Models enable you to optimize database access and synchronize your application's user interface.

This chapter introduces Models and explains how to use them.

Model Concepts

A Model is a “snapshot” of the physical structure of a database and is made up of elements—tables, columns, and relationships—that describe a database structure in entity-relationship terms. A Model tells Data Director about the structure of the data your application will access and facilitates most of Data Director's design-time and runtime functionality. For example, a Model enhances the design process by providing you with access to your database structure without requiring you to log on to the database.

To capture the structure of your database, you simply import the database to a Model. The Model stores information about the characteristics of your database such as the table names and column data types, which columns are primary keys and which are foreign keys, and how the tables in the database are related to each other. Data Director then accesses the information contained in the Model file when you design and run your application.

At design time, Data Director reads the Model file to provide information about your database tables and columns. At runtime, Data Director reads the Model file to determine the structure of the database and thereby generate appropriate SQL statements.

Text and Binary Model Files

Data Director stores information about the structure of your database in two Model files: a text (.mlt) file and a binary (.mlx) file. Both of these files contain information about the database your application accesses at design time and runtime.

Data Director uses the text Model file at design time and the binary Model file at runtime. In addition, you distribute the binary Model file with your Data Director application. You can use the text Model file with your source code control system to make changes in the Model when building multiple-developer projects. For more information about maintaining Model files, see [“Working with Model Files” on page 8-10](#).

Reading a Model at Design Time

At design time, Data Director reads the Model file to provide application developers with information about database tables and columns.

Because a Model stores information about all of the tables and columns in a database, you do not need to have a connection to the database at design time to read database information. This means that when you start linking database columns to Visual Basic controls on a form at design time, Data Director uses the Model representation of your database to access database information.

Reading a Model at Runtime

At runtime, Data Director uses the information in your Model file to access your database, query the appropriate tables, and return a result set. The information about tables and columns in a Model determines what data appears in controls on your form, while the relationships determine additional application behavior, such as querying lookup data or detail records.

Data Director also uses a Model to dynamically generate SQL statements for your application. For example, when a user saves data at runtime, Data Director automatically generates the SQL INSERT or UPDATE statement required to save the data to the appropriate tables.



Important: If you move a compiled Model (.mlx) file, you might see an error when you run your project, stating that Data Director cannot locate your Model file. Data Director looks for a Model file in the Visual Basic environment using the following information: first, the hard-coded path of the Navigation Control's **ModelFile** property; then the directory containing the Data Director project (.ddx) file; and finally, the current directory. If you see this error message, either update the path of the Model in the Navigation Control Properties dialog box, or move the compiled Model (.mlx) file to the directory containing the Data Director project (.ddx) file.

Reflecting Changes in Your Database

When a source database changes, you can refresh the associated Model to ensure that both the database and the Model contain the same information. Data Director refreshes a Model with any changes that have occurred in the underlying database.

Sharing Models Between Development Teams

You can easily share a single Model across development teams and projects for the simultaneous development of multiple applications. Sharing a Model enables you to easily update and maintain information about your database in a central location—one change to a Model propagates across work groups and projects and is visible to all users of the Model.

Because Data Director stores Model information in a text Model (.mlt) file, you can save the shared Model file in a network directory. When you save a Model file in a network directory, the file is accessible by any developer that has access to the directory and file.

When sharing Models, keep the following considerations in mind:

- **Source code control system.** If you use a source code control system, you can check out Model files and make changes to them. When you check in the file, your changes are merged into the development environment. For more information about using multiple-developer projects, see [“Building Multiple-Developer Projects” on page 8-3](#).

- **Changes to shared Models.** Any saved changes to the Model affect all projects that use the Model. If you plan to modify a shared Model, you should ensure that any changes you make will not adversely affect the work of other developers.

For example, you might delete a table in the Model because you no longer have use for the table. However, if another developer has a project that uses the deleted table, that developer will experience problems. If another developer uses a DataGroup that contains the deleted table and has DataLinks based on the deleted table, all of the DataLinks to the deleted table will be broken.

- **Local copies of Models.** If you are going to make significant modifications to a Model file that is specific only to your project, make a local copy of that Model and then make the required modifications.

If you make local copies of a Model, ensure that your changes are integrated into the Model that you ship with the deployed application, or ensure that the changed Model file is shipped with the deployed application (see [Chapter 10, “Distributing an Application”](#)).

Using Multiple Models

You can use multiple Models in your application if you use databases from different vendors. For example, you can use databases from both Informix and Microsoft to develop an application.

Capturing Your Database Structure

To capture your database structure, you import information about an existing database to a Model—the resulting Model is the mirror image of the structure and characteristics of your database. That is, a Model contains information about all of the tables and columns in your database. It knows which columns are primary keys and which are foreign keys, and it knows how the tables in the database are related to each other.

Important: *Importing a database to a Model does not modify the database.*



You can import most databases to a Model. The Data Director ODBC DataDriver provides database access and automates connectivity to many commercially available databases.

Planning Which Tables to Import

Before you capture your database structure, you should determine which tables you want to import. You can import a subset of the tables in your database, or you can import the entire structure of your database. Generally, you import a subset of your database structure when you plan to develop a single application, and the entire structure of your database when you plan to share the Model with other developers to develop multiple applications within a work group. You can also import database objects that you own or all objects to which you have access.

Generally, a database administrator creates a database and grants privileges to users of the database, such as system administrators who manage the structure of the database.



Important: When you import a database to a Model, you must ensure that you have read privileges to all the tables you want to import.

Using a Model to Develop a Single Application

If you are developing a single application, you probably do not need to work with the entire structure of your database. If so, import a subset of your database scheme, importing only the specific tables that are relevant to the application you are designing.

For example, if you are developing an accounts payable application, import tables related to vendors and employees so that end users can enter invoices and employee expense reports. You do not need to import tables related to inventory items because these tables have no relevance to an accounts payable application.

Following are some of the advantages of working with a single Model:

- The Model is easily maintainable.
- The Model contains only those elements crucial to your development effort.

Sharing a Model Across Applications

If you plan on using a Model for a development project that involves multiple applications, you probably want to work with the entire contents of your database. If so, import the entire structure of your database into a single Model. You can then share the Model across a work group or development team.

For example, if you are developing a complex suite of financial applications, many of the tables in your database will be shared between different applications. Customers are entered into a Customers application, orders are entered for customers when they purchase products, invoices are generated from a Receivables application to send to customers that ordered products, and finally, account transactions are posted to a General Ledger product to maintain account reconciliations.

While you might not use all of the tables in a Model to develop each application, many of the tables in the Model are shared across applications. For example, the table that contains customer information can be shared between the Customers application and the Orders application.

Although you might end up with a large Model when you import your entire database structure to one Model, there are a number of benefits to sharing a Model across applications:

- The Model is easily shared and maintainable across a work group.
- One change to the Model propagates to all users of the Model.

Launching the Model Import Wizard

Data Director provides you with a Model Import Wizard that enables you to easily capture your database structure. When you run the Model Import Wizard, it guides you through each step of the import process.

Tip: Importing a database to a Model does not modify the database.



To launch the Model Import Wizard

1. Choose **Data Director→Model→Import** from the Visual Basic menu bar.

The Model Import Wizard is shown in [Figure 4-1](#).

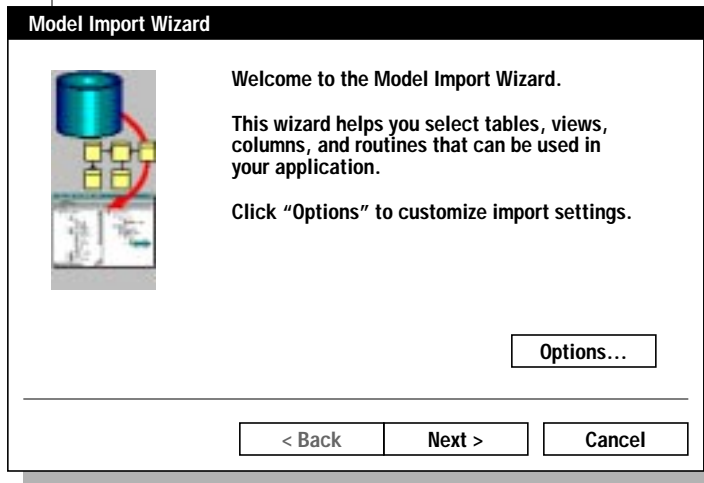


Figure 4-1
Launching the Model Import Wizard

Setting Import Options

Data Director provides you with import options that help facilitate the import process, as shown in [Figure 4-2](#). Before you import a database, you should review the options and update them for the database that you are importing. Default options are set after you import a database for the first time.

Import options persist for the current Model. You can also save preference settings to use as default settings for any additional databases that you import. When you import a database, Data Director reads the database system catalog to determine the structure of your database.

Import Options

Objects

- ☒ Tables
- ☒ Views
- ☐ Routines
- ☒ Primary Keys
- ☒ Foreign Keys
- ☒ Inferred Relationships

Qualification

- ☐ Include System Objects
- ☒ Owned Objects Only

Verification

- ☒ Primary Keys
- ☒ Inferred Relationships

Prompting

- ☒ Drop Old Table Columns
- ☒ Different Data Types

Default:

- ☐ Use Model Settings
- ☒ Use Database Settings

Primary Key Default

- ☒ Use Model Settings
- ☐ Use Database Settings

☒ Set Current Settings as Default

☐ Full Logon Prompting

☐ Convert Routines to Tables

OK Cancel

Figure 4-2
Setting Import
Options

The following table describes the import options that you can set.

Option	Description
Objects: Tables	Use this option to choose the specific database tables to import. When you import tables, Data Director imports the table names, owners, descriptions, primary keys, columns, column descriptions, and column data types.
Objects: Views	Use this option to choose the specific database views that you want to import during the import process.
Objects: Routines	Use this option to choose the specific routines to import.
Objects: Primary Keys	You can import primary keys if they are specified in the underlying database. If primary keys are not specified in the underlying database, during the import process you can choose to specify a primary key column for each table in the database if verification is turned on for primary keys.
Objects: Foreign Keys	You can import foreign keys (relationships) if they are specified in the underlying database. In some databases, foreign keys are specified as part of referential integrity.

(1 of 3)

Option	Description
Objects: Inferred Relationships	When you use this option, Data Director analyzes the primary keys column settings in the database and attempts to infer relationships based on the column name and data type.
Qualification: Include System Objects	Use this option to import system objects.
Qualification: Owned Objects Only	<p>Use this option to import database elements that have been created with your user ID. For example, if you log on using the user ID of sysadmin, Data Director imports all of the database elements created by the user ID sysadmin.</p> <p>To import all tables you have access to, uncheck this check box. For example, if you log on using the user ID of sysadmin, and have table access granted to you from dba, Data Director imports all of the database elements created by the user IDs of both sysadmin and dba.</p>
Verification: Primary Keys	Use this option to verify primary key selection during the import process.
Verification: Inferred Relationships	Use this option to verify inferred relationships during the import process.
Prompting	The settings in this group enable you to confirm differences that Data Director finds between a database and a Model when you refresh a Model.
Drop Old Table Columns	Use this option to drop columns in the Model when Data Director encounters columns in a Model that no longer exist in a database during a refresh.
Different Data Types	Use this option to choose between Model data types or database data types when Data Director encounters differences between Model values and database values during a refresh.
Prompting: Default	This setting indicates whether Data Director uses Model values or database values when it encounters conflicts between a Model and a database during a refresh.

(2 of 3)

Option	Description
Primary Key Default	<p>This setting indicates whether Data Director uses Model values or database values when it encounters conflicts between a Model and a database during a refresh.</p> <p>If a table is found in a Model that does not have a primary key set, but the database does, Data Director uses the database settings for the table in the Model.</p>
Set Current Settings as Default	<p>Use this option to save the current Model import options for future imports into new Models.</p>
Full Logon Prompting	<p>This setting indicates whether Data Director prompts for the database name, login, and password for the DBMS associated with a Model each time you refresh an imported Model.</p>
Convert Routines to Tables	<p>Use this option to convert routines to virtual tables during the import process rather than manually converting them. Only routines with no input or output parameters are converted. When you convert routines to virtual tables, Data Director names the virtual tables using the name of the routine from which it was created, and the ROUTINES statement is set as the query command for the virtual table.</p>

(3 of 3)

Selecting a Database Driver and Data Source

The Model Import Wizard prompts you to select the database driver and data source that you want to import, as shown in [Figure 4-3](#).

Once you import a database to a Model, you cannot change the type of database that the Model is associated with using the Model Import Wizard.

Tip: To change the target database that a Model is associated with, set the DBMS after you import a database to a Model (see [“Setting a DBMS” on page 4-20](#)).



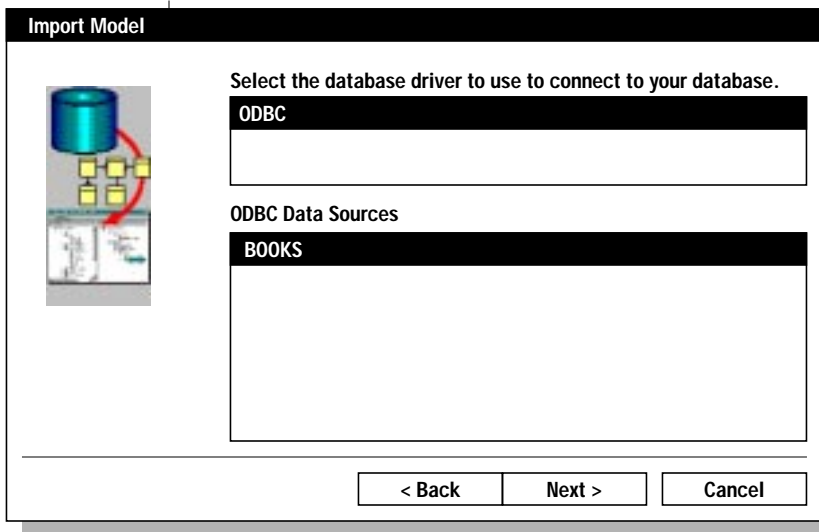


Figure 4-3
*Selecting a Database
Driver and Data Source*

Logging On to Your Database

After you choose a data source, specify which database to import and provide a logon ID and password for the database.

If you have already associated a database name, logon ID, and password for the data source when you set up the data source, and do not choose **Full logon prompting** in the Import Options dialog box, Data Director does not prompt you for logon information.

After you enter logon information for a database, the Model Import Wizard prompts you to choose the database that you want to import to a Model.

Database logon information such as user name, database, and server, which is used when importing a Model, is stored in the Data Director Model file. Additionally, some DBMS-specific information might be contained in the ODBC connect string stored in the Model file.

Data Director re-uses this logon information for convenience at runtime. If a Model file created against one DBMS is used by an application running against another DBMS, application users might be prompted for logon information, because they are logging on with default values that were appropriate for the first DBMS but not for the current one.

For security reasons, passwords are not stored in the Model file. Application users will typically be prompted for a password if the database requires one and the **Logon Prompt** option is turned on (the default).

Selecting Import Items

During the import process, Data Director logs on to your database to provide you with a live view of the database elements available for import. You can easily select the tables, views, and routines that you want to import into your Model.

Select the database items you want to import to a Model by checking the boxes next to the items that you want to import, as shown in [Figure 4-4](#). If you do not want a database item in your Model, uncheck the check box next to the item name.

When you click a “+” check box, Data Director queries your database to display the detail items for the table. If there are no detail items for the table, the “+” check box disappears when you click it.

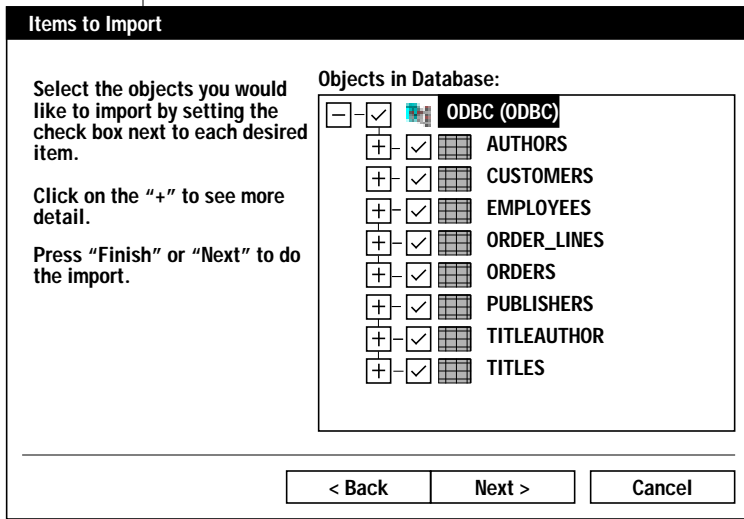


Figure 4-4
Selecting Import Items

Updating Columns and Column Attributes

When you refresh an existing Model, Data Director compares the imported database structure with the elements in the Model.

If your Model contains columns that conflict with those specified in your database, the Model Import Wizard prompts you to remove any columns in the Model that are not consistent with those in the underlying database.

Tip: This prompt depends on whether you set the **Drop Old Table Columns** import option.

Similarly, if your Model contains column attributes that conflict with those specified in the database, the Model Import Wizard prompts you to update the columns and column attributes. You can use either the database column attributes or the Model column attributes.



Verifying Primary Key Columns

If Data Director encounters any tables in the underlying database that are missing primary key columns, it prompts you to select primary key columns for those tables, as shown in [Figure 4-5](#).

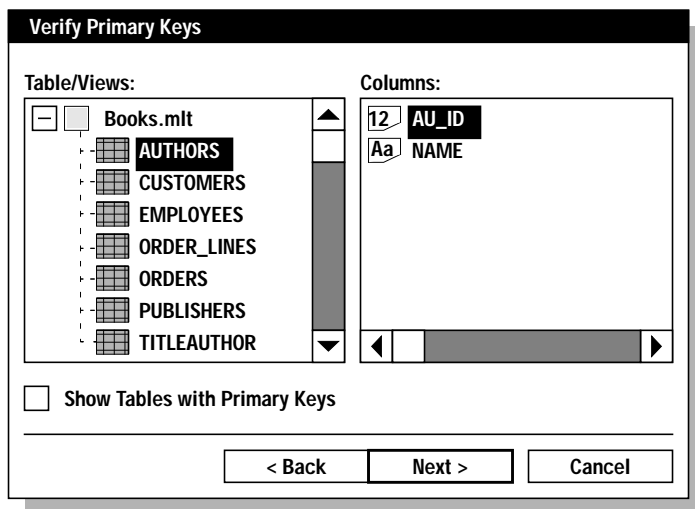


Figure 4-5
Verifying
Primary Keys



Tip: If you choose to skip the primary key selection for a particular table, you can specify a primary key for a column at a later time using the Primary Keys dialog box (see [“Setting Primary Keys” on page 4-21](#)).

When you refresh an existing Model, and the Model contains primary key information that conflicts with primary key information specified in the database, Data Director uses the primary key default that is set in the Import Options dialog box.



Important: If you specify primary keys in the Model but not in the underlying database, the Model primary keys are automatically used when you refresh a Model.

Data Director uses primary key information when updating and deleting rows of data. Relationships are also determined by primary key information. Relationships are discussed in greater detail in [“Working with Relationships” on page 4-41](#).

Verifying Inferred Relationships

Data Director infers relationship types based on primary key information. When Data Director encounters a primary key column, it searches for columns of the same name in other tables in the Model. Each time Data Director finds a match between two columns, it infers a relationship based on the primary key information; you can then accept the inferred relationship if it is valid.

For example, if Data Director encounters the primary key **CUST_ID** in the **CUSTOMERS** table, it searches for columns having the same name in other tables. When Data Director finds the **CUST_ID** column in the **ORDERS** table, it is able to infer a one-to-many relationship between the **CUSTOMERS** table and the **ORDERS** table based on the **CUST_ID** column in the **ORDERS** table.

When you verify relationships, the Model Import Wizard displays the originating column as the origin column and the inferred foreign key column as the destination column. Selecting each origin column updates the destination column with the inferred relationships. You can also change the relationship type of each inferred relationship (also see [“Modifying Relationships” on page 4-49](#)).

Verify Inferred Relationships

Data Director has found some possible relationships between the imported tables. Please accept or reject these relationships by selecting each origin column, the related destination column, and setting the “Accepted” check box.

Origin Column	Type	Destination Column
12 AUTHORS.AU_ID	<input type="radio"/> One-to-Many <input checked="" type="radio"/> Many-to-One	12 AUTHORS.AU_ID
12 TITLEAUTHOR.AU_ID		
12 CUSTOMERS.CUST_ID		
12 ORDERS.CUST_ID		
12 PUBLISHERS.PUB_ID		
12 TITLES.PUB_ID		

☐ Accepted

Mark All Accepted

Show Existing Relationships...

< Back Finish > Cancel

Figure 4-6
Verifying Inferred Relationships

During a refresh, you can view the currently defined relationships between origin tables and destination tables as shown in [Figure 4-7](#) by clicking the **Show Existing Relationships** button.

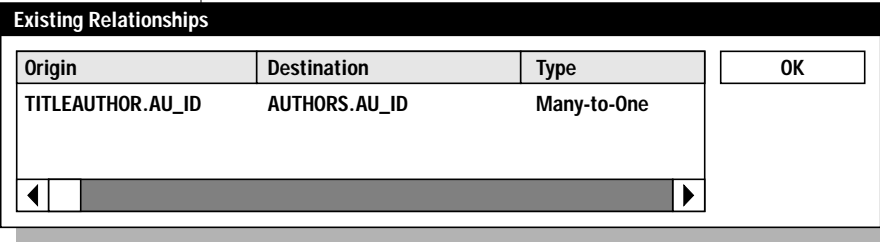


Figure 4-7
Viewing Existing Relationships



Tip: Data Director imports but does not infer multiple-key relationships. If you use multiple-column primary keys and have not yet defined them in the database, you might need to manually create relationships in the Model Viewer (see [“Creating Relationships” on page 4-45](#)).

Working with Imported Models

Once you capture your database structure, there are some basic tasks that you can perform using a Model, such as opening and saving the Model.



Tip: If you import a Model with the Model Viewer open, it remains open. If you import a Model without opening the Model Viewer, it is opened during the import and closes after you are prompted to save the Model.

Viewing a Model

You can open a Model by choosing **File→Open Model** from the Visual Basic menu bar. After you import your database, Data Director displays information about the database in the Model Viewer window, as shown in [Figure 4-8](#). For more information about using the Model Viewer, see [“The Model Viewer User Interface”](#) on page 1-10.

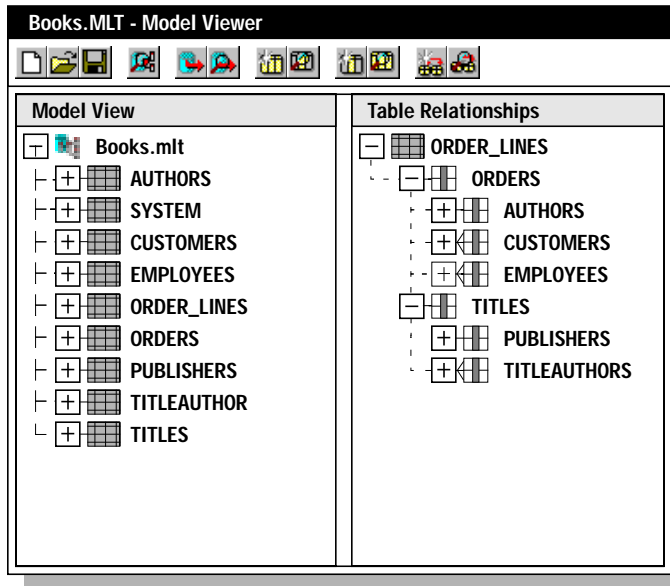


Figure 4-8
Viewing an
Imported Model

The left side of the Model Viewer shows the tables and columns in the Model, including the table owners. The right side of the Model Viewer shows the relationships between tables in the Model.

The expanded Model indicates the type of relationship that is defined between tables. Relationships between tables are graphically denoted in the table tree view as follows:

- Many-to-one (one-to-one) relationships are denoted by a straight line.
- One-to-many relationships are denoted by ←

For example, [Figure 4-8](#) indicates that **CUSTOMERS** is related to **EMPLOYEES** through a many-to-one relationship and that **CUSTOMERS** is related to **ORDERS** through a one-to-many relationship.

Setting a DBMS

Setting a DBMS enables you to change the type of database on which a Model is based. For example, you can design an application using a Microsoft Access database, and then deploy the application using an INFORMIX-OnLine database.

To ensure that the Model is consistent with the data types supported by the DBMS, refresh your model after changing the DBMS (see [“Refreshing and Saving a Model” on page 4-22](#)).

To set a DBMS

1. Select the Model in the Model Viewer.
2. Right-click the Model.
3. Choose **Set DBMS** from the popup menu.
4. Select a database driver and a data source from the lists in the Select Model Data Source dialog box, as shown in [Figure 4-9](#).

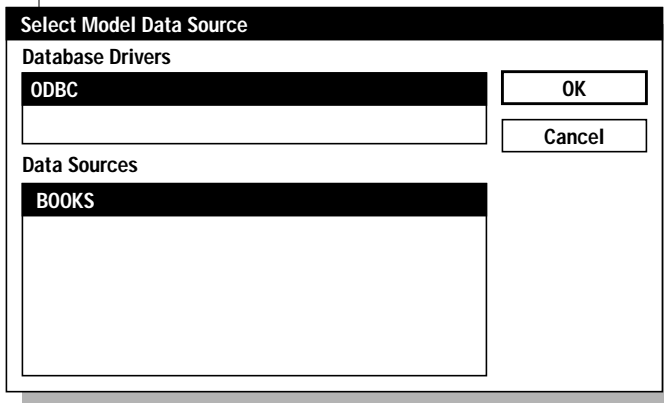


Figure 4-9
*Selecting a Database Driver
and Data Source*

Connecting to a Database

You can easily connect to a database from the Model Viewer after you associate a Model with a database. Connecting to a database is useful if you are performing operations that require frequent connections to your database, such as creating virtual tables.

To connect to a database

1. Right-click the Model in the Model Viewer.
2. Choose **Connect** from the popup menu.
3. Enter your password, and click **OK**.

***Important:** Some databases do not require a password. In these cases, you will not see a dialog box asking for a password.*



Disconnecting from a Database

After you complete your database operations, you can disconnect from your database.

To disconnect from a database

1. Right-click the Model in the Model Viewer.
2. Choose **Disconnect** from the popup menu.

Setting Primary Keys

If you did not choose to set primary keys during the import process, you can set primary key information for columns after you import a Model.

To set primary keys

1. Right-click the Model in the Model Viewer.
2. Choose **Set Primary Keys** from the popup menu.
3. Select primary keys for any tables that are missing primary key information.

Figure 4-10 shows the Primary Keys dialog box.

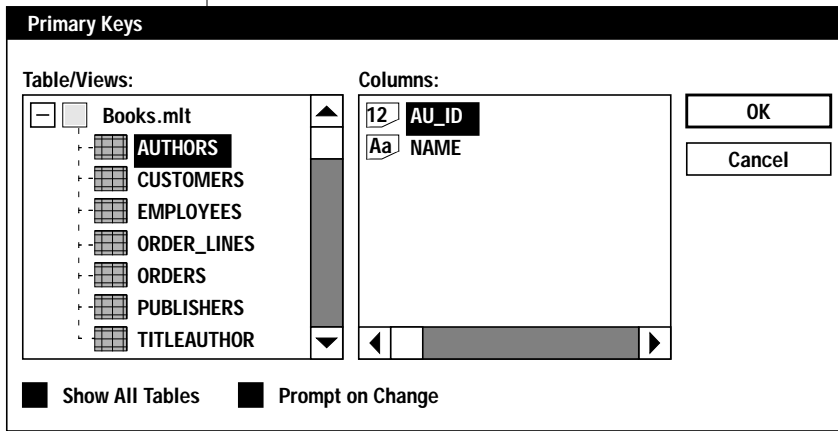


Figure 4-10
Selecting Primary Keys

Refreshing and Saving a Model

If you make changes to your database, you should ensure that the following Model elements match the underlying database:

- Table names and owners
- Column names, data types, and sizes

All other Model information, such as table and column descriptions, is database-independent. For example, if you enter a column default value in your Model and then update the Model, the column default value is not overwritten.

If the structure of your database changes, you should refresh your Model to ensure that both your database and the Model contain the same information.

If you make changes to the underlying database that is associated with a Model, you can easily refresh the Model by importing your database to the existing Model. Data Director identifies any differences between the database and your Model and prompts you to refresh your Model to reflect those differences.

For example, if a column has the same name in both the Model and the database, but the data type does not match in both sources, Data Director notifies you that the columns do not match and prompts you to specify whether you want to overwrite the Model with the database data type value.

To refresh a Model

1. Open the Model in the Model Viewer.
2. Choose **Data Director→Model→Import** from the Visual Basic menu bar.
3. Click **Refresh Current** in the Refresh or Import? dialog box.
4. Click Options to modify the last-used import options.
If you do not change the import options, the Model will be refreshed as specified by the last-used options. Only changed and additional database items specified by the Model Import Options criteria will be used to refresh the Model.
5. Follow the Model Import Wizard instructions.

To save a Model

1. Choose **File→Save Model** from the Visual Basic menu bar.

Setting Model Properties

There are several properties for Models that you can view or modify. For example, you can view information about an imported Model, such as the database associated with the current Model.

If you modify the property settings for a Model, your changes take effect when you close the dialog box, and they persist until the next time you modify them.

To view the Model Properties dialog box

1. Choose **Data Director→Model→Properties** from the Visual Basic menu bar.

Setting General Properties

General properties provide you with information about a Model, such as the type of database with which the Model is associated and the number of tables and columns in the Model. The **General** page is shown in [Figure 4-11](#).

Model Properties - Books.mlt

General

Name:Books.mlt

Description:

File Name:C:\Program Files\Informix\Data Director\Books

DBMS:ODBC

Type:Informix

Database:BOOKS

Data Source:BOOKS

Creator:admin

Time Created:Sun Jun 09 16:51:34 1997

Number of Tables:9

Number of Columns:78

Number of Relationships:18

OK

Cancel

Apply

Figure 4-11
Viewing Model Properties

The following table describes each Model property; you can modify the Model name, description, and database type. To change additional Model information, refresh the database or save the Model to a new name or file location using the Visual Basic **File** menu.

Property	Description
Name	Displays the filename of the current Model.
Description	Displays the description of the current Model.
Filename	Displays the fully specified filename of the current Model. To modify this information, choose File→Save Model As from the Visual Basic menu bar, and then choose a new location and name for the Model.
DBMS	Displays the vendor name of the underlying database.
Type	Specifies that generic ODBC or DBMS-specific SQL syntax should be used, such as join and system-generated key syntax.
Database	Displays the database associated with the Model. The INFORMIX-OnLine databases require a database name. (If the DBMS for this Model does not require a database name, this field is empty.)
Data Source	Displays the name of the data source associated with the current Model.
Creator	Displays the user name of the person who created the current Model.
Time Created	Displays the time the current Model was created.
Number of Tables	Displays the number of tables in the current Model.
Number of Columns	Displays the number of columns in the current Model.
Number of Relationships	Displays the number of relationships in the current Model.

Working with Tables

A table contains information about specific elements in a database. For example, a **CUSTOMERS** table contains information about customers, such as customer names and addresses.

You can work with the following types of tables using Data Director:

- **Tables.** Tables refer to standard database tables that reside in a database.
- **Views.** Views refer to standard database views that reside in a database.
- **Virtual tables.** Virtual tables are constructed by Data Director based on user-defined SQL statements, routines, or a combination of both. Use virtual tables to pass custom SQL statements and routines directly through Data Director to your database.

Creating Tables

When you import a database, Data Director automatically creates tables in a Model based on the items that you import. However, there might be times when you need to create a new table, such as when you want to quickly update your Model to match your database without refreshing your entire Model.

Data Director requires that you provide each table in a Model with a unique name; all other information is optional. The description can include anything you like, such as the purpose of the table or the type of information it stores.

Important: *The table names in your Model must match the table names in your database. If the database is case sensitive, consider case when naming tables.*

To create a table

1. Choose the **Add Table** menu item:
Choose **Insert→Add Table** from the Visual Basic menu bar. ♦
Choose **Project→Add Table** from the Visual Basic menu bar. ♦
Alternatively, you can right-click the Model (the **.mlt** icon) in the Model Viewer, and choose **Add Table** from the popup menu.



VB 4

VB 5

2. Enter a name for the new table in the Table Properties dialog box.
3. Optionally, enter additional information for the new table.

The Table Properties dialog box is shown in [Figure 4-12](#).

The image shows a 'Table Properties - AUTHORS' dialog box. It has three tabs: 'General', 'Relationships', and 'Commands'. The 'General' tab is selected. Inside the 'General' tab, there are four input fields: 'Name' (containing 'AUTHORS'), 'Alias', 'Owner', and 'Description'. Below these fields is a checkbox labeled 'Virtual'. At the bottom of the dialog are three buttons: 'OK', 'Cancel', and 'Apply'.

Figure 4-12
Creating a New Table

Specifying a Table Alias

A table alias provides a name for a table that is easily identifiable. For example, if a database table is named **CUST_TRX** in the underlying database, you can specify an alias of **Customer Transactions** to easily identify the table while designing an application.

Creating Virtual Tables

Virtual tables enable you to pass custom SQL statements directly through Data Director to your database. For example, virtual tables can be used to execute the following commands and queries:

- Ad hoc queries
- Subqueries
- Group-by clauses
- Routines

How Virtual Tables Work

At design time, you can easily create a virtual table and assemble the SQL statements that construct the table. When you check the syntax for a virtual table, Data Director executes the SQL statement specified for the table and returns the columns generated by the SQL statement. You can then link those columns to Visual Basic controls in an application window.

At runtime, Data Director executes the SQL statements stored in a virtual table instead of dynamically creating SQL statements as it does for regular tables. Virtual tables also store the result set returned by the statements set in the virtual table, allowing you to access and manipulate the data in the table in an application window.

If your initial SQL query is a single-table SELECT statement, Data Director automatically generates any necessary INSERT, UPDATE, and DELETE statements in the virtual table.

To create a virtual table

1. Create a table using the Table Properties dialog box.

For detailed instructions, see [“To create a table” on page 4-26](#).

2. Check the **Virtual** check box on the **General** page to enable the **Commands** page.

The **Commands** page is visible only when a virtual table is created or selected.

3. On the **Commands** page, enter the SQL statements that you want to associate with the virtual table.
4. Click **Check Syntax** to check your SQL syntax.

The virtual table will not be created unless the syntax is checked and passes.

The **Commands** page is shown in [Figure 4-13](#).

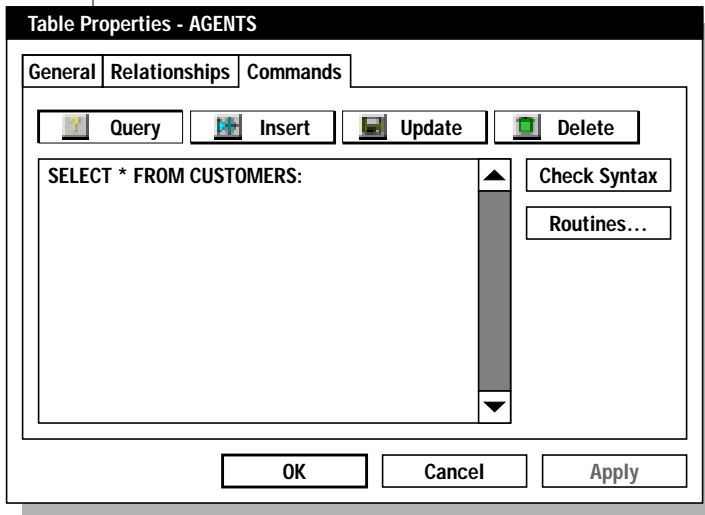


Figure 4-13
Specifying SQL Statements

Specifying SQL Statements

Specify the SQL statements that you want to associate with the virtual table. To enter a SELECT statement, click **Query** to add the SELECT syntax. The SELECT statement determines the set of columns that the virtual table contains, while the remaining statements determine what operations Data Director performs with the data in the table.

To enter additional statements, such as INSERT statements, click the appropriate button, and enter the rest of your SQL statements.

If a single-table SELECT statement is specified for a query using an ad hoc query (SELECT *), Data Director automatically generates the INSERT, UPDATE, and DELETE statements for a virtual table. You can override this behavior by defining and setting each command explicitly in the virtual table.

Setting SQL statements in a virtual table allows you to customize the commands that Data Director executes. For example, when an end user deletes a record, Data Director automatically executes the DELETE statement associated with the virtual table. Using a virtual table, you could execute a customized routine that runs in place of the DELETE statement when Data Director attempts to delete a record.

Using Data from a Physical Table

You can retrieve data from the current record in a physical table and place it into the SQL statement used by the virtual table. For example, if you have a virtual table called **ORDERS**, and you want to show only orders for the current customer in your query, your SQL statement would look like this:

```
SELECT * FROM ORDERS WHERE CUST_ID = ::CUSTOMERS.CUST_ID
```

The double colon (: :) indicates to Data Director that `CUSTOMERS.CUST_ID` is a bind variable. Data Director then replaces the variable with the value in the current record of **CUSTOMERS** for **CUST_ID**.

You can use replacement parameters for the **SELECT**, **INSERT**, **UPDATE**, and **DELETE** statements. Because parameter replacement provides a runtime binding to a fetched data value in the current record of a referenced table, you can use replacement parameters only at runtime.

Checking Your Syntax

Click **Check Syntax** to check the syntax in your SQL statements. When you check syntax, Data Director connects to your database if necessary and describes the command. Checking your syntax allows Data Director to catch errors in your commands and is a required step for Data Director to extract column information from your database.

After Data Director describes the **SELECT** statement, it creates a column in the virtual table for each output column. You can view the SQL text that Data Director generates for each column in the Model Viewer and DataLink Manager, enabling you to link columns resulting from a **SELECT** statement to Visual Basic controls.

If your **SELECT** statement is an ad hoc query (**SELECT ***), Data Director does not automatically populate the SQL text field for the properties of each column returned by the command (see [“Creating Calculated Columns” on page 4-39](#)).

Using a Routine

In traditional, relational-database parlance, a routine is simply a stored procedure. To use a routine in your virtual table, click **Routines** on the **Commands** page of the Table Properties dialog box. Data Director logs on to your database and retrieves a list of the available routines. Select a routine from the list to include it in your SQL statement.

The Routine Selection dialog box is shown in [Figure 4-14](#).

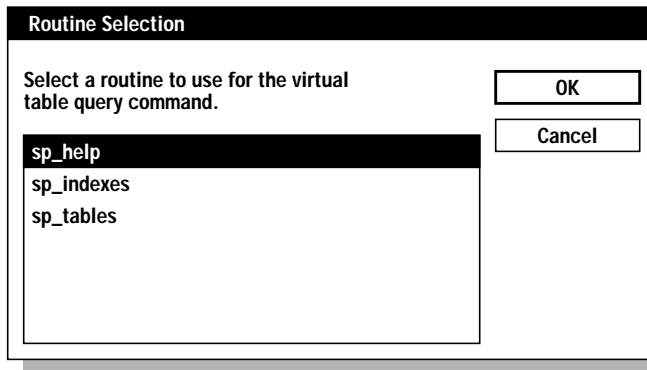


Figure 4-14
Using Routines

To use a routine with INFORMIX-OnLine servers, use the standard ODBC syntax, which ODBC converts to the appropriate syntax for the database, for example:

```
{call sp_help('parameter')}
```



Tip: If your ODBC driver supports parameterized statements, you can also use a question mark (?) in place of “parameter,” which signifies that the question mark is to be replaced with a parameter supplied by virtual-table code at runtime. Using parameterized statements in this way is more flexible and can speed the performance of your application.

To determine if your ODBC driver supports parameterized statements

1. Open Data Director Connect, and connect to your database.
The **Data Director Connect** icon is located in your **Data Director** icon group.
2. Click the **ODBC API Functions** button.

3. Look for the **SQLDescribeParam** check box in the **ODBC Level Two Functions** group.

If the check box is checked, your ODBC driver supports parameterized statements. Otherwise, your ODBC driver does not support parameterized statements, and you must include actual parameter values within your routines.

Modifying Tables

If you have made changes to your database, manually making the same changes in your Model can be a time-saving alternative to refreshing the Model if you have a large database and your database changes are minimal. For example, if you only add one column or table to your database, you can manually add the same column or table in your Model instead of updating the Model.

You might also modify imported tables to provide additional information such as table descriptions. This information does not affect your application, but might be useful for members of a group project.

To modify a table

1. Select the table in the Model Viewer.
2. Choose **Data Director→Model→Table Properties** from the Visual Basic menu bar.
3. Modify the table's properties.

For detailed instructions, see [“To create a table” on page 4-26](#).

Working with Columns

The data in columns determines what information appears within fields in the windows of your application. For example, the information in a **Customer Name** field generally comes from the **NAME** column of the **CUSTOMERS** table.

A table can contain standard columns or calculated columns. In addition, you can specify an alias for a column and you can modify the SQL text that is derived for calculated columns and columns returned for a virtual table.



VB 4

VB 5

Creating Columns

Columns must have a unique name, a data type, and a size; all other column information is optional.

Important: *The column names, data types, and sizes specified in your Model must match the underlying database.*

To create a column

1. In the Model Viewer, select the table to which you want to add a column.
2. Choose the **Add Column** menu item:
 Choose **Insert→Add Column** from the Visual Basic menu bar. ♦
 Choose **Project→Add Column** from the Visual Basic menu bar. ♦
 Alternatively, you can right-click a table in the Model Viewer, and choose **Add Column** from the popup menu.
3. Enter a name, data type, and size for the new column in the Column Properties dialog box.
 Data Director requires a column name and data type. For certain fields, such as those with a data type of CHAR, VARCHAR, and DECIMAL, Data Director also requires a size; for other data types, this text-entry box is disabled. All other column information is optional. You can enter optional information at any time.
4. Optionally, enter an alias, default, and SQL text for the column.
5. Optionally, specify whether the column is a primary-key column, a mandatory column, a system-generated-key column, or a row-indicator column.

The column description can include information such as the purpose of the column or the type of information it stores.

The Column Properties dialog box is shown in [Figure 4-15](#).

Column Properties - ADDRESS1

General

Name: ADDRESS1

Alias:

Data Type: VARCHAR Size: 255

Default:

SQL Text:

Description:

☐ Primary Key ☐ System Generated Key

☐ Mandatory ☐ Change Row Indicator

Date Format String:

Intervals...

Figure 4-15
Creating a
Column

Specifying a Column Alias

A column alias provides a name for a column that is easily identifiable. For example, if a database column is named **CUST_TRX_NUMBER** in the underlying database, you can specify an alias of **Customer Transaction Number** to easily identify the column when designing an application.

Column aliases are also useful to easily identify the columns generated for a virtual table. For example, a column generated for a virtual table might be named **AMOUNT*20**, which might not be useful information when designing an application. Using column aliases, you could rename the column to **Commission Earned** to easily identify the column in the DataLink Manager.



Specifying a Data Type

A data type determines what type of information a column stores. For example, the data type NUMERIC stores fixed-point numbers. If you are using an Oracle database and you specify a DATE format, the **Date Format String** text box is enabled so that you can enter a date format.

Important: *Because a data type specifies the kind of values that will be entered into a field in an application window, be aware of type conflicts when choosing data types. For example, you cannot store text data in a field that has been defined with the NUMERIC data type.*

Specifying a Numeric Column Size

Specify a size for columns of data type CHAR and NUMERIC. For CHAR columns, enter the maximum number of characters that a column can store in an application field. For NUMERIC columns, enter the size as you want it to display in an application field.

You can enter numeric sizes in precision, scale, or decimal format. For example, to denote a nine-digit number with seven digits on the left side of the decimal point and two digits on the right (0000000.00), you could enter either of the following values:

- 9, 2 (precision, scale)
- 7. 2 (decimal)

Setting Default Values

A default value provides an initial value for new records when an end user runs an application. A default value can be a constant or a function, and it persists in an application until an end user changes the value.

If you enter a default value for a column in your Model, Data Director automatically uses it for any DataLink that uses that column. You can enter a different default value or disable the Model default value for a specific table and column in a DataGroup (see [“Setting DataGroup Column Properties” on page 5-37](#)).

Constant Values

You can enter any constant value as long as it matches the data type of the selected column. For a CHAR column, a constant value must be less than or equal to the column size.

All characters are valid constants except for the equals (=) and single quote (') symbols. To include either of these symbols in a constant value string, precede the string with a single quote. Examples are listed in the following table.

If You Enter	The Default Value Is
ABC	ABC
'ABC	ABC
' 'ABC	'ABC
'=ABC	=ABC

Calculation Functions

You can enter one of the following calculation functions for columns of data type DATE and TIME. Functions are not case sensitive and do not require an argument, but you must precede a function with an equals sign (=).

Function	Returns	Syntax
Date	The current local system date	=Date or =Date()
Time	The current local system time	=Time or =Time()

Selecting a Primary Key Column

A primary key column uniquely identifies each record stored in a table. For example, each employee in the **EMPLOYEES** table might be uniquely identified by the value in the **EMP_ID** column. You can select multiple primary keys for a table if the table has more than one column that uniquely identifies the data in the table.

Selecting a primary key column for a table enables Data Director to coordinate data between multiple tables in a database. Data Director also uses primary keys to update and delete data.

Setting a Mandatory Column

Data Director enables you to specify a column as a mandatory column. At runtime as the application end user enters data, Data Director verifies that data has been entered for any mandatory columns. If no data exists for that column, the user is prompted to enter data.

Setting a System-Generated Key

A system-generated key enables you to generate a unique record number for each new record that a user creates in an application. For example, you can use a system-generated key column to automatically generate unique, sequential employee numbers for new employees entered into a database.

Frequently referred to as identity columns, system-generated keys are generally primary key columns, but you can use any column in a database as a system-generated key.

Data Director currently provides system-generated key support for the following databases:

- OnLine databases
- Microsoft Access, Sybase, and Microsoft's SQL Server

Any OnLine column that you designate as a system-generated key must have a data type of SERIAL.

You can use a system-generated key column to create DataLinks from a control in your application window to the database column that generates the unique value. Currently, you can create DataLinks to system-generated key columns using text box controls only.

When a user runs an application that contains a DataLink to a system-generated key, Data Director automatically generates a unique sequence number when a user saves data. If a user types a value into a DataLink that is linked to a system-generated key, Data Director removes that value and replaces it with the automatically generated value.

For information about creating DataLinks, see [“Creating DataLinks” on page 6-6](#).

Selecting a Changed Row Indicator Column

Data Director uses a changed row indicator column as an identifying column when detecting if values in a cached result set have changed relative to the underlying database when using the optimistic detect concurrency control method. For more information about using concurrency control, see [“Concurrency Control Concepts” on page 5-7](#).

Specifying Intervals

If you select the DATETIME or INTERVAL data types when using an Informix database, the **Intervals** button is enabled. Click **Intervals** to select a start value (the largest qualifier) and an end value (the smallest qualifier) for the range of values you want this column to return when queried.

Creating Calculated Columns

Calculated columns enable you to perform a calculation on database columns and return the calculated value in a result set. When you create calculated columns, you essentially add a column to the table that will contain the calculated value. For example, you might want to add two columns together and return the sum of the two columns in a separate field in an application window.

You can add calculated columns to regular and virtual tables. By default, all columns in virtual tables are calculated columns.

Calculated columns cannot be updated and are used to display information in an application window.

There are two types of calculated columns that you can create:

- Aggregate columns
- Non-aggregate columns

Aggregate Columns

Aggregate columns return one value based on either a SELECT statement on a single database column or based on the result of a calculation performed on multiple database columns. For example, you can retrieve the maximum value of a record in a column.



Tip: Data Director queries aggregates into a separate table. If you wish to use aggregates that return multiple rows requiring special qualification, such as group-by or order-by, construct a related, virtual table at design time for this purpose.

Non-Aggregate Columns

Non-aggregate columns can contain any calculations supported by the underlying database. For example, you can concatenate an employee's first name and last name and display the full name in a field in an application window.

Types of Calculation Functions

The following table lists some of the types of calculations you can perform using calculated columns. The calculations that you can perform are based on the native SQL format that your database uses and the types of calculations that your database supports.

Function Name	Description
+ - * / %	Simple math operators
AVG(TABLE . COLUMN)	Returns the average value of a column
Date functions	Returns date values
Count(*)	Returns the total number of records in the result set
MAX(TABLE . COLUMN)	Returns the maximum value of a record in a column
MIN(TABLE . COLUMN)	Returns the minimum value of a record in a column
SUM(TABLE . COLUMN)	Returns the total of TABLE . COLUMN for all records in the result set

To create a calculated column

- 1. Create a column using the Column Properties dialog box.
For detailed instructions, see “To create a column” on page 4-33.
- 2. Specify the SQL text for the calculated column.

Entering SQL Text

To specify the calculation for a calculated column, enter the SQL text in the **SQL Text** field. For example, to determine the maximum value of the **SALARY** column in the **EMPLOYEES** table, your syntax would look as follows:

```
MAX( EMPLOYEES . SALARY )
```

Important: If you are defining a calculated column for a virtual table, include the table name in the FROM clause of the SELECT statement (see “Creating Virtual Tables” on page 4-27). In the preceding example, ensure that the **EMPLOYEES** table is included in the FROM clause of the SELECT statement.



Modifying Columns

After you import a database, you might want to modify the imported elements to add more functionality to an application that uses the Model. For example, to include column default values in your application, enter them in the Model Viewer.

To modify a column

1. Select the column in the Model Viewer.
2. Choose **Data Director→Model→Column Properties** from the Visual Basic menu bar.
Alternatively, you can right-click a column in the Model Viewer, and choose **Properties** from the popup menu.
3. Modify the column's properties (see [Figure 4-15 on page 4-34](#)).
For detailed instructions, see [“To create a column” on page 4-33](#).

Working with Relationships

In a relational database, tables can correspond to one another through a relationship. Data Director uses relationships between tables to represent the organization of data in a database and to find associated information stored in your database.

Specifying Master-Detail Records and Lookup Fields

Relationships between tables affect the runtime behavior of Data Director. For example, relationships determine how Data Director coordinates data displayed in master-detail records and lookup data fields.

Origin and Destination Columns

The relationship between two tables is determined by the relationship between specific columns in each of the tables. A relationship starts at the *origin* column and ends at the *destination* column.

For example, the relationship between an **AUTHORS** table and a **TITLES** table is that for every author, there can be one or more titles. In this example, the **AUTHORS** table contains the origin column and the **TITLES** table contains the destination column.

Primary Keys and Foreign Keys

A primary key column uniquely identifies each record stored in a table. A foreign key column is a column in a table that is related to a primary key in a different table.

For example, if the **EMPLOYEES** table has a primary key column of **EMP_ID** and the **ORDERS** table has a related foreign key column **EMP_ID**, the **EMP_ID** column in the **EMPLOYEES** table is used to uniquely identify employees, while the **EMP_ID** column in the **ORDERS** table is used to identify which employee took a particular order for a customer.

At runtime, Data Director uses the value of the primary key column to perform a lookup query on the related table. Records in the related table whose foreign key column's value matches those of the primary key column are included in the result set.

Understanding Relationship Types

Data Director supports the following relationship types:

- One-to-many
- Many-to-one (one-to-one)

One-to-Many Relationships

In a one-to-many relationship, one record in the origin table corresponds to multiple records in the destination table. For example, one relationship between an **ORDERS** table and an **ORDER_LINES** table specifies that each order can contain one or more order lines.

Figure 4-16 shows this relationship.

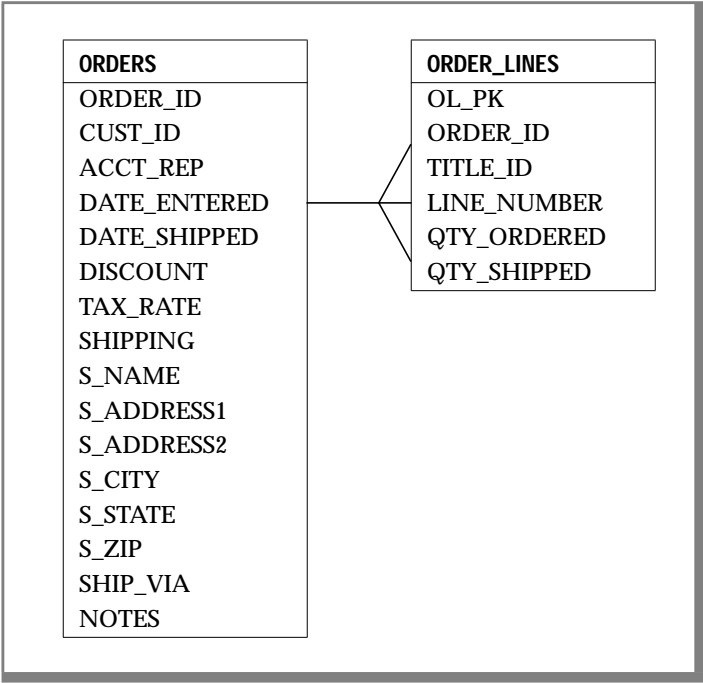


Figure 4-16
A One-to-Many Relationship

Many-to-One Relationships

In a many-to-one relationship, one record in the destination table corresponds to many records in the origin table. For example, multiple orders may be associated with the same customer.

Figure 4-17 shows this relationship.

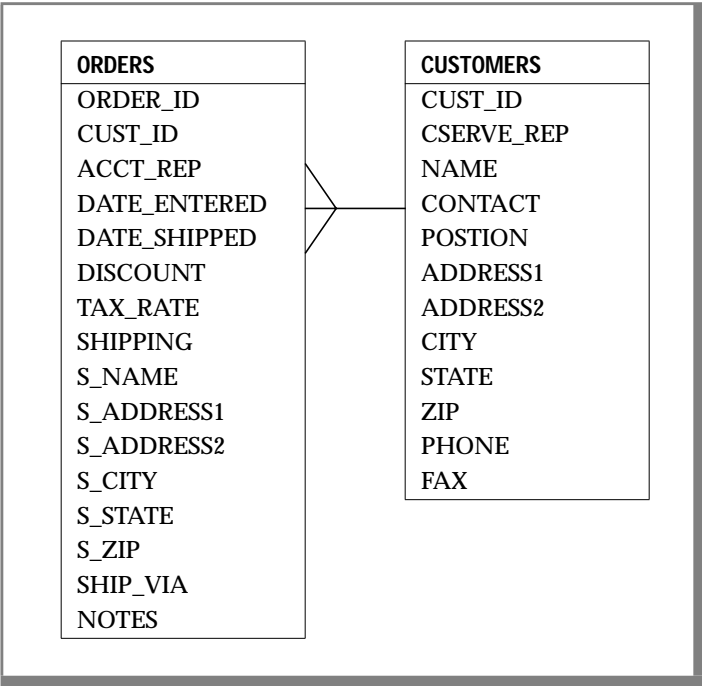


Figure 4-17
A Many-to-One Relationship

Implied Relationships

Relationships are bidirectional. That is, when you create a relationship, an implied relationship exists in the opposite direction (the origin and destination columns switch places). Data Director automatically creates the implied relationship for you.

For example, if you create a relationship in which the **CUSTOMERS** table is the origin table (**CUSTOMERS.CUST_ID**) and the **ORDERS** table is the destination table (**ORDERS.CUST_ID**), Data Director automatically creates the implied relationship in which the **ORDERS** table is the origin table (**ORDERS.CUST_ID**) and the **CUSTOMERS** table is the destination table (**CUSTOMERS.CUST_ID**).

Creating Relationships

Even if you have already imported a database to a Model, you might need to create relationships if you use multiple-column primary keys or if you did not ask Data Director to infer relationships during the import process.

You can also create relationships between virtual tables and physical tables. Use the following procedure to create a relationship. Details for each step are provided in the sections that follow.

To create a relationship

1. In the Model Viewer, select the originating table for which you want to create a relationship.
2. Choose the **Add Relationship** menu item:
 Choose **Insert→Add Relationship** from the Visual Basic menu bar. ♦
 Choose **Project→Add Relationship** from the Visual Basic menu bar. ♦
 Alternatively, you can right-click a table in the Model Viewer, and choose **Properties** from the popup menu.
3. On the **Relationships** page, specify the origin column.
4. Specify the destination table and column.
 Data Director will choose an appropriate relationship type for you based on whether the column is a primary key.

***Tip:** To quickly define a relationship, simply drag a table or a column and drop it onto another table or column. You can drag and drop items to create relationships within the same pane or across panes in the Model Viewer window.*



VB 4

VB 5

The Create Relationship dialog box is shown in [Figure 4-18](#).

Create Relationship

Origin

Table: EMPLOYEES

Columns:

12	EMP_ID	▲
Aa	NAME	
Aa	ADDRESS1	
Aa	ADDRESS2	▼

◀ ▶

Destination

Table: CUSTOMERS

Columns:

12	CUST_ID	▲
Aa	CSERVE_REP	
Aa	NAME	
Aa	CONTACT	▼

◀ ▶

☐ Show Owner Name After Table Name

OK

Cancel

Key Order...

Type

☒ One-to-Many

☐ Many-to-One

Figure 4-18
Creating a Relationship

Specifying the Origin Column

Before choosing the **Add Relationship** menu item, in the Model Viewer, select the column from the origin table that defines the origin side of the relationship.

Specifying Destination Information

Select the table and columns that define the destination side of the relationship. You can select any table and column on the destination side for a new relationship; however, both the destination side and the origin side must contain the same number of columns. In addition, the corresponding columns on both sides of the relationship must be of comparable data types.

Default Relationship Types

Data Director chooses a default relationship type based on primary key and foreign key information in the origin and destination tables.

If neither the origin nor the destination column is a primary key, the default relationship is many-to-one. You can override the default relationship type for the selected origin and destination columns.

Tip: To create a one-to-one relationship, choose many-to-one as the relationship type.



Using Multiple-Column Relationships

A relationship can include any number of columns, as long as both the origin side and the destination side of the relationship include the same number of columns. The order of the columns in a multiple-column relationship determines which column on the origin side is related to which column on the destination side.

For example, suppose your database includes a multiple-column one-to-many relationship in which the **CUSTOMERS** table is the origin table (`CUSTOMERS.CUST_ID`, `CUSTOMERS.NAME`) and the **ORDERS** table is the destination table (`ORDERS.CUST_ID`, `ORDERS.S_NAME`).

If Data Director queries a customer record where `CUSTOMERS.CUST_ID=100` and `CUSTOMERS.NAME='ACORN BOOKS'`, then to query the related orders records, Data Director compares the values of the master record (**CUSTOMERS**) with the detail records (**ORDERS**) to find records where `ORDERS.CUST_ID=100` and `ORDERS.S_NAME='ACORN BOOKS'`.

Tip: Data Director imports multiple-column primary keys, but does not infer multiple-key relationships; if you use multiple-column primary keys, you will need to manually create relationships.



Modifying the Key Order

If multiple columns exist on both sides of a relationship, you can rearrange the key order of the destination columns as they are related to the origin columns.

The order of the columns in a multiple-column relationship determines which column on the origin side is related to which column on the destination side.

For example, if you have two columns on both the origin side and the destination side of a relationship, the key order determines which origin column maps to which destination column. With this example, you could have two mappings between origin columns and destination columns.

The first possible mapping is shown in [Figure 4-19](#).

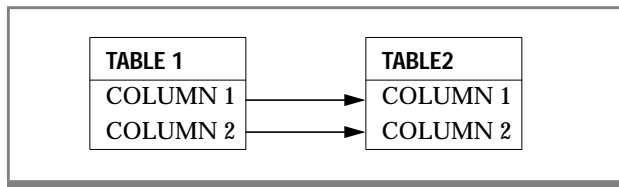


Figure 4-19
*One Mapping
Between Origin and
Destination Columns*

The second possible mapping is shown in [Figure 4-20](#).

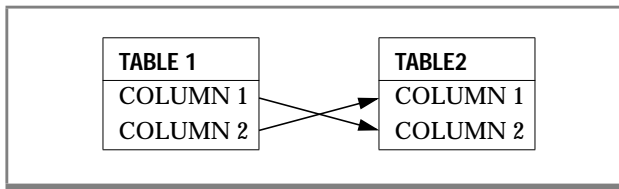


Figure 4-20
*A Second Mapping
Between Origin and
Destination Columns*

To modify the relationship key order

1. In the Model Viewer, select the table whose relationship key order you want to modify.
2. Choose **Data Director→Model→Table Properties** from the Visual Basic menu bar.
3. Click the **Relationships** tab in the Table Properties dialog box.
4. Select the relationship to modify.

5. Click **Modify** on the **Relationships** page.
6. Click **Key Order** in the Modify Relationships dialog box.
7. Change the key order of the origin and destination columns in the relationship.

The Modify Key Order dialog box is shown in [Figure 4-21](#).

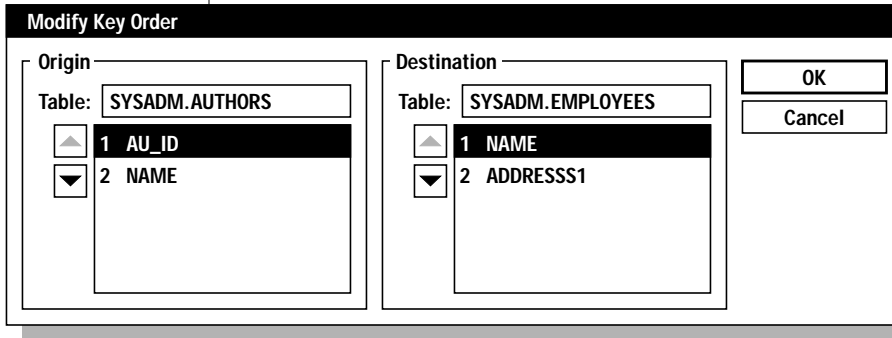


Figure 4-21
Modifying Key Order

Modifying Relationships

After you import a database, you might want to modify the imported elements to add more functionality to an application that uses the Model. For example, you might want to change the destination column information for a relationship or change the relationship type of a relationship.

To modify a relationship

1. In the Model Viewer, select the table whose relationship you want to modify.
2. Choose **Data Director→Model→Table Properties** from the Visual Basic menu bar.
 Alternatively, you can right-click a table in the Model Viewer, and choose **Properties** from the popup menu.
3. Click the **Relationships** tab in the Table Properties dialog box.
4. Select the relationship to modify.
5. Click **Modify** on the **Relationships** page.
6. Change the origin or destination column (or both) and specify the relationship type (one-to-many or many-to-one).

Working with DataGroups

Using DataGroups	5-3
DataGroup Concepts	5-3
DataGroups Reference Models	5-4
DataGroups Organize Data	5-5
DataGroups Use Master Tables for Queries	5-5
About DataGroup Files.	5-6
DataGroups Enable Data-Centric Development	5-6
DataGroups Manage Data Operations	5-7
Concurrency Control Concepts	5-7
Concurrency Control	5-8
Locking	5-8
Read Consistency Concepts	5-9
Using Optimistic Concurrency Control	5-9
When to Use Optimistic Concurrency Control	5-10
Using Optimistic Detect	5-11
Using Optimistic No-Detect	5-13
Using Pessimistic Concurrency Control	5-14
When to Use Pessimistic Locking	5-14
Creating DataGroups.	5-14
Deleting DataGroups.	5-16
Recovering Deleted DataGroups.	5-18
Setting DataGroup Properties	5-19
Setting Query Properties	5-20
Smart Query	5-20
Auto Query.	5-20
Save and Delete Confirmation	5-21
Number of Rows to Fetch	5-21
Parameterized Detail Fetch	5-21
Setting Error-Handling Properties	5-22
Categories of Errors	5-22
Displayed Type	5-23
Message Captions	5-23

Setting Database Properties.	5-24
Query and Login Time-Out Values	5-24
Auto Commit After Every Save	5-24
Setting Connection Properties	5-25
Smart Logon	5-25
Show Logon Dialog	5-25
Override Database and Owner Name	5-26
Setting Concurrency Properties	5-26
Optimistic, No-Detect	5-26
Optimistic, Detect.	5-27
Pessimistic Locking	5-28
Setting Blob Support Properties	5-29
Chunk Size (in Kilobytes)	5-29
Setting Join Properties	5-29
Outer Join Type	5-31
Maximum Number of Outer Joins	5-31
Setting DataGroup Table Properties	5-31
Setting Conditions	5-31
Elements of Condition Statements	5-32
Using Condition Operators	5-32
Setting Data Sorting	5-34
Specifying a Sort Order for Multiple Columns	5-36
Sorting List Box Links	5-36
Setting DataGroup Column Properties	5-37
Specifying a Custom Default Value	5-38
Defining Constants	5-38
Entering Calculation Functions	5-38
Using Multiple DataGroups	5-39

DataGroups enable you to coordinate the use of data across many database tables and application windows. This chapter provides a description of how DataGroups work, how to create them, and how to set properties for DataGroups and their associated tables and columns.

Using DataGroups

DataGroups enable users to complete data usage tasks across an entire application, not just for a single application window. Data usage tasks are the tasks that users typically want to complete when they use an application, such as querying or saving data. The flexibility to query and save data at the application level makes DataGroups very powerful.

This section describes how DataGroups work and how to create and delete them.

DataGroup Concepts

A DataGroup is a collection of related tables that coordinates the use of data across many database tables and many application windows. *Use of data* refers to the tasks that end users typically want to complete when they use an application. These tasks can include querying, browsing, modifying, inserting, saving, and deleting data. DataGroups allow you to complete data-use tasks across your entire application, not just for a specific window.

A DataGroup is actually an interface to a Model. Each DataGroup that you create references a Model and contains a subset of the tables in that Model.

This section describes the important features of DataGroups:

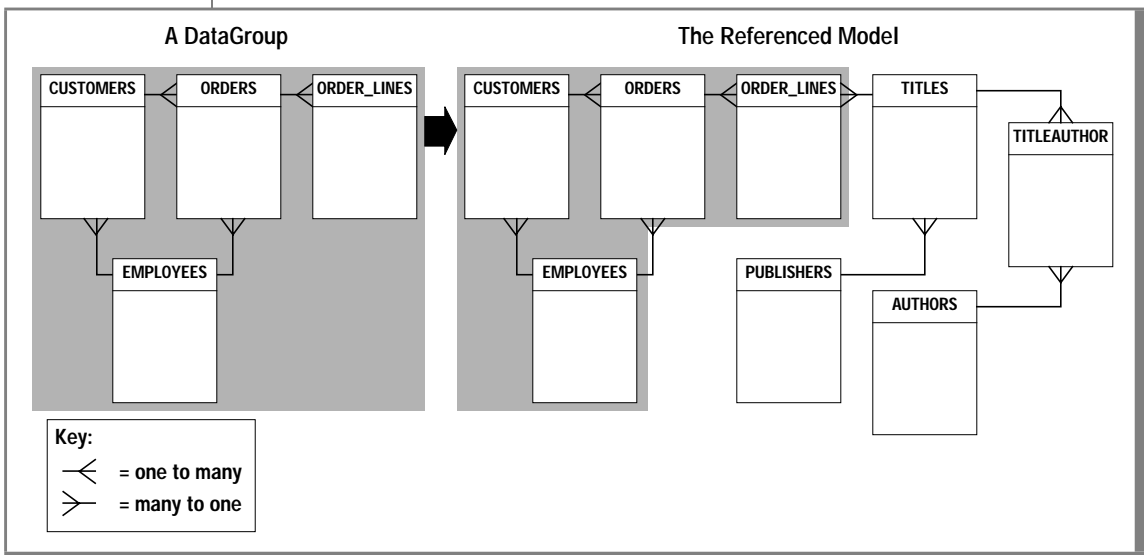
- DataGroups reference Models.
- DataGroups organize data to automate data access and data management features in your application.
- DataGroups use master tables for queries.
- DataGroups have associated text files that can be used for making changes to DataGroups used by multiple projects.
- DataGroups enable data-centric development, so that, for example, data displayed in multiple controls in an application point to the same memory location in which the data is stored.
- DataGroups manage query, new, delete, and save data operations.

DataGroups Reference Models

Each DataGroup that you create references one Model and contains a subset of the tables in that Model. The referenced Model determines which database you want to access.

Figure 5-1 shows a DataGroup and its referenced Model.

Figure 5-1
A DataGroup References a Model





The same Model can be used for more than one DataGroup, and an application can access multiple Models.

Important: DataGroups are completely independent of one another. Think of DataGroups that share a Model as many businesses that share the same office building; except for a common office space, the businesses are not related.

DataGroups Organize Data

DataGroups facilitate, and to some extent automate, data access and data management features in your application. This is accomplished by organizing data into meaningful groups of data and synchronizing the data within each DataGroup.

In addition to being an interface to a Model, a DataGroup stores a group of DataLinks. DataLinks are the connections that you create between database columns and Visual Basic controls; these connections determine which data is displayed in a particular control on a form within an application.

Grouping DataLinks together allows you to access your data at the application level rather than at the window level. This provides your application with a consistent view of data and makes data the focus of your application.

DataLinks are discussed in greater detail in [Chapter 6, “Working with DataLinks.”](#)

DataGroups Use Master Tables for Queries

When you define a DataGroup, you specify one table from the associated Model as the master table. The master table is the table from which Data Director starts queries when retrieving data from the database. If there are DataLinks in a DataGroup that specify additional tables, then those tables are queried after the master table has been queried, based on the relationships between tables.

For example, if a DataGroup corresponding to customer orders specifies the **ORDERS** table as the master table, Data Director first queries the **ORDERS** table. Then Data Director queries any related data, such as the name of the employee who took the order.

Even though multiple DataGroups can share the same Model, each DataGroup can have a unique master table that affects all of its DataLinks. By specifying the first table to query from, the master table allows Data Director to provide master-detail querying, as well as runtime data management. The master table also affects runtime behaviors such as how queried data is sorted and when conditions are applied to a query.

About DataGroup Files

When you create a new DataGroup, Data Director automatically creates a DataGroup (.dgp) file for the DataGroup after you save the project. The DataGroup file is a text file that contains information about the DataGroup for the project, such as the DataGroup name and DataGroup properties. Data Director automatically saves the DataGroup file with the same base name as the DataGroup.

You can use the text DataGroup file with your source code control system to make changes in the DataGroup when building multiple-developer projects. For more information about maintaining a DataGroup file, see [“Working with DataGroup Files”](#) on page 8-11.

DataGroups Enable Data-Centric Development

A DataGroup enables you to build applications using a data-centric development approach. Because DataGroups treat your application as a whole rather than as separate, distinct windows that have no knowledge of one another, you do not need to code complex messaging routines. Instead of creating forms and then making data an attribute of a form, DataGroups let you make a form an attribute of your data. A form becomes a place to view data—if data is displayed in multiple controls in an application, each control points to the same memory location in which the data is stored.

DataGroups Manage Data Operations

To provide temporary storage of data, a DataGroup automatically caches data that is retrieved from a database in Data Director–managed result sets. These result sets can be edited and updated through your application’s user interface or programmatically. A DataGroup also coordinates the retrieval of data based on the relationships specified among the tables in the group.

Data Director manages the following data operations by DataGroup:

- Query
- New
- Save
- Delete

In addition, all transactions are managed by DataGroup.

Concurrency Control Concepts

Concurrency in an application concerns the level of throughput that occurs between the client and the server. Issues include how quickly users can access data from the database and the response time they receive between client/server communication without decreasing the performance of an application.

The problem that developers must address with concurrency is how to control the concurrent access to data by multiple application users while also providing users with a high level of throughput and data integrity.

Concurrency Control

Concurrency control refers to the simultaneous access of the same data in a database by more than one user and the operations the client/server application performs to accommodate this access. If two users attempt to access and change the same record at the same time, both the database server and the client application must work together to help both users reconcile the conflicting operations they are trying to perform.

When a user requests data from a database using Data Director, Data Director creates a copy of that data on the client-side application. When a user needs to save data in the cached result set to the database, a concurrency control issue arises: How to determine when the data on the client side has become stale relative to data on the database side. Data Director assists in determining when the cached data is stale relative to the database data.

For example, suppose two users simultaneously retrieve the same result set of data from the database. Then the first user changes data in the result set and saves the data to the database. After the first user saves data to the database, the second user has stale data because the data in the result set no longer matches the database data from the original query. Data Director concurrency control methods help you detect when data is stale if you want to avoid committing stale data to a database.

Locking

The database-level solution to concurrency issues is locking. A lock is a hold that a database places on a record, and this lock prevents other users from accessing or updating that row until the lock is released. Any user who tries to access or update a row that is locked must wait until the lock is released.

While Data Director provides you with methods of managing concurrency in an application, the database that you use controls locking at the database level.

Read Consistency Concepts

Another concurrency issue is known as *read consistency*, which means the consistency of data as viewed by multiple client applications. For example, if one user fetches a list of employees from a database table, that user will not see any new names that a different user has added until the first user queries the database again. Depending on the method used to fetch result sets of data, updates to existing records made by other users might or might not be visible.

The level of read consistency of data between client and server depends on the techniques used on the client to fetch and cache data. The more data is cached on the client, the less in sync it might be with actual data on the server. If a client application fetches a list of all orders placed today, there is no guarantee that the list will be accurate five minutes later if a lot of orders are being entered into the database.

Data Director brings result-set data to the client from the database through a standard read-only cursor. The self consistency, or stability, of this data relative to other transactions as the result set is fetched can vary, depending on the default transaction level and cursor model of the database. Refer to your database documentation for information on the stability of its read-only cursors.

Using Optimistic Concurrency Control

Optimistic concurrency control maximizes the number of people who can use the same data by assuming that a record will not be updated by more than one user at a time. When a user queries records, no locks are placed on the records. If the user updates information and proceeds to save the changes, the data is queried again to see if anyone has changed the data since it was originally queried. A lock is then placed on a record, and released after data is updated in the database.

A deadlock in an application occurs when two users are trying to access the same record in a database at the same time. The chance of a deadlock in optimistic concurrency control is minimal, because the lock is held for a short period of time, allowing for an optimistic assumption that other users will not need to access or update the data in that period of time. If a conflict does occur, data might be lost and require re-entry if the update is rejected or cancelled. If the update is committed regardless of conflict, newer data might be overwritten with older, stale data.

When to Use Optimistic Concurrency Control

Optimistic concurrency control is the best approach to use when you want high throughput in your application and want to avoid blocking or deadlocking the simultaneous access of data. Optimistic concurrency control is a good approach to use when the likelihood is small that two users will be changing the same data at the same time. This likelihood is generally determined from the nature of the application and the data it uses.

For example, suppose you are designing an order-entry application where multiple users might be entering different orders for the same customer at the same time. Because you do not want users to lock down customer data when they are entering orders for a customer, thereby preventing other users from entering information for the same customer, optimistic concurrency control is a good approach to use.

Data Director provides you with two methods of optimistic concurrency control; optimistic no-detect and optimistic detect. The optimistic detect method also provides you with additional configuration options for how you want Data Director to detect conflicts in changed data.

Using Optimistic Detect

With the optimistic detect method, Data Director compares values in a result set cached on the client side relative to the records in the underlying database when updating data. When a user updates data, Data Director locks a record immediately prior to updating it and releases the lock as soon as the update completes successfully. This method minimizes the amount of time the record is locked.

For example, when a user queries records from the database, Data Director caches the result set on the client-side application. When the user changes data in the cached data and attempts to save changes in the data, Data Director re-fetches the data from the database and places locks on the records fetched. Data Director then compares the records in the cached result set with the records re-queried from the database, performs updates as necessary, and then releases the locks placed on the queried records.

During an update, Data Director first determines whether there are any conflicts between the data in the database and the data on the client side by detecting if data in the database has been updated by another user. If the data has not been changed in the database by another user, Data Director updates records in the database based on the data in the client application. If the data has been changed in the database, Data Director rolls back the data changes and raises an error.

The amount of conflict detection Data Director performs depends on the settings you configure for the methods and granularity of detection.

Concurrency in Data Director is implemented at the DataGroup level. To implement concurrency control, simply choose a concurrency method when you set DataGroup properties (see [“Setting DataGroup Properties” on page 5-18](#)).

Following are some additional concurrency control design considerations:

- Data Director does not compare blob values that might have changed in a result set to the values in a database. This includes both text and image blob values.
- Data Director provides concurrency control for runtime virtual tables if the virtual table is a simple SELECT statement (SELECT *) and if the virtual table has a primary key set at design time.

Methods of Detection

Methods of detection are mechanisms that Data Director uses to detect changes to an underlying database table relative to the data cached on the client side just prior to updating. You can choose from the following detection methods:

- **CRCs (cyclic redundancy codes).** Data Director detects if the cached client rows have become stale relative to the underlying data in the database by comparing CRC values with those for the underlying data.

CRCs are a highly reliable error-detecting codes used to detect transmission errors that can randomly corrupt messages (strings of bits) as they are transmitted around a network. Using a form of binary arithmetic modeled on polynomial arithmetic modulo 2, CRCs can be computed relative to the message and some generator polynomial that renders the probability of a transmission error going undetected almost infinitesimal.
- **Changed row indicator.** Data Director compares a predefined changed row indicator column in the cached result set against the previously fetched value to determine if it has changed. This column is set at design time and is used as an accurate indicator when determining if values in the underlying database have changed. To set the changed row indicator column in your database, see [“Creating Columns” on page 4-33](#).
- **Value.** Data Director compares actual column values that have changed in the result set relative to the column values stored in a database.

Granularity of Detection

Granularity methods of detection concern the amount of data that Data Director compares in a cached result set relative to a database to determine if the underlying data has changed. You can choose from the following methods:

- **Queried/cached columns.** Data Director compares the entire record that is cached on the client side relative to the database when updating records. This does not include columns that have not been changed in the result set, and does not include the entire row, even if the entire row is fetched.
- **Update columns only.** Data Director compares columns that have been updated in the records cached on the client side relative to the database when updating records. This does not include columns that have not changed in the result set, and does not include the entire row, even if the entire row is fetched.
- **Whole row.** When updating records, Data Director compares the entire row that is cached on the client side relative to the database.

Using Optimistic No-Detect

With the optimistic no-detect concurrency control method, Data Director simply sends queries directly to the server to perform data operations. Conflicts are handled by the default, transaction-level concurrency processing defined by your database or ODBC driver. Refer to your database or ODBC driver documentation for more information about the default transaction level of concurrency control.

While optimistic no-detect concurrency control does not provide you with any control over how Data Director detects changes in cached data relative to a database, this method does provide you with a high level of throughput and ensures that records are not locked down in a database.

Using Pessimistic Concurrency Control

Pessimistic locking ensures that multiple users cannot change the same data at the same time. Whenever data is queried, the queried records are locked, preventing others from querying the same data at the same time. Records are unlocked when a user saves data or exits the application window.

While pessimistic locking provides more security for the data in your database, it can also reduce the performance level of your application. For example, if one user attempts to access a record that another user already has open, the second user will probably receive an “access denied” error message, even if that user just wants to view the data, not update it.

One solution is to use a read-only method for most data operations. This reduces the possibility of conflict between those reading data and those changing data. Data Director does not currently provide the ability to implement a read-only method for data access; however, many databases provide this functionality at the transaction level. Refer to your database documentation for more information about how your database handles read stability cursors.

When to Use Pessimistic Locking

Pessimistic locking is appropriate for those situations in which data integrity at all levels (read and write) is imperative. For example, suppose you are building a commissions application where multiple users have the same responsibilities to update commission information for salespersons. Because you would not want multiple users updating the commissions information at the same time, you would want to lock the data each time a user accesses it.

Creating DataGroups

Generally, a DataGroup is based on one business process. The number of processes an application performs usually determines the number of DataGroups in that application. For example, you might define one DataGroup to query data pertaining to order information and one to query data pertaining to customer information.

When you create a DataGroup, you specify the Model of which the DataGroup is a subset and the master table within the Model to use as a starting point for database queries.

To create a DataGroup

VB 4

VB 5

1. Choose the **Add DataGroup** menu item:
Choose **Insert→Add DataGroup** from the Visual Basic menu bar. ♦
Choose **Project→Add DataGroup** from the Visual Basic menu bar. ♦
The Create DataGroup dialog box appears, as shown in [Figure 5-2](#).

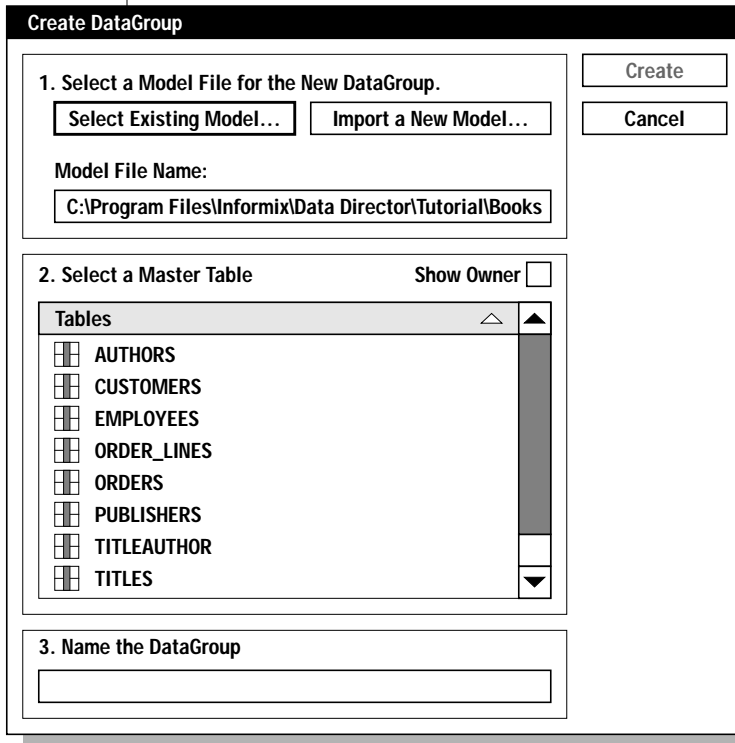


Figure 5-2
Creating a New DataGroup



**Create
DataGroup
button**

2. Alternatively, you can access this dialog box by clicking the **Create DataGroup** button in the DataLink Manager's Model pane.
3. Specify an existing Model for the new DataGroup, or import a database to a new Model.
The same Model can be used for more than one DataGroup, and your application can access multiple Models.
4. Specify one table from the Model as the master table.
5. Accept the default name for the DataGroup, or enter a new name.



Important: Due to Visual Basic control-name requirements, use a letter (A to Z) to begin your DataGroup names. Because Data Director automatically creates a hidden Navigation Control when necessary, Data Director uses the DataGroup name as a prefix and cannot create a control with a name that does not start with a letter.

You can choose any table in a Model as the master table, but you should consider the purpose of an application carefully before selecting a master table. You might think about the master table in terms of a specific business process. Although you are still querying the same body of information (the referenced Model), you can return radically different result sets, depending on the master table.

For example, order entry, customer service, and human resources are three processes that require looking at data very differently. In the order entry process, you would typically query the **ORDERS** table first, and then query any related data (which could include the name of the employee who took the order). In the customer service process, you would typically query the **CUSTOMERS** table first, and then query the order lines associated with a particular customer. In the human resource process, the **EMPLOYEES** table would be queried first, and then any related data.

Deleting DataGroups

You can delete a DataGroup at any time during the development cycle, but keep in mind that deleting a DataGroup deletes all of the DataLinks in the DataGroup.



Warning: Although you can undelete a DataGroup, undeleting a DataGroup for which DataLinks existed does not restore the deleted DataLinks.

To delete a DataGroup

1. Choose **Data Director→DataGroup→Delete** from the Visual Basic menu bar.

The Delete DataGroup dialog box appears, listing all existing DataGroups, as shown in [Figure 5-3](#).

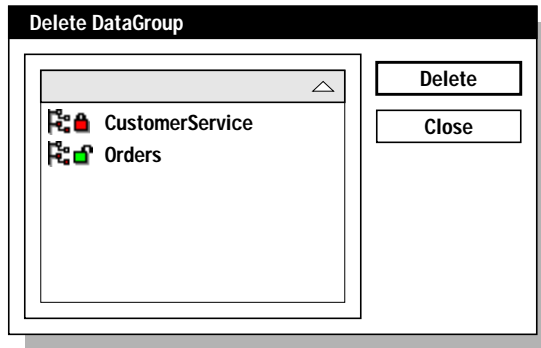


Figure 5-3
Deleting a DataGroup

2. Select the Data Group you want to delete, and click **Delete**.

Warning: *Deleting a DataGroup deletes all of the DataLinks in the DataGroup.*



Recovering Deleted DataGroups

Until you exit a project, you can recover a deleted DataGroup.

To recover a deleted DataGroup

1. Choose **Data Director→DataGroup→Undelete** from the Visual Basic menu bar.

The Undelete DataGroup dialog box appears, listing all previously deleted DataGroups, as shown in [Figure 5-4](#).

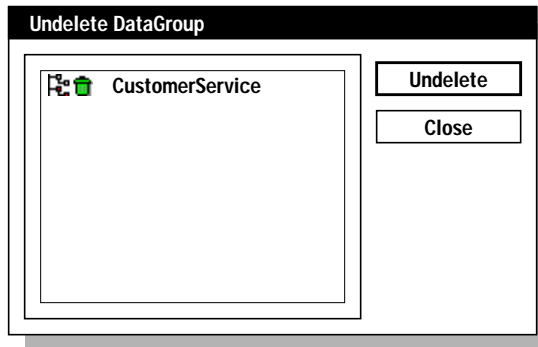


Figure 5-4
Undeleting a DataGroup

2. Select the DataGroup you want to restore and click **Undelete**.

If a DataGroup of the same name already exists, Data Director prompts you to rename the restored DataGroup.

Warning: *Undeleting a DataGroup for which DataLinks existed does not restore the deleted DataLinks.*



Setting DataGroup Properties

DataGroup properties affect the behavior of your application at runtime. You can set DataGroup properties at any time; the settings persist for a project until you change them.

To set DataGroup properties

1. Select the DataGroup that you want to modify in the DataLink Manager.
2. Choose **Data Director→DataGroup→Properties** from the Visual Basic menu bar.

The DataGroup Properties dialog box appears, as shown in [Figure 5-5](#).

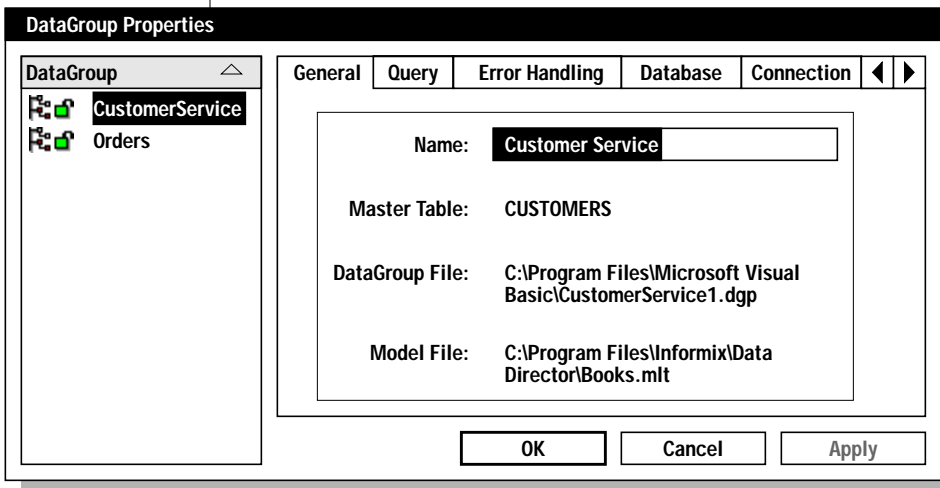


Figure 5-5
*Setting
DataGroup
Properties*

3. Use the tabbed pages to set different kinds of properties, as described in the following sections.

Setting Query Properties

The **Query** page enables you to specify query and confirmation properties for a DataGroup and set the number of rows you want to fetch for master and detail records. [Figure 5-6](#) shows an example of the **Query** page.

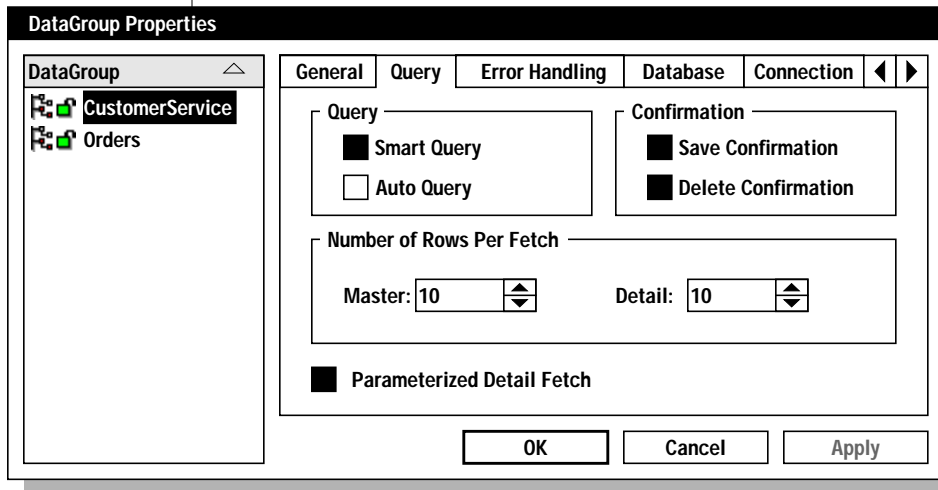


Figure 5-6
Setting Query
Properties

Smart Query

Smart Query ensures that Data Director queries data only when the data can be displayed in an application window. If you do not check **Smart Query**, Data Director queries data regardless of whether any linked controls are visible to display the data.

If your database contains a large number of rows that might be returned by a query, you can use **Smart Query** to increase application performance.

Auto Query

Auto Query tells Data Director to automatically query data for a specific DataGroup when you run your application. The actual query is delayed until a window that contains DataLinks from the DataGroup is opened to display the data.



When this option is selected, the Data Director Logon dialog box automatically appears when the first application window containing DataLinks is opened.

***Tip:** To prevent Data Director from displaying the default Data Director Logon dialog box, call the **LogonInfo** object in your application code before the first application window containing DataLinks is opened.*

Save and Delete Confirmation

Check **Save Confirmation** to prompt application end users to confirm that they want to save changes to a database. Check **Delete Confirmation** to prompt application end users to confirm the deletion of records.

Number of Rows to Fetch

Specify the number of records that you want to fetch from a result set for master and detail records. This number constitutes the number of rows that Data Director fetches internally, not the number of rows visible to a user when a user executes a query.

For example, suppose a result set contains 10,000 records. If you set this option to **10**, Data Director requests 10 records at a time from the database until all 10,000 records have been returned or a user quits the application.

If a quick display of the first row of a result set is important to your application, specify a lower number of rows to fetch. If it is more important that all the data be fetched at once, or if network bandwidth is a consideration, specify a higher number of rows to fetch.

Parameterized Detail Fetch

Check **Parameterized Detail Fetch** to tell Data Director to prepare parameterized queries, if possible. If the application user's ODBC driver supports this option, Data Director uses a prepared statement during querying to enhance the performance of a detail fetch.

Setting Error-Handling Properties

The **Error Handling** page enables you to specify the type of error information that Data Director provides to an end user, as shown in [Figure 5-7](#).

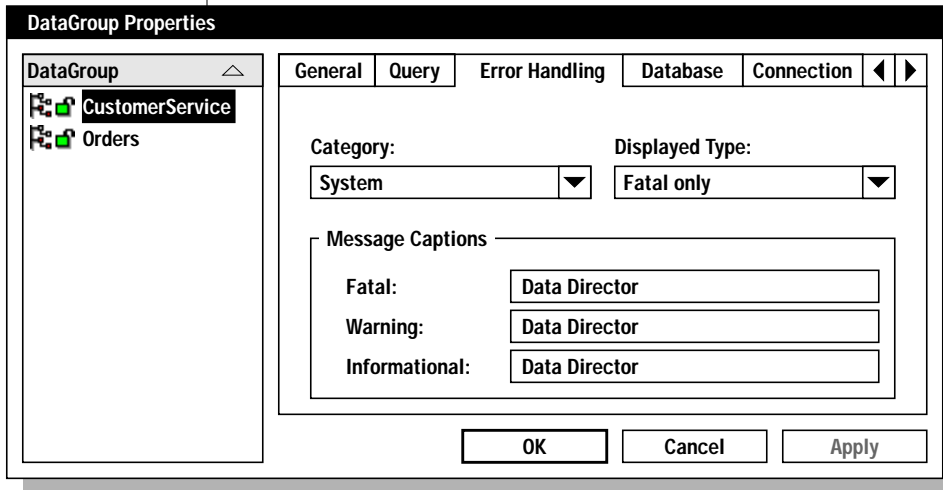


Figure 5-7
*Setting Error
Handling
Properties*

Categories of Errors

Specify the category of errors that you want to customize. For each category that you select, you can choose the type of errors to display and the message caption that appears in the title of the error message box. You can choose from the following categories:

- **System.** Notify users of all system errors. System and resource problems are examples of system errors.
- **Database.** Notify users of all database errors. Problems logging on or executing SQL statements are examples of database errors.
- **Data Director.** Notify users of all Data Director errors. Incorrect syntax in a method call is an example of a Data Director error.

Displayed Type

Select a severity level to restrict the kinds of error messages displayed to end users. The severity level determines when Data Director informs an application end user about errors that occur in an application. For each error category, you can display the following error types:

- **Fatal only.** Notify users of fatal errors only.
- **Fatal and warnings.** Notify users of all warnings and fatal errors.
- **All.** Notify users of all application errors, including informational, warning, and fatal errors.
- **Do not display.** Do not display any type of error.

Message Captions

Message captioning enables you to customize the default text that appears in the title of the error message box when Data Director notifies an end user of an error situation. You can specify a message caption for each combination of category and message types that are displayed to end users.

Setting Database Properties

The **Database** page enables you to specify database-specific properties, as shown in [Figure 5-8](#).

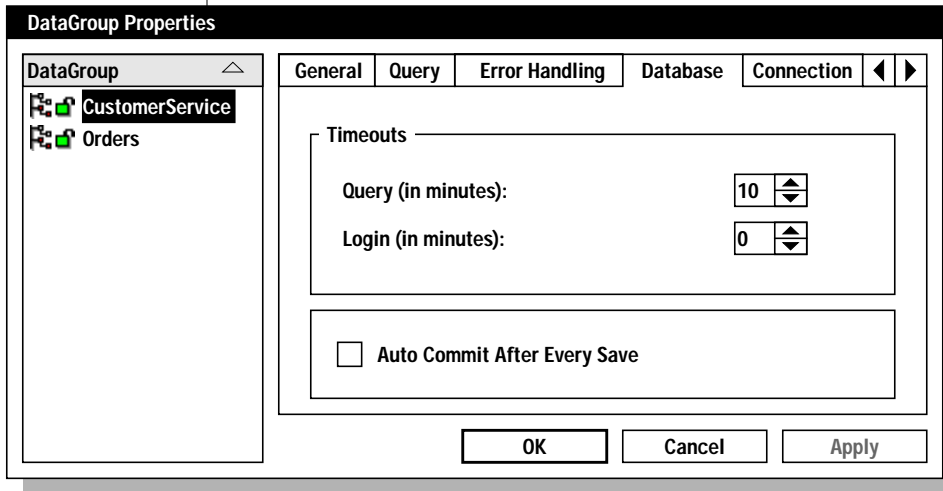


Figure 5-8
Setting
Database
Properties

Query and Login Time-Out Values

Specify the amount of time in seconds that will elapse before a query or a logon attempt times out for a particular DBMS. If you are executing long queries, you might need to increase the query time-out duration for your DBMS.



Tip: The query time-out settings are ignored for DBMSs that do not support query time-out, like the INFORMIX-OnLine databases.

Auto Commit After Every Save

This option ensures that changes in data are automatically committed to the database after a user saves changes.

Setting Connection Properties

The **Connection** page enables you to specify connection or logon information for your database, as shown in [Figure 5-9](#).

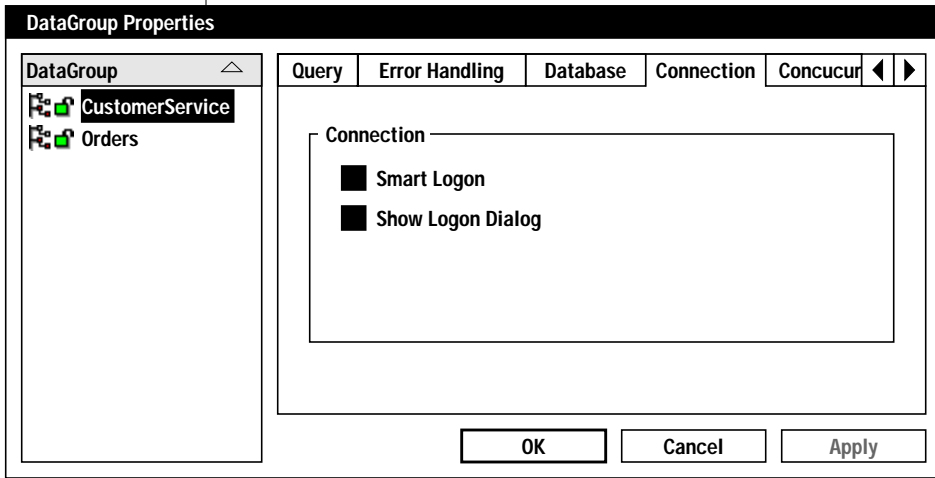


Figure 5-9
Setting
Connection
Properties

Smart Logon

This option tells Data Director to automatically log on to additional databases with the same logon information once an initial database logon is active.

At runtime, **Smart Logon** uses the current user name and password to attempt to log on to each additional database as necessary. If **Smart Logon** fails, the ODBC Logon dialog box appears.

Show Logon Dialog

This option tells Data Director to display the default Logon dialog box when connecting to a database. If you do not choose this option, you need to provide your own connection logon dialog box for application users.

Override Database and Owner Name

These properties enable you to override the database and owner name during logon. These properties are useful for application deployment; they enable you to test the application using a prototype database name and then deploy the application against a production database.

Setting Concurrency Properties

The **Concurrency** page enables you to choose concurrency control properties for a project (see “[Concurrency Control Concepts](#)” on page 5-7). [Figure 5-10](#) shows an example of the **Concurrency** page.

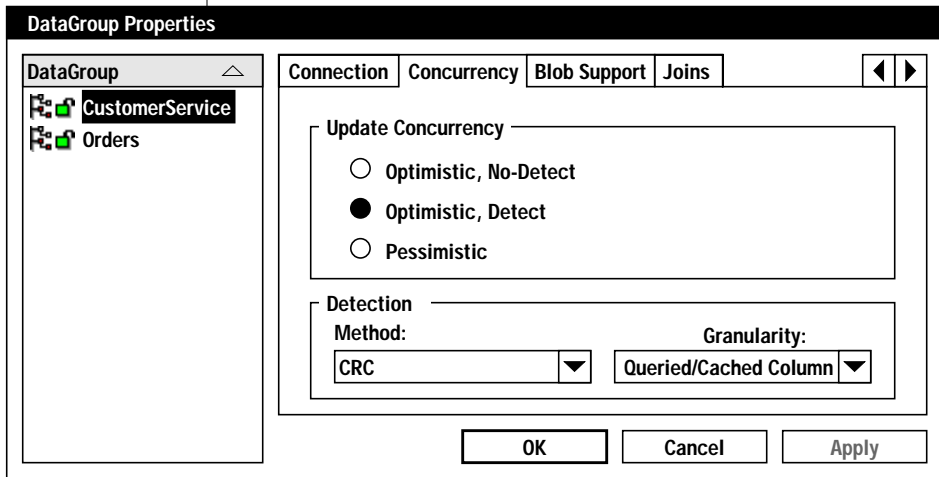


Figure 5-10
*Setting
Concurrency
Properties*

Optimistic, No-Detect

If you use this option, concurrency control does not check for changes in data in a cached result set relative to the data in a database when updating data. During an update, Data Director updates records in the database based on the default locking scheme used by the database.

Optimistic, Detect

If you use this option, concurrency control checks for changes in data in a cached result set relative to the data in a database when updating data. During an update, Data Director first determines whether the row has been updated by another user and then updates records in the database if the data has not been changed. If the data has been changed in the database, Data Director can raise an error and roll back the user's changes; you can override this behavior, if you choose.

Methods of Detection

If you use the **Optimistic, Detect** option, you need to select a method of detection. Methods of detection are mechanisms that Data Director uses to detect changes to an underlying database relative to the data cached on the client side when updating data. Choose from the following methods:

- **CRCs.** Cyclic redundancy codes, or CRCs, are highly reliable error-detecting codes. If you use this method, just prior to updating, Data Director can detect if the cached client rows have become stale relative to the underlying data in the database by comparing CRC values with those for the underlying data.

The CRC method of detection works best when you specify the cached/queried-columns level of granularity.

- **Value.** Data Director takes the changed values of an actual column in the result set and compares them to the column values stored in a database.
- **Changed row indicator.** To determine if a row has changed, Data Director compares a predefined changed row indicator column in the cached result set to the previously fetched value. This column is set at design time and is used as an accurate indicator when determining if a row in the underlying database has changed. To set the changed row indicator column in your database, see [“Creating Columns” on page 4-33](#).

Important: Data Director supports the changed row indicator method of detection only when you specify the whole-row level of granularity.



Granularity of Detection

If you use the **Optimistic, Detect** option, you will also have to specify the granularity of detection. Granularity methods of detection concern the amount of data that Data Director compares in a cached result set to a database to determine if the underlying data has changed when updating data. Choose from the following methods:

- **Cached/queried columns.** When updating records, Data Director compares the entire record that is cached on the client side to the part of the database row queried. This does not include columns that have not been changed in the result set, and it does not include the entire row, even if the entire row has been fetched.
- **Update columns only.** When updating records, Data Director compares columns that have been updated in the records cached on the client side to the database. This does not include columns that have not been changed in the result set, and it does not include the entire row, even if the entire row has been fetched.
- **Whole row.** When updating records, Data Director compares the entire row that is cached on the client side relative to the database.

Pessimistic Locking

Pessimistic locking locks a row of data when it is initially queried and retains those locks for the duration of the transaction (until data is saved or requeried).

Setting Blob Support Properties

The **Blob Support** page enables you to specify the chunk size of cached blob values, as shown in [Figure 5-11](#).

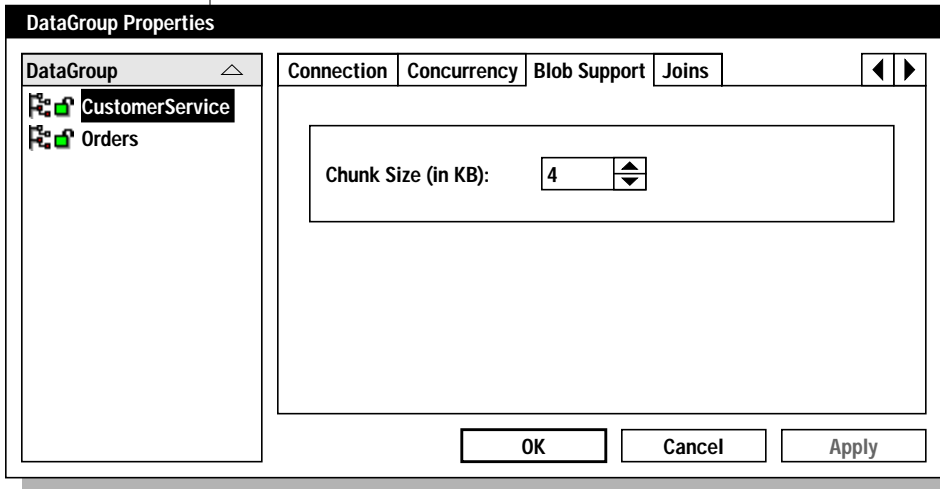


Figure 5-11
Setting Blob
Support
Properties

Chunk Size (in Kilobytes)

This option is useful for performance tuning in your application. For example, if you want to write a blob value that is 22K to the database, but do not want to perform this operation in one step, specify a chunk size of 4K. Data Director will internally send the blob value to the database in incremental values, the first five times sending 4K of the blob value, and the last time sending 2K of the blob value.

Setting Join Properties

The **Joins** page enables you to configure how you want to construct SQL statements for outer joins.

Currently, ODBC supports only one outer join for each SQL SELECT statement. However, some databases support multiple outer joins for each SQL SELECT statement. Choosing a specific database type enables you to take advantage of the native SQL functionality available in your database.

Currently, Data Director supports native outer joins for the following databases:

- INFORMIX-OnLine databases
- Microsoft Access, Microsoft SQL Server, and Sybase

INFORMIX-OnLine databases have no outer-join limit for each SQL SELECT statement and support the following outer join functionality:

- A simple outer join on two tables
- A simple outer join to a third table
- An outer join for a simple join to a third table
- An outer join for an outer join to a third table

All other databases use the outer join support available through ODBC.

Refer to your database documentation for more information about outer join support for your database.

Important: You need to specify a database type for your Model if you want Data Director to generate outer joins for SQL statements (see [“Setting Model Properties” on page 4-23](#)).

Figure 5-12 shows an example of the Joins page.

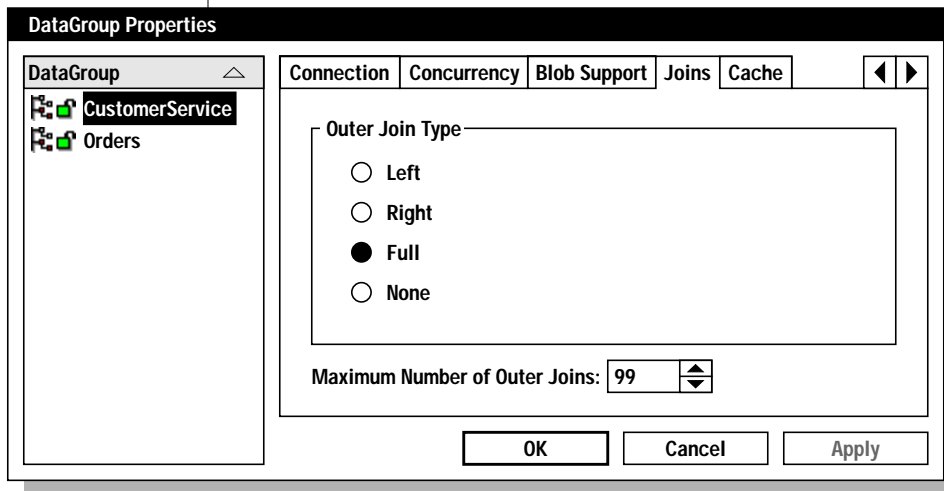


Figure 5-12
Setting
Properties for
Outer Joins

Outer Join Type

Specify the type of outer join that you want Data Director to generate when constructing SQL statements.

Maximum Number of Outer Joins

Specify the maximum number of outer joins that you want Data Director to generate for each SQL SELECT statement. For example, the INFORMIX-OnLine databases have no outer-join support limits.

Setting DataGroup Table Properties

DataGroup table properties enable you to set data conditions and sorting order for tables in a DataGroup.

Setting Conditions

Setting conditions enables you to restrict query results for a table. For example, suppose you want to restrict the results of a query to either customers who are headquartered in California, or customers whose headquarters match the city specified as the shipping site. The following statement satisfies this restriction:

```
customers.state = 'california' or customers.state = orders.s_state
```

You can enter any constructs, SQL-92, or database-specific condition statements.

To set conditions for a table

1. Select the table in the DataGroup pane in the DataLink Manager.
2. Choose **Data Director→DataGroup→Table Properties** from the Visual Basic menu bar.
3. Click the **Conditions** tab in the Table Properties dialog box.
4. Enter a condition statement for the table.

Figure 5-13 shows the **Conditions** page.

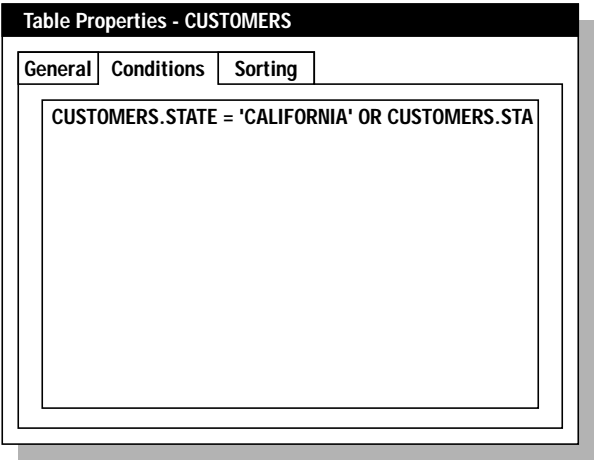


Figure 5-13
Conditions Page

Elements of Condition Statements

A condition statement includes the following elements.

Element	Description
TABLE.COLUMN	Each condition you enter must be preceded by the TABLE.COLUMN specified in the selected DataLink.
One or more logical operators (=, <, >)	To insert a logical operator in a condition statement, enter it in the Conditions field.
One or more condition values	A condition value may be a constant or a TABLE.COLUMN. Constants are case sensitive and must be enclosed in single quotes ('VALUE').

Using Condition Operators

You can combine multiple condition statements using the operators AND and OR. For example, the following condition statement returns only customers having the last name Smith or Jones:

```
CUSTOMERS.NAME = 'SMITH' OR CUSTOMERS.NAME = 'JONES'
```



Tip: Each condition in a multiple condition statement must be preceded by the **TABLE.COLUMN**, and may be compounded only with the operators **AND** or **OR**.

The following table lists some condition operators that you can use.

Operator	Description	Syntax	Returns
=	Is equal to	TABLE.COLUMN = 'value'	Records that match 'value' exactly
< > !=	Is not equal to	TABLE.COLUMN < > 'value' TABLE.COLUMN != 'value'	Records that do not match 'value'
>	Is greater than	TABLE.COLUMN > 'value'	Records greater than 'value'
>=	Is greater than or equal to	TABLE.COLUMN >= 'value'	Records greater than or equal to 'value'
<	Is less than	TABLE.COLUMN < 'value'	Records less than 'value'
<=	Is less than or equal to	TABLE.COLUMN <= 'value'	Records less than or equal to 'value'
AND	Connector	TABLE.COLUMN = 'value1' and TABLE.COLUMN = 'value2' Each condition must be preceded by the TABLE.COLUMN ; you can enter an unlimited number of conditions and parameters.	Records that match both criteria
OR	Either	TABLE.COLUMN = 'value1' or TABLE.COLUMN = 'value2' Each condition must be preceded by the TABLE.COLUMN ; you can enter an unlimited number of conditions and parameters.	Records that match either criteria
IS Null	Null	TABLE.COLUMN is null	Records for which the specified TABLE.COLUMN has no data
IS NOT Null	Not null	TABLE.COLUMN is not null	Records for which the specified TABLE.COLUMN has data

(1 of 2)

Operator	Description	Syntax	Returns
BETWEEN (<i>value1</i>) AND (<i>value2</i>)	Range	TABLE.COLUMN is between ' <i>value1</i> ' and ' <i>value2</i> '	Records within the specified range
NOT BETWEEN (<i>value1</i>) AND (<i>value2</i>)	Range	TABLE.COLUMN is not between ' <i>value1</i> ' and ' <i>value2</i> '	Records outside the specified range
IN (<i>value1</i> , <i>value2</i> ,... <i>valueN</i>)	Values	TABLE.COLUMN is in (' <i>value1</i> ', ' <i>value2</i> ' ... ' <i>valueN</i> ') Each listed value must be enclosed in parentheses.	Records that match one of the listed values
NOT IN (<i>value1</i> , <i>value2</i> ,... <i>valueN</i>)	Values	TABLE.COLUMN is not in (' <i>value1</i> ', ' <i>value2</i> ' ... ' <i>valueN</i> ') Each listed value must be enclosed in parentheses.	Records that match none of the listed values

(2 of 2)

Setting Data Sorting

Data sorting determines the order in which records are sorted for a table when your application displays queried records. The advantage of setting data sorting is that your application can return records in a result set in a logical, organized manner without requiring you to write a lot of code. Data sorting enables users to easily access information in a sorted result set without having to sift through large amounts of unordered information.

Usually, if you do not specify data sorting for a table, data is displayed in the order in which it was entered into the underlying database. This can result in applications that return unmanageable result sets and make it difficult for end users to quickly access the information they need.

To set data sorting for a table

1. Select the table that you want to sort in the DataGroup pane in the DataLink Manager.
2. Choose **Data Director→DataGroup→Table Properties** from the Visual Basic Data Director menu.
3. Click the **Sorting** tab in the Table Properties dialog box.

4. In the **Unsorted Columns** list box, select one or more columns that you want to sort.
5. Click the right arrow to move the columns to the **Sorted Columns** list box.
6. Position the columns in the order in which you would like them sorted.
7. Specify whether you want the columns sorted in ascending or descending order.

Figure 5-14 shows the **Sorting** page.

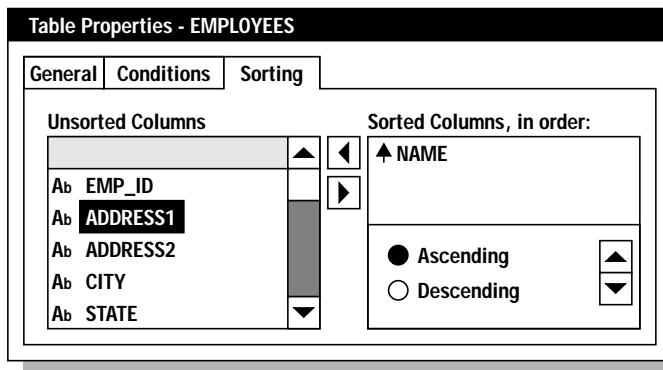


Figure 5-14
Setting Data
Sorting for a Table

Once you set data sorting for a specific table and column, all subsequent DataLinks you create to the same table and column are also automatically sorted in the same manner.

Specifying a Sort Order for Multiple Columns

You can easily sort multiple columns within a table and specify a column sorting order for individual columns in ascending or descending order.

For example, suppose you are designing a customer service application and want to provide users with quick access to the names of their customer contacts. Because it is likely that each customer has multiple entries in the database, you want to sort each customer based on the city where the customer is located and the name of the contact entered for the customer. Based on this criteria, your sorting order would be as follows, using the following tables and columns in your project for the **CUSTOMERS** table.

Sort On	Table and Column
Customer name	CUSTOMERS.NAME
Customer city	CUSTOMERS.CITY
Customer contact	CUSTOMERS.CONTACT

If you did not specify data sorting, your application would display records without a specific order—customer names would not be in alphabetical order, customer records would not be sorted according to their location—users would have a hard time finding the contact name for a specific customer.

Sorting List Box Links

If you want to sort a list box, you must do so through the Data Director **Sorting** option rather than through the Visual Basic **Sorted** property. Using the Data Director **Sorting** option ensures that data value synchronization occurs properly. This also applies to reference links (see [“Creating Reference Links” on page 6-11](#)).

By default, a DataLink is not sorted when you create it unless it is linked to a list box control. If the Visual Basic **Sorted** property is set to **True** before you create a DataLink to a list box, Data Director automatically sets it to **False** and uses the Data Director sorting option (in ascending order) for the table and column specified in the DataLink.

Setting DataGroup Column Properties

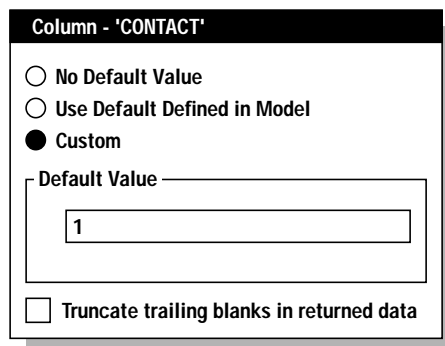
DataGroup column properties enable you to define default values for columns. A default value provides an initial value for new records when an end user runs an application. A default value can be a constant or a function, and persists in an application until an end user changes the value.

If a default value is already specified for a column at the Model level, Data Director automatically uses that default value unless you override or disable the Model default value at the DataGroup level.

To set a default value

1. Select the column in the DataGroup pane in the DataLink Manager.
2. Choose **Data Director→DataGroup→Column Properties** from the Visual Basic Data Director menu.
3. Click the appropriate radio button in the Column Properties dialog box.
4. If you clicked **Custom**, enter a constant or a function as a default value.

Figure 5-15 shows the Column Properties dialog box.



The screenshot shows a dialog box titled "Column - 'CONTACT'". It contains three radio buttons: "No Default Value", "Use Default Defined in Model", and "Custom". The "Custom" radio button is selected. Below the radio buttons is a text box labeled "Default Value" containing the number "1". At the bottom of the dialog box is a checkbox labeled "Truncate trailing blanks in returned data", which is currently unchecked.

Figure 5-15
Setting a Default Column Value

Specifying a Custom Default Value

To specify a custom default value for a column, click **Custom**, and then enter a constant or a calculation function. This value overrides any default value set at the Model level for the column (see [“Setting Default Values” on page 4-35](#)).

Defining Constants

You can enter any constant value as long as it matches the data type of the column. For a CHAR column, a constant value must be less than or equal to the column size.

All characters are valid constants except for the equals (=) and single quote (') symbols. To include either of these symbols in a constant value string, precede the string with a single quote ('). The following table lists some examples.

If You Enter	The Default Value Is
ABC	ABC
' ABC	ABC
' ' ABC	' ABC

Entering Calculation Functions

You can enter the calculation functions listed in the following table. Functions are not case sensitive and do not require an argument, but you must precede a function with an equals sign (=).

Function	Returns	Syntax
Date()	The current local system date	=Date or =Date()
Time()	The current local system time	=Time or =Time()



***Tip:** Some databases that have fixed-length text fields and they automatically pad fields with blank spaces. To display data without trailing spaces, check the **Truncate trailing blanks in the returned data** check box in the DataGroup Column Properties dialog box.*

Using Multiple DataGroups

You can use multiple DataGroups to access different sets of data in databases from a single vendor or in databases from multiple vendors. The data that you access depends on the master table specified for the DataGroup.

Each DataGroup can have a different master table, enabling Data Director to access different views of data from each database. In this example, Data Director accesses two different types of data from the SQL server database: customers data and orders data.

While Data Director synchronizes data within DataGroups, it does not synchronize data across multiple DataGroups.

Working with DataLinks

DataLink Concepts	6-3
DataLinks, DataGroups, and Models	6-4
Standard Visual Basic Control Support.	6-5
Using Bound Controls with Data Director	6-6
Creating DataLinks.	6-6
Creating Standard DataLinks	6-8
Important DataLinks Considerations	6-9
Creating DataLinks to Option Buttons	6-10
Creating DataLinks to Control Arrays	6-10
Creating Reference Links	6-11
Replacing DataLinks	6-12
Deleting DataLinks	6-12
Working with Data Paths.	6-13
Multiple Data Paths	6-13
How Data Director Traverses a Model	6-14
Selecting a Data Path	6-15
Modifying Data Paths	6-16
Viewing Data Path Syntax	6-19
Setting DataLink Properties	6-19
Setting Text Box DataLink Properties	6-19
Setting Option Button DataLink Properties	6-20
Assigning a Data Value.	6-21
Runtime Behavior of Option Button DataLink Properties	6-21
Setting Check Box DataLink Properties	6-21
Using Predefined Data Values	6-22
Using Custom Data Values	6-23
Setting Command Button DataLink Properties	6-23
Enabling or Disabling Command Button Properties	6-24
Changing the Label of the Control	6-24

DataLinks enable you to create connections between database columns and Visual Basic controls in an application window.

This chapter describes DataLinks and explains how to use them.

DataLink Concepts

DataLinks are the connections you create between database columns and Visual Basic controls. A DataLink coordinates data between a database and your application and tells Data Director where data should be displayed in your application.

For example, when you create a DataLink between a text box named **Employee Name** and a database column that stores employee names, the text box displays employee names in a running application.

DataLinks also contain additional information that Data Director uses when it queries data, such as default values and special properties used for DataLinks to certain controls, such as check boxes.

DataLinks, DataGroups, and Models

DataLinks are organized by DataGroups and determine which tables Data Director queries at runtime. When you query data, the DataGroup tells Data Director which master table to start the query from, and the DataLinks in the DataGroup tell Data Director what data to query and in which fields to display the resulting data.

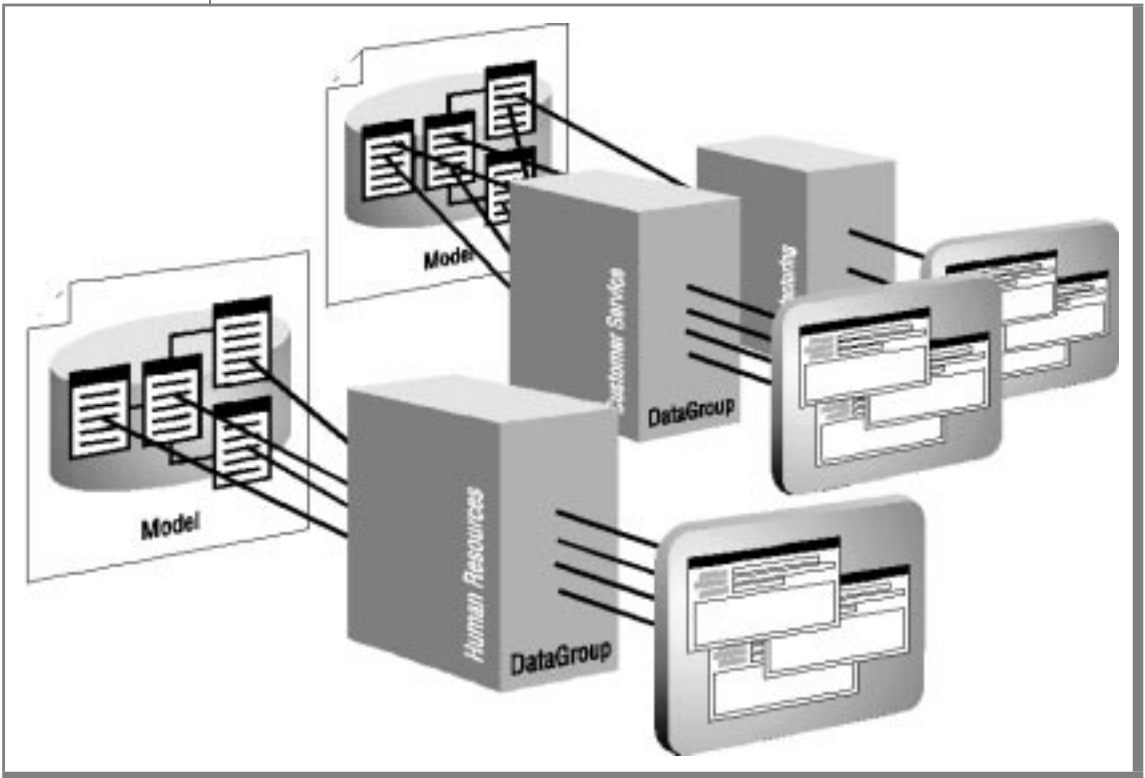
For example, if you are designing a View Customers form and want to see only customer-related information, you probably do not want to query every single table in your Model. For this form, you would create DataLinks to the **CUSTOMERS** table only. If you also want to view each customer's orders, you would create DataLinks to the **ORDERS** table.

Because a DataGroup references a Model, Data Director automatically knows about the relationships in the Model. For example, a one-to-many relationship in a Model tells Data Director to automatically query the detail information for the master records if there are DataLinks to the detail records.

The following process describes the relationship between Models, DataGroups, and DataLinks, as shown in [Figure 6-1](#):

1. You access database columns from your Model, and then link them to controls in your project.
2. A DataGroup saves your DataLinks and queries them together at runtime through a DataGroup.
3. Based on your DataLinks, the controls in an application window display queried data at runtime.

Figure 6-1
Relationship Between Models, DataGroups, and DataLinks



Standard Visual Basic Control Support

Data Director supports DataLinks to most Visual Basic Standard Edition controls, Professional Edition controls, and Enterprise Edition controls.

Data Director supports the following Visual Basic controls:

- Check box
- Combo box
- List box
- Text box
- Command button

- Option button
- DB grid

Using Bound Controls with Data Director

A bound control is a control that you can “bind” to a Visual Basic data control for the retrieval and modification of data. A bound control includes additional Visual Basic properties that non-data-aware controls do not have:

- **DataField**
- **DataSource**

To use bound controls with Data Director, you do not need to create a data control or set any additional properties in the bound control. Instead, you can create a **DataLink** to a bound control just as you would to any other Visual Basic control.

When you create a **DataLink** to a bound control, Data Director automatically sets the **DataField** and **DataSource** properties with the **DataLink** information. At runtime, the control displays data without any additional effort on your part.

For more information about bound controls, refer to your Visual Basic documentation.

Creating DataLinks

Data Director provides you with a graphical user interface that enables you to easily connect database columns to Visual Basic controls.

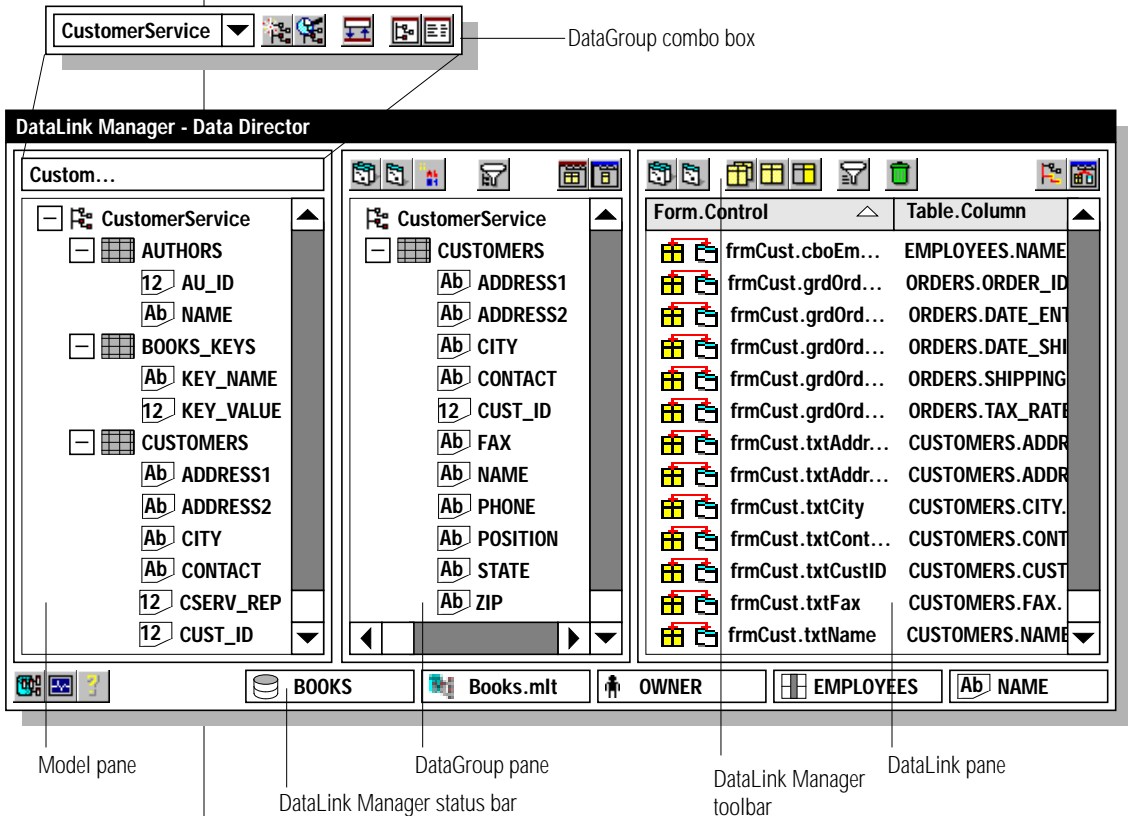
Data Director requires only that you create a Visual Basic form, one or more linkable controls, and at least one **DataGroup** before you can create **DataLinks**.

This section explains how to create **DataLinks** to a variety of controls:

- Standard Visual Basic controls
- Option button controls
- Control arrays
- Reference links

This section also discusses replacing and deleting DataLinks. You use the DataLink Manager, shown in [Figure 6-2](#), to create DataLinks to Visual Basic controls. (For more information about using the DataLink Manager, see [“The DataLink Manager User Interface”](#) on page 1-13.)

Figure 6-2
The DataLink Manager



The features of the DataLink Manager are listed in the following table.

DataLink Manager Feature	Description
DataGroup combo box	Used to select a DataGroup
Model pane	Enables you to work with tables and columns in your Model
The DataGroup pane	Enables you to work with specific tables and columns in a DataGroup
Filtering options controls	Used to set filtering options to customize your view of DataLinks
The DataLink pane	Displays which database tables and columns are linked to which Visual Basic forms and controls

Creating Standard DataLinks

You can create DataLinks to any standard Visual Basic controls. For more information about which Visual Basic controls you can create DataLinks to, see [“Standard Visual Basic Control Support” on page 6-5](#).

To create a DataLink

1. Open the Visual Basic form containing the control to which you want to create a DataLink.
2. Open the Model containing the column you want to link to the Visual Basic control.
3. In the Model pane of the DataLink Manager, expand the table containing the column that you want to link to the Visual Basic control.
4. Select the column that you want to link to the Visual Basic control.
5. Drag and drop the column on the control.

As you drag the DataLink icon over different controls, a blue border appears around controls to which you can link.

Important: If multiple data paths exist for the DataLink being created, Data Director prompts you to select a data path for the DataLink.



The new DataLink and data path for the DataLink appear in the DataLink pane and DataGroup pane.

You can also create a Visual Basic control and a DataLink in a single step, or a single-drop DataLink. When you create a single-drop DataLink, Data Director creates a Visual Basic control, names the control according to the column in the DataLink, creates a DataLink to the control, and creates a label for the column.

To create a single-drop DataLink

1. Select a database column in the Model pane of the DataLink Manager.
2. Drag and drop the column anywhere on a form where a control does not already exist.

You can also double-click a column in the DataLink Manager to create a single-drop DataLink.

Important DataLinks Considerations

Data Director requires specific settings for property values when you create DataLinks to certain controls. In the following cases, Data Director automatically sets the property value when you create the DataLink:

- Data Director automatically sets the **Text** property for text controls such as text boxes and list boxes to the table and column the control is linked to.
- Data Director automatically sets the **Sorted** property for combo box and list box DataLinks to **False**.
- Data Director automatically sets the **MaxLength** property of text boxes linked to columns that have a data type of CHAR or VARCHAR to the field width of the column in the database.

Following are some additional design considerations regarding creating DataLinks:

- Some DataLink and column data type combinations are incompatible. For example, if you link a column of data type CHAR to a scroll bar, your application might exhibit unexpected behavior.

- DataLinks to system-generated key columns must be linked to text box controls.
- If multiple data paths exist for the DataLink being created, Data Director prompts you to select a data path for the DataLink (see [“Selecting a Data Path” on page 6-15](#)).
- If you rename a control or a form, DataLinks to the control or to controls on the form are not renamed. You must delete and re-create the DataLinks.

Creating DataLinks to Option Buttons

DataLinks to option buttons apply to the group of option button controls; Data Director automatically determines which controls constitute one group based on the position of the controls on a form, in a frame, or in a picture box.

To create a DataLink to an option button group

1. Create the option buttons that you want to include in the group using the Visual Basic Toolbox.

Data Director automatically recognizes all the buttons that appear on a form, in a frame, or in a picture box as one option button group.

2. Drag-and-drop columns from the Model pane of the DataLink Manager to one or more of the controls in the option button group.

Creating DataLinks to Control Arrays

A DataLink to a single control in a control array is no different from a DataLink to a single control. The DataLink Manager identifies each control in an array by its unique index number.

To create a DataLink to a control array

1. Create a Visual Basic control array.
2. Drag and drop columns from the Model pane of the DataLink Manager to one or more of the controls in the array.

To distinguish each control in an array, Data Director automatically appends the control name with the appropriate index number in the DataLink list.

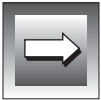
Creating Reference Links

A reference link is a type of DataLink that you use to query lookup data. Lookup data refers to data that appears in a list box—such as lists of employee names.

For example, suppose you are building an application window for entering new order information. To display a comprehensive list of employee names that you want to assign as a customer service representative for each new order, you would query and return all values from the **NAME** column of the **EMPLOYEES** table in the **CS Rep** field. Users can then choose an employee name for the **CS Rep** from a list of all employees when entering an order.

To create a reference link

1. Create a list box using the Visual Basic Toolbox.
2. Drag and drop a column from the Model pane of the DataLink Manager to the list box or combo box control.



Important: When you create a DataLink to a combo box or list box, Data Director automatically sets the **Sorted** property to **False**. If you change the value of the **Sorted** property, the reference group will not update properly.

Data Director automatically creates a reference link when you create a DataLink to a table that is related to its parent table through a many-to-one relationship. If the table specified in a DataLink represents a one-to-many relationship, a standard DataLink is created and Data Director returns only related data based on the current record in the parent table.

When you create a DataLink to a combo box or list box, Data Director checks the data path for the DataLink. If the next-to-last table and last table in the data path are related through a many-to-one relationship, the DataLink is a reference link, and a hidden DataGroup is automatically created. Data Director uses the hidden DataGroup internally to query the reference data for the table and column specified in the reference link—you cannot access this group or modify it.



Tip: If you are not sure which tables are in the data path, double-click the DataLink to open the **Modify Data Path** dialog box.

For more information about how Data Director queries lookup data at runtime and how to refresh lookup data, see [“Using Lookup Data” on page 9-10](#).

Replacing DataLinks

You can replace a DataLink at any time. If you drag and drop a DataLink on a control that is already linked to a database column, the DataLink Manager overwrites the existing DataLink with the new link.

For example, if you inadvertently drag and drop a DataLink on a text box using the wrong database column, simply drag and drop the correct column on the linked text box.

You can choose to display a dialog box that prompts you to confirm whether you want to replace the existing DataLink by setting a DataLink Manager option (see [“Setting DataLink Manager Options” on page 3-3](#)).

Deleting DataLinks

You can delete a DataLink from the DataLink pane at any time while you are designing and building your application but not at runtime. (You can delete DataLinks programmatically at runtime; see the [Data Director Objects Guide](#).) Deleting a DataLink deletes only the link itself, not the control or the database column specified in the DataLink. However, if you delete a linked control, both the DataLink and the object are deleted.

You can choose to display a dialog box that prompts you to confirm whether you want to delete a DataLink by setting a DataLink Manager option (see [“Setting DataLink Manager Options” on page 3-3](#)).

Working with Data Paths

A data path is a series of relationships that Data Director traverses in a Model in order to arrive at the table specified in a DataLink. A DataLink is always saved with a data path that starts at the master table of its DataGroup and ends at the table and column specified in the DataLink. Sometimes, there is only one possible path from the master table to any other table in the Model, and Data Director automatically uses that path when you create a DataLink.

Often however, databases are complex systems that involve many tables and many relationships. As a database becomes more complex, there might be more than one way to access a particular table. When this is the case, Data Director prompts you to specify the correct data path to retrieve the desired data.

Multiple Data Paths

[Figure 6-3](#) shows part of a Model used in an application. The **CUSTOMERS** table is the master table. If you create a DataLink to the **NAME** column of the **EMPLOYEES** table, Data Director prompts you to select a data path because **EMPLOYEES.NAME** has multiple paths. That is, there is a relationship from the **EMPLOYEES** table to the **CUSTOMERS** table as well as from the **EMPLOYEES** table to the **ORDERS** table; resulting in two possible data paths from **CUSTOMERS** to **EMPLOYEES**:

- Path A, from ① to ②
- Path B, from ① to ③ to ②

The **EMPLOYEES** table is a “referenced” table; both the **CUSTOMERS** and the **ORDERS** tables are “referencing” tables. The **Cserve_rep** and **acct_rep** columns are foreign key columns. When you create a DataLink to **EMPLOYEES.NAME**, Data Director detects multiple data paths and prompts you to select a data path.

These tables and their relationships are shown in [Figure 6-3](#).

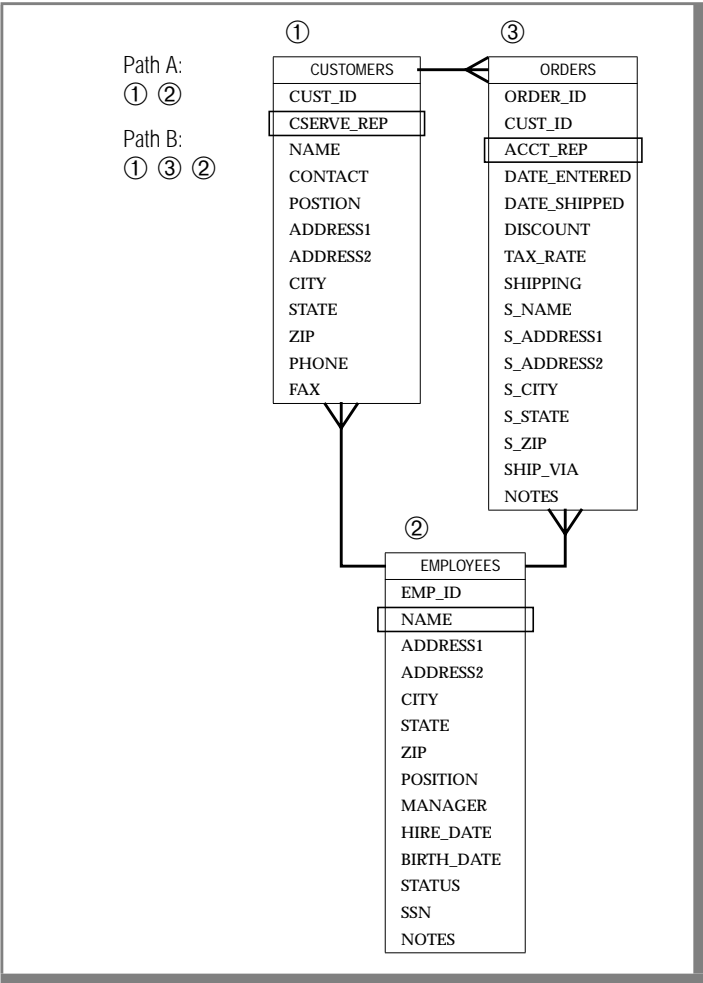


Figure 6-3
*Multiple Data
Paths to the
EMPLOYEES Table*

How Data Director Traverses a Model

To locate the table specified in a DataLink, Data Director starts traversing the Model at the DataGroup's master table, and then traverses through a Model's relationships to get to the correct table. When Data Director reaches the table, the data path to the table and column specified in the DataLink is complete.

A data path describes how to navigate from a DataGroup's master table to the table specified in a DataLink much the same way that a file system path describes how to navigate from the root directory to a file in a subdirectory.

For example, consider how Data Director traverses the Model in the previous example (Figure 6-3) to find a data path:

- Path A is **CUSTOMERS→EMPLOYEES**. If you want to see which employee is assigned to each customer, the data path starts at the **CUSTOMERS** table and ends at the **EMPLOYEES** table.
- Path B is **CUSTOMERS→ORDERS→EMPLOYEES**. If you want to see which employee took each order for a particular customer, the path starts at the **CUSTOMERS** table, then continues to the **ORDERS** table, and ends at the **EMPLOYEES** table.

Selecting a Data Path

When you create a DataLink that has multiple data paths, Data Director prompts you to select a data path. You should select the path that corresponds to the data you want to retrieve. If you select an inappropriate path, your application might display the wrong data. You can change a data path at any time. You can display detailed information for each table in the selected data path, and you can choose different methods of viewing the syntax for the selected data path.

The Data Path viewer graphically displays all possible data paths that start from the master table of the selected DataGroup and end at the table specified in the selected DataLink. This information is based on the relationships defined in your Model. The Path Syntax viewer displays the syntax for the selected data path.

Data Director assists you in selecting a data path. For example, if there is only one possible path to the column in the DataLink, Data Director automatically locates this path for you.

Clicking a table expands your view of the Model. When fully expanded, the Data Path viewer displays all intervening relationships between the master table and the table specified in the selected DataLink. To select a path, click on a succession of tables.



To locate a complete data path, click on a succession of tables until you reach the end-of-path table you want for the data path. An end-of-path table is the last table in a series of tables that, when traversed in order, provides a valid method of accessing the table in the new DataLink.

Tip: The **AutoPath** check box in the View Data Path dialog box enables you to find valid data paths without scrolling through the entire structure of your Model. When the **AutoPath** check box is checked, Data Director searches through the tables and columns in your Model to find the first valid data path. You can then accept this data path for your DataLink or continue to find the next valid data path.

If there are multiple paths, you can specify a selected data path as the default path for other DataLinks to the same table. The existing data path and the data path you select are displayed in different colors. Dead-end paths are paths that Data Director cannot reach to retrieve data.



Tip: If there are multiple paths and you want Data Director to use the same selected data path for other DataLinks to the current table, use the **Use as default** check box to specify the selected data path as the default data path. This prevents you from having to specify data paths when you create other DataLinks to the current table. (This check box is shown in [Figure 6-5 on page 6-18.](#))

Modifying Data Paths

There may be times when you want to change the data path of a DataLink. For example, if you discover that your application is not displaying the data you want, you might want to choose a different path for the DataLink so that it can display the correct data.

To view or modify a data path

1. In the DataLink pane of the DataLink Manager, select the DataLink whose path you want to change.

2. Choose **Data Director→DataLink→Data Path** from the Visual Basic menu bar.

If there is only one possible data path from the master table to the table containing the column in the DataLink, the View Data Path dialog box appears. Because there is only one data path, most options in this dialog box are dimmed. You can copy the path syntax from this dialog box, but you cannot modify the path.

If the selected DataLink has more than one possible path, the Modify Data Path dialog box appears, as shown in [Figure 6-5](#), and you can use it to make your modifications.

3. View or modify the data path:
 - View a data path in the View Data Path dialog box, as shown in [Figure 6-4](#).

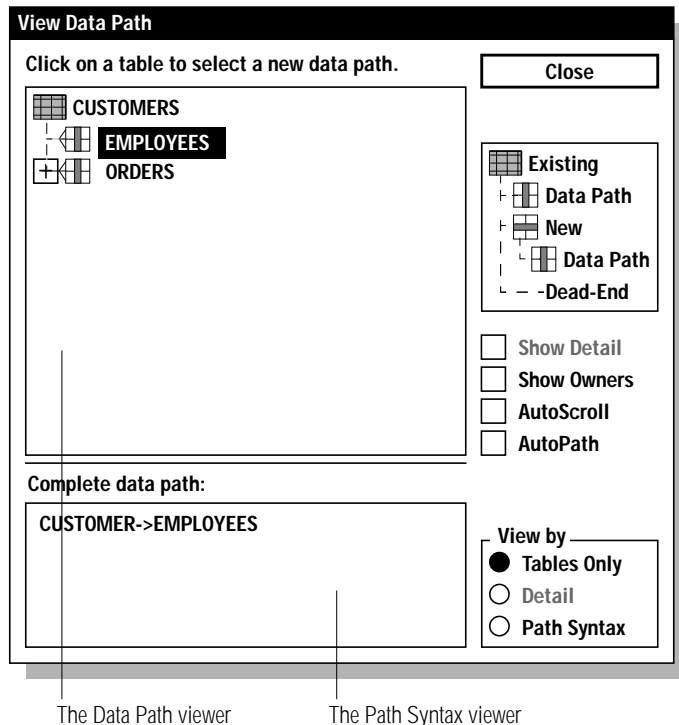


Figure 6-4
Selecting a
Data Path

- Modify a data path in the Modify Data Path dialog box, as shown in [Figure 6-5](#).

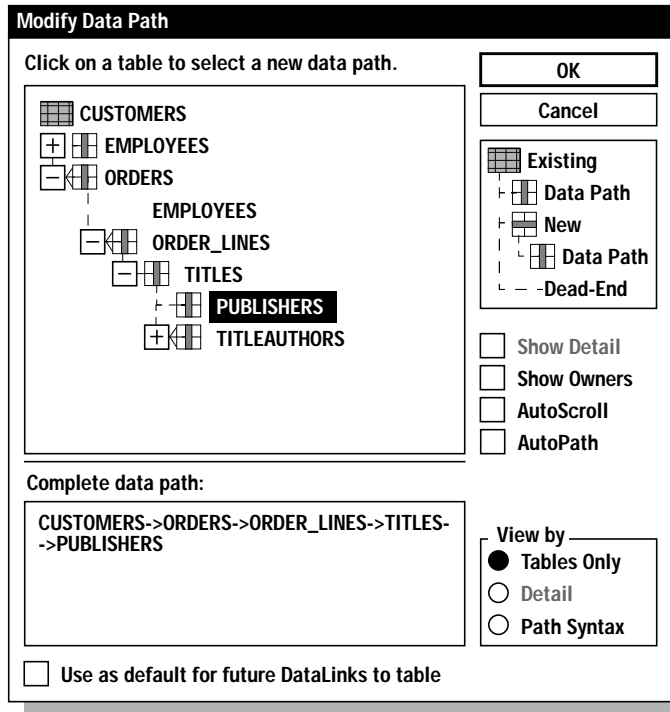


Figure 6-5
Modify Data Path
Dialog Box

The **AutoPath** check box in the Modify Data Path dialog box enables you to find valid data paths without scrolling through the entire structure of your Model. When the **AutoPath** check box is checked, Data Director searches through the tables and columns in your Model to find the first valid data path. You can then accept this data path for your DataLink or continue to find the next valid data path.



Tip: You can also double-click a DataLink in the DataLink pane to view the DataLink's data path.



Viewing Data Path Syntax

You can view the full path syntax of a data path in the View Data Path dialog box. To view the full path syntax for the selected data path, click **Path Syntax** in the **View by** group. Each table in the data path is separated by a back slash (\). The **TABLE.COLUMN** is shown only for the last table in the data path.

***Tip:** You can copy the data path syntax and paste it into your application code (in the **ddoDataGroup.CreateLink** call) when you need to programmatically specify a data path.*

Setting DataLink Properties

DataLink properties affect the runtime behavior of linked controls, such as option buttons and text boxes. Different DataLink types have different properties. For example, a command button DataLink and a text box DataLink have different properties that you can set.

Setting Text Box DataLink Properties

Text box DataLink properties enable you to synchronize events that occur in a running application. (An event is an end-user activity that affects the running application.) When a user modifies data that is displayed in multiple controls, Data Director synchronizes the data in all controls linked to the same table and column within a DataGroup.

For example, if you modify a customer name, this change occurs in all controls that display a customer name.

To set text box DataLink properties

1. Select the DataLink in the DataLink pane of the DataLink Manager.
2. Choose **Data Director→DataLink→Properties** from the Visual Basic menu bar.

3. Complete the DataLink Properties dialog box that appears (shown in [Figure 6-6](#)).

You can synchronize the contents of related controls when the user exits the linked text box or when the user types a value in the linked text box.

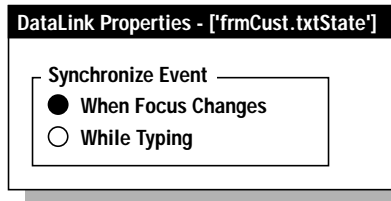


Figure 6-6
*Setting Properties for
Text Box DataLinks*

Setting Option Button DataLink Properties

Option button DataLink properties enable you to assign different data values to option buttons that a user selects in a running application. In a Visual Basic project, option buttons appear in groups and end users can select only one button from a group at a time. Each option button in the linked group is listed according to its Visual Basic name.

To set option button DataLink properties

1. Select the DataLink in the DataLink pane of the DataLink Manager.
2. Choose **Data Director**→**DataLink**→**Properties** from the Visual Basic menu bar.
3. Complete the DataLink Properties dialog box that appears (shown in [Figure 6-7](#)).

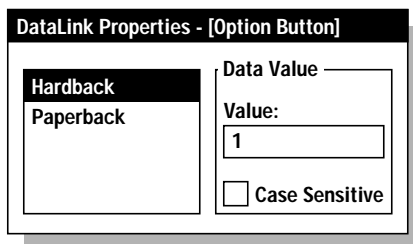


Figure 6-7
*Setting Properties for
Option Button DataLinks*

Assigning a Data Value

To assign a custom data value, select an option button name from the list, and then enter a data value in the **Data Value** text box.

Each button in a group should have a unique data value. You can use predefined data values or supply your own, but you must ensure that the column specified in the DataLink stores data values corresponding to those you assign.

Runtime Behavior of Option Button DataLink Properties

When Data Director queries your database, it compares the saved data value with a data value that you assign to each option button in a group to determine which button is selected. When data is inserted or modified (when a user selects an option button), Data Director saves the button's assigned data value to your database.

For example, suppose your project includes an option button group to track the type of books sold by author:

- Hard cover
- Paperback

You could assign the data values `HC` and `PB` respectively to each option button. If a user clicks the **Hard Cover** option button, the data value `HC` is saved to the column specified in the DataLink (`TITLES.TYPE`); if the user clicks **Paperback**, `PB` is saved.

Setting Check Box DataLink Properties

Check box DataLink properties enable you to assign the data values you want Data Director to use to indicate the state of a check box in a running application. When Data Director queries your database, it compares the saved data value with the data value that you assign to a check box to determine whether it is checked or unchecked. When data is inserted or modified (when a user checks or unchecks a check box), Data Director saves the check box's assigned data value to your database.

To set check box DataLink properties

- 1. Select the DataLink in the DataLink pane of the DataLink Manager.
- 2. Choose **Data Director→DataLink→Properties** from the Visual Basic menu bar.
- 3. Complete the DataLink Properties dialog box that appears (shown in [Figure 6-8](#)).

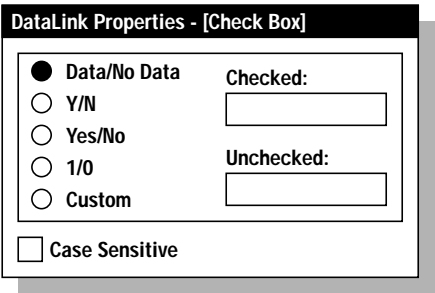


Figure 6-8
*Setting Properties for
Check Box DataLinks*



Tip: For check box DataLinks, you assign two data values: one to indicate when a check box is checked, and another to indicate when it is unchecked.

Using Predefined Data Values

You can use the following predefined data values that Data Director provides.

Data Value	Description
Data/No Data	Saves the current column value when the check box is checked, and saves no value (Null) when the check box is unchecked
Y/N	Saves the value Y when the check box is checked, and N when the check box is unchecked
Yes/No	Saves Yes when the check box is checked, and No when the check box is unchecked
1/0	Saves the number one (1) when the check box is checked, and zero (0) when the check box is unchecked

Using Custom Data Values

To specify custom data values, click **Custom** and enter the values in the **Checked** and **Unchecked** text boxes.

You can use data values predefined by Data Director or supply your own, but you must ensure that the column specified in the DataLink stores the same data values as those you assign.

You can enter custom data values of any type and length as long as the value is compatible with the data type of the column specified in the DataLink.

For example, suppose your project includes the following three check boxes to allow users to specify the cross-platform compatibility they require:

- MS DOS
- Macintosh
- MS Windows

The corresponding check box DataLinks might be **PLATFORMS.DOS**, **PLATFORMS.MAC**, and **PLATFORMS.WIN**. You can assign a custom data value to each check box (such as **DOS**, **MAC**, and **WIN**) to indicate that the corresponding check box is checked, and use an empty string value to indicate that the corresponding check box is unchecked.

When Data Director queries your database, it uses these assigned data values to determine whether a check box is checked or unchecked. If a user checks the **MS DOS** and **MS Windows** check boxes, Data Director saves the data values **DOS** and **WIN** to the **DOS** and **WIN** columns; no value is saved to the **MAC** column.

Setting Command Button DataLink Properties

Command button DataLink properties enable you to control the behavior of a linked command button, depending on the current value of the column specified in the DataLink. Specifically, you can control the following properties:

- Enable or disable a command button
- Change the label of a command button

You can control whether a command button is enabled or disabled based on data values that you specify, or on the presence or absence of any data value.

You can also specify whether you want the label on the command button to change to display the current value of the column specified in the DataLink.

To set command button properties

1. Select the DataLink in the DataLink pane of the DataLink Manager.
2. Choose **Data Director→DataLink→Properties** from the Visual Basic menu bar.
3. Complete the DataLink Properties dialog box that appears (shown in [Figure 6-9](#)).

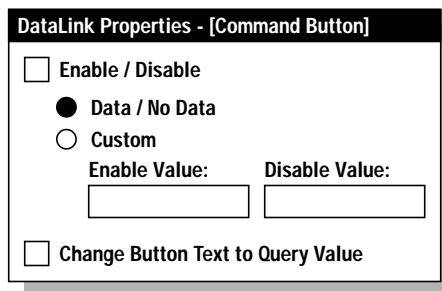


Figure 6-9
*Setting Properties
for Command
Button DataLinks*

Enabling or Disabling Command Button Properties

To specify data values that determine whether a linked command button is enabled or disabled, check **Enable/Disable**. You then have the following two options:

- To enable a linked command button when a column value is present, and disable the command button when the column value is Null, click **Data/No Data**.
- To enter custom **Enable/Disable** values, click **Custom**.

Enter the data values you want Data Director to use to determine whether to enable or disable the linked command button in the **Enable Value** and **Disable Value** text boxes. You can enter custom values of any type and length as long as the values are compatible with the data type of the column specified in the DataLink.

Changing the Label of the Control

To make the label of the linked command button reflect the current data value, check **Change Button Text to Query Value**.

Using the Navigation Control

Overview of the Navigation Control	7-5
Adding the Navigation Control to a Project	7-6
Adding the Navigation Control to a Form	7-7
Customizing the Navigation Control.	7-7
Setting General Properties	7-7
Assigning a DataGroup and Table	7-8
Specifying a Layout Mode Option	7-8
Setting Control Selection Properties.	7-9
Adding Navigation Buttons	7-10
Adding Bookmark Buttons	7-11
Adding a Status Panel	7-11
Adding a Custom Caption	7-11
Adding Action Buttons.	7-12
Setting Action Button Properties Programmatically	7-13
Adding a Custom Button	7-14
Setting Control Appearance Properties	7-15
Setting Font Properties	7-16
Setting Color Properties	7-17
Initializing an Application	7-18
Using Multiple Navigation Controls	7-19
Writing Event Procedures	7-19
Navigation Control Events.	7-20
Intercepting Events	7-21
AfterDataChanged	7-21
AfterDelete	7-22
AfterEnterQBE	7-22
AfterExitQBE	7-23
AfterFetchBlob	7-23
AfterGotoRecord	7-24

AfterLogon	7-24
AfterNewRecord	7-25
AfterQuery	7-26
AfterSave	7-26
BeforeDataChanged	7-27
BeforeDelete	7-28
BeforeEnterQBE	7-28
BeforeExitQBE	7-29
BeforeFetchBlob.	7-30
BeforeGotoRecord	7-31
BeforeLogon	7-32
BeforeNewRecord	7-33
BeforeQuery	7-33
BeforeSave	7-34
CustomEvent	7-35
Error	7-36
OptimisticDetectChanged	7-37
Navigation Control Events Quick Reference	7-38
Navigation Control Properties	7-40
BookmarkButtonsVisible Property	7-40
ButtonStyle Property	7-41
Cancel Property.	7-42
DataGroup Property	7-43
DefaultButtonSize Property	7-44
FrameStyle Property	7-45
LayoutMode Property	7-46
ModelFile Property	7-47
NavigationButtonsVisible Property	7-47
OptimisticDetect Property	7-48
PreferSingleGroupPerLine Property.	7-49
SpacingHorizontal Property	7-50
SpacingVertical Property.	7-50
StatusBackColor Property	7-51
StatusCaption Property	7-51
StatusForeColor Property	7-53
StatusVisible Property	7-53
Table Property	7-54

Navigation Control Properties Quick Reference	7-55
Navigation Control Objects	7-57
CustomButton1 Object	7-57
DeleteButton Object	7-58
NewButton Object	7-59
QBEButton Object	7-61
QueryButton Object	7-62
SaveButton Object	7-63
Navigation Control Objects Quick Reference	7-65

In addition to supporting many Visual Basic and other third-party custom controls, Data Director provides its own custom control, the Data Director Navigation Control. The Navigation Control enables users to navigate through a result set, insert bookmarks, perform QBE queries, and insert, update, and delete records through a visual interface. This chapter describes the Navigation Control and explains how to use it.

Overview of the Navigation Control

A custom control is an extension to the Visual Basic Toolbox. Once added to a project, a custom control is integrated with the Visual Basic environment and provides additional functionality to your application. You can add the Data Director Navigation Control to a project and use it just as you would any native Visual Basic control.

The Navigation Control is a custom control that allows you to implement data access and navigation in an application without programming. At runtime, end users can use the Navigation Control to navigate through a result set, insert bookmarks, perform queries, and insert, update, and delete records through a visual interface. It provides you with a number of data events that allow you to trap specific operations that occur in a running application, and it is easily customized through the Visual Basic Properties dialog box and the Navigation Control Properties dialog box.

The Navigation Control provides the following functionality:

- Initializes your application's use of Data Director
- Specifies which DataGroup and database table you want to navigate
- Provides users with navigation, bookmark, and data operation controls
- Includes buttons that you can customize for your application

- Includes a status display that indicates the table being queried and the record number currently displayed
- Enables you to trap data events

Adding the Navigation Control to a Project

When you install Data Director, the Navigation Control is automatically added to the Visual Basic **auto32ld.vbp** file. This enables Visual Basic to automatically load the Navigation Control into the Visual Basic Toolbox for every new project that you create.

The Navigation Control is a required component of Data Director and enables the data binding functionality in an application. If the Navigation Control is not available for a project, you should ensure that it is added to the project before designing and running a Data Director application.

To add the Navigation Control to a project

1. Open your project in Visual Basic.
2. Add the Navigation Control to the Visual Basic Toolbox:
Choose **Tools→Custom Controls** from the Visual Basic menu bar. ♦
Choose **Project→Components** from the Visual Basic menu bar. ♦
3. Make sure the **Informix Navigation Control** check box is checked, and click **OK**.

The Navigation Control is added to the Visual Basic Toolbox.

Tip: If the **Informix Navigation Control** option is not available, click **Browse** and add the **navctl32.ocx** file located in the directory in which Data Director is installed; for example, **C:\Program Files\Informix\Data Director**.

VB 4

VB 5



Adding the Navigation Control to a Form

You create a Navigation Control the same way you create all other Visual Basic controls. As long as the **navctl32.ocx** file is included in your project, you can create a Navigation Control at any time.

To add the Navigation Control to a form

1. Double-click the **Navigation Control** button in the Visual Basic Toolbox.

The Navigation Control appears on your form.



Navigation Control
button

Customizing the Navigation Control

You can customize the appearance and functionality of a Navigation Control by using the Data Director Navigation Control Properties dialog box and the Visual Basic Properties dialog box. You can also set Navigation Control properties programmatically.

The Navigation Control includes standard Visual Basic properties as well as additional Data Director-specific design-time and runtime properties. These properties are discussed in detail in [“Navigation Control Properties” on page 7-40](#).

To access Data Director Navigation Control properties

1. Select the Navigation Control on a form.
2. Right-click the control.
3. Choose **Properties** from the popup menu.

Setting General Properties

The **General** page enables you to assign a DataGroup and table to a Navigation Control and specify a layout option. The **General** page is shown in Figure 7-1.

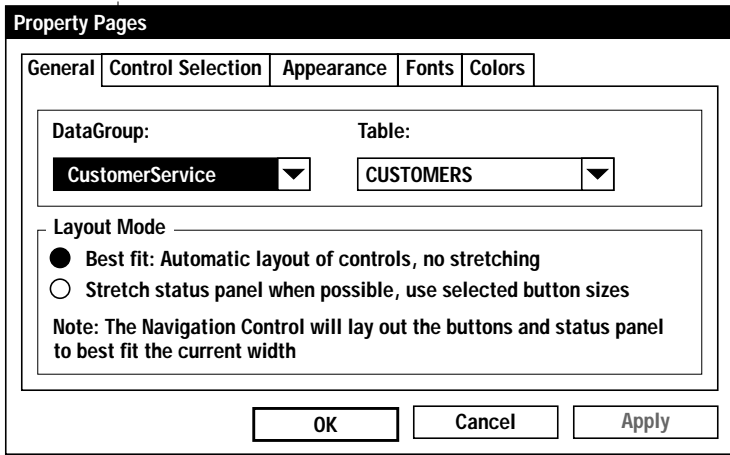


Figure 7-1
Setting General Properties

Assigning a DataGroup and Table

Assigning a DataGroup and table to a Navigation Control enables users to access specific records in the database. For example, if you are designing a Customer Service form, you would assign the **CustomerService** DataGroup and the **CUSTOMERS** table so that users can access information that pertains to customers.

Specifying a Layout Mode Option

Layout options enable you to customize how you want to lay out the Navigation Control on an application form:

- **Best fit.** The Navigation Control lays out items on the control to fit within the boundaries defined by the control's sizing rectangle. While you cannot change the width of the Navigation Control by using this option, the Navigation Control will optimize the use of the space available on the control as you change items on the control.
- **Stretch status panel.** The Navigation Control sizes the status panel on the control as you change the width of the control's sizing rectangle while retaining the size of buttons on the Navigation Control.

If at runtime you programmatically change any of the display properties of the Navigation Control, such as changing button sizes or hiding buttons, you can force the Navigation Control layout to be updated so that your changes will take effect. Use the **UpdateLayout** method of the Navigation Control to update the layout after all programmatic display changes are made.

If at runtime you programmatically change any of the display properties of the Navigation Control, such as changing button sizes or hiding buttons, you can force the Navigation Control layout to be updated so that your changes will take effect. Use the **UpdateLayout** method of the Navigation Control to update the layout after all programmatic display changes are made.

Syntax

```
[form.]NavigationControl.UpdateLayout
```

Example

```
NavControl1.NavigationButtonsVisible = False  
NavControl1.UpdateLayout
```

Setting Control Selection Properties

The **Control Selection** page enables you to choose which buttons you want to have appear on a Navigation Control. You can also customize the text or icon that appears on labeled buttons on the Navigation Control.

The **Control Selection** page is shown in Figure 7-2.

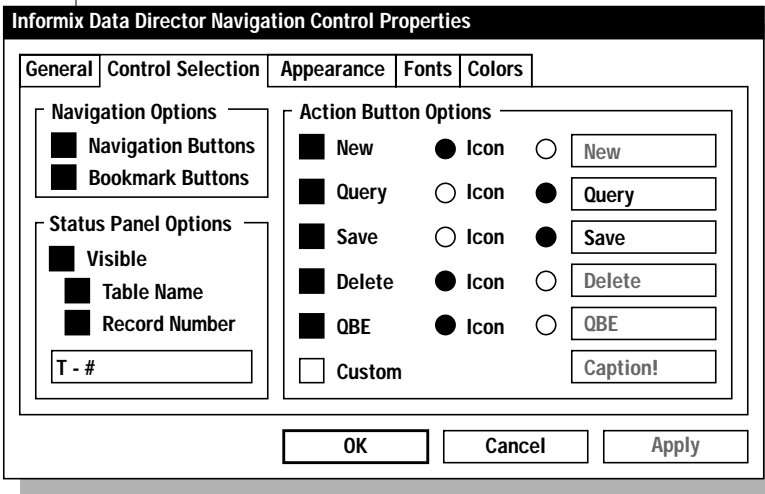


Figure 7-2
*Setting Control
Selection Properties*




Adding Navigation Buttons

Navigation buttons enable users to easily navigate through the records in a result set. Each navigation button is described in the following table.

Icon	Button	Description
	First Record	Navigates to the first record in the result set
	Previous Record	Navigates to the previous record in the result set relative to the current record
	Next Record	Navigates to the next record in the result set relative to the current record
	Last Record	Navigates to the last record in the result set

Adding Bookmark Buttons

Bookmark buttons enable users to set bookmarks on records and navigate through bookmarked records. Each bookmark button is described in the following table.

Icon	Button	Description
	Set Bookmark	Sets a bookmark on the current record The Set Bookmark icon is a toggle switch; clicking this button when the current record is already bookmarked removes the bookmark. This button appears pressed in each time you navigate to a bookmarked record.
	Previous Bookmark	Navigates to the previous bookmarked record relative to the current record
	Next Bookmark	Navigates to the next bookmarked record relative to the current record

Adding a Status Panel

The status panel enables users to identify information about the current record displayed in an application window. You can customize the status panel to display the table name, current record number, or both.

Adding a Custom Caption

To add a custom caption that appears before the table name or record number on the status panel, type the caption before the T or # symbol that appears in the text box at the bottom of the **Status Panel Options** group. T corresponds to table name and # corresponds to record number.

For example, if the current table name is **CUSTOMERS** and the current record number is 15, you can achieve a variety of results based on what you specify as the syntax for this property.



The following table shows a few examples of how you can set the status caption.




Status Caption	Displays
T - #	CUSTOMERS - 15
Table: T	Table: CUSTOMERS
Record: #	Record: 15
Updating Database	Updating Database

Adding Action Buttons

Action buttons enable users to perform operations on database records, such as querying or creating data. You can display action buttons as icons or labeled buttons, customize the text that appears on the labeled buttons, and define your own custom button.

Each action button is described in the following table.

Icon	Action Button	Description
Custom Image	Custom	Enables you to programmatically trap an event and perform a custom data operation
	Delete	Deletes the current record from the database
	New	Creates a new record in the database

Icon	Action Button	Description
	QBE	Switches the application to Query-by-Example (QBE) mode and allows the user to enter search values into any editable field of the form window that uses the DataGroup specified for the Navigation Control
	Query	Queries records from the database; queries all data for the Navigation Control's DataGroup, starting with the master table
	Save	Saves any data that has changed since the last query or save operation for the Navigation Control's DataGroup and table

(2 of 2)

Setting Action Button Properties Programmatically

The following code example shows how to programmatically set properties for the action buttons. Based on the logon ID used to log on to a database, certain action buttons are enabled on the Navigation Control.

In this example, if a user logs on using the logon ID of **ADMIN**, that user is able to query, insert, update, and delete records from the database. Any other logon enables the user to query information from the database only; those users cannot insert, update, or delete records from the database.

Tip: *These properties are not available in the Visual Basic Properties dialog box.*



```
Private Sub ctlAuthNav_BeforeLogon(Username as String)
    navObj = frmBooks.ctlAuthNav
    Case Select Username
        case "ADMIN"
            'show all Action Buttons
            With navObj
                .DeleteButton.Visible = True
                .NewButton.Visible = True
                .QBEBUTTON.Visible = True
                .QueryButton.Visible = True
                .SaveButton.Visible = True
            End With
        case Else
            'hide all non-Query Action Buttons
            With navObj
                .DeleteButton.Visible = False
                .NewButton.Visible = False
                .QBEBUTTON.Visible = True
                .QueryButton.Visible = True
                .SaveButton.Visible = False
            End With
        End Select 'UserName
    End Sub 'ctlAuthNav_BeforeLogon
```

Adding a Custom Button

You can add a custom button that enables you to trap the **CustomEvent** event and perform a special operation based on that event. For example, if you want to launch a calculator program when a user clicks a **Calculator** button on a Navigation Control, perform the following steps.

To add a Calculator button to the Navigation Control

1. Check the **Custom** check box on the **Control Selection** page, shown in [Figure 7-2 on page 7-10](#).
2. Enter a caption for the custom button, such as **Calculator**.
3. Open the code window for the Navigation Control.
4. Select **CustomEvent** from the **Procedures** list box.

5. Add the following code:

```
Private Sub NavControl1_CustomEvent(Table_as_String)
    Shell "CALC.EXE", 1 ' Run Calculator.
End Sub
```

When a user clicks the **Calculator** button in a running application, the Navigation Control automatically launches the **calculator** program.

Setting Control Appearance Properties

The **Appearance** page enables you to easily customize the appearance of the Navigation Control on an application form. For example, you can customize how buttons and the panel surrounding the buttons appear on the Navigation Control.

The **Appearance** page is shown in Figure 7-3.

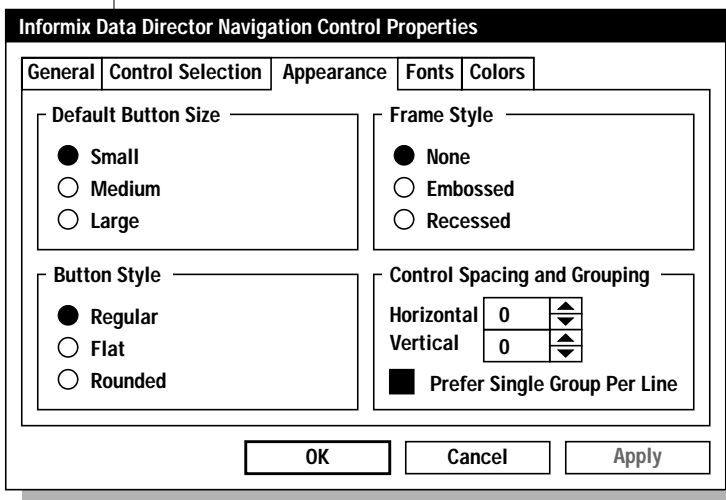


Figure 7-3
Setting Control
Appearance Properties

The **Button Style** and **Frame Style** options work together. Try mixing and matching styles to find the look you like best for the Navigation Control.

You can specify how much horizontal and vertical space you want between items on the Navigation Control. You can also choose to group like items together on a single line of the Navigation Control. For example, you can group all navigation buttons, bookmark buttons, and action buttons on single lines according to their group.

Setting Font Properties

The **Fonts** page enables you to customize the font on labeled buttons and the status panel on the Navigation Control.

The **Fonts** page is shown in [Figure 7-4](#).

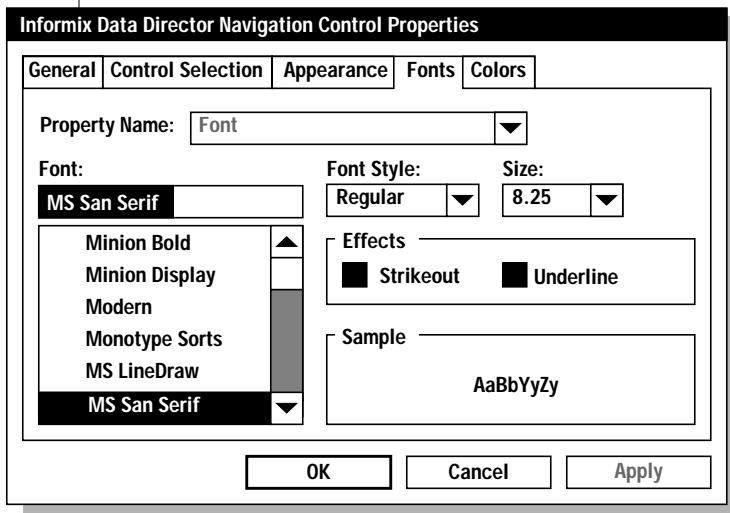


Figure 7-4
Setting Font Properties

Setting Color Properties

The **Colors** page enables you to customize the background color and foreground color of the status panel on the Navigation Control.

The **Colors** page is shown in [Figure 7-5](#).

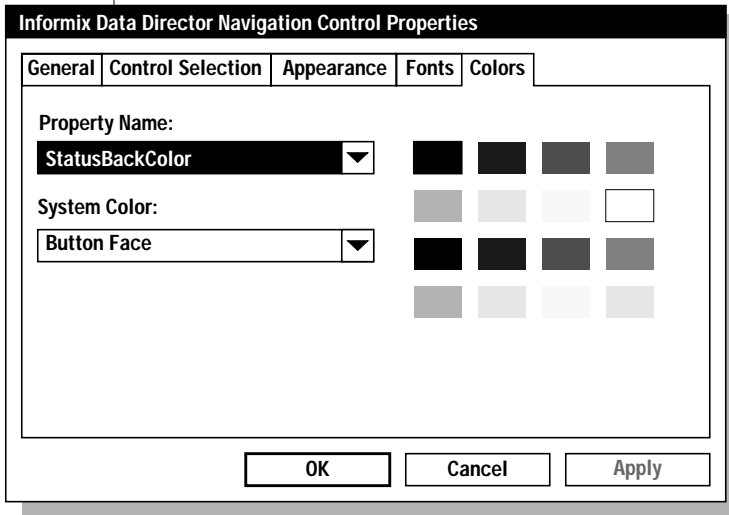


Figure 7-5
Setting Color Properties

Initializing an Application

For some projects, you might prefer to use the Data Director Objects to navigate through data rather than using the Navigation Control. For example, your project might include menu items or coded command buttons for navigating through records instead of a Navigation Control. For more information, see the [Data Director Objects Guide](#).

However, one of the key activities of a Navigation Control is to initialize Data Director when you run a project, as well as to provide access to Navigation Control events. Initialization involves loading a Data Director project (.ddx) file, and then creating an application context within Data Director.

The Navigation Control calls the necessary Data Director functions behind the scenes to automate initialization for you. Although you can manually initialize Data Director, using the Navigation Control is easier and faster. Also, if you use a Navigation Control to initialize an application, you can access the following information as a read-only, runtime Navigation Control property:

- The name of the Navigation Control's selected DataGroup
- The name of the Navigation Control's selected table

You can use a Navigation Control to initialize Data Director and to trap Data Director events regardless of whether the Navigation Control is visible on an application form. You can also use a Navigation Control to access Data Director Objects and Navigation Control events.

To initialize an application

1. Create a Navigation Control.
2. Specify a DataGroup and table for the Navigation Control.
3. If you want the Navigation Control to be invisible at runtime, set the **Visible** property to **False**.

Important: If you use a Navigation Control for initialization, make sure that you create the Navigation Control before calling any Data Director Objects.



Using Multiple Navigation Controls

Because each instance of a Navigation Control can navigate only the specified table associated with a DataGroup, you can include multiple Navigation Controls within an application to provide users with more functionality. For example, a customer service application can use one Navigation Control to navigate through customer records and a second Navigation Control to navigate through order records for each customer.

When an application uses multiple DataGroups, you can use a Navigation Control for each DataGroup to execute different queries and to trap Data Director events for specific DataGroups.

When you want to provide users with restricted data operations for a table, you can create a Navigation Control with only one function, such as a **Query** button to query a specific table. For example, if you want to provide users with the ability to query only records from a specific table, select that table and set the Navigation Control to display only the **Query** button for that table.

Similarly, you can create Navigation Controls that allow only QBEs, or only QBEs and saves, for the table you specify in the Navigation Control's Properties dialog box.

Writing Event Procedures

Data Director sends events to the Navigation Control when certain runtime data events occur, such as when a user modifies data or an error occurs in an application. For example, when a user navigates to the next record in a result set, two Data Director events occur: one immediately before Data Director navigates to the next record (**BeforeGotoRecord**), and another immediately after Data Director navigates to the record (**AfterGotoRecord**).

Although Data Director always sends an event in response to certain data events, you do not have to use the event unless you want to.

Navigation Control Events

Data Director provides events that enable you to trap specific operations that occur in a running application. Most event types correspond to trapping operations before they occur or to operations determining if they have already occurred. You can also trap a custom event and trap errors that occur in a running application. Data Director provides the following kinds of events, which are discussed individually later in this chapter:

- **Before events.** All **Before** events are sent immediately before a specific data event begins. These events enable you to perform actions or execute code before the data operation or to cancel the data operation. For example, **BeforeSave** is sent immediately before Data Director begins to insert or update data in your database.
- **After events.** All **After** events indicate that a specific data event completed successfully and cannot be cancelled. For example, **AfterSave** is sent immediately after Data Director inserts or updates data in your database.
- **Custom events.** Custom events enable you to add a custom button to a Navigation Control and perform a special operation when a user clicks the custom button in a running application.
- **Error events.** Error events are sent any time an error occurs in a Data Director application. You can trap an error event and correct the cause of the error in your application code by using the **Error** object. For more information about error handling in Data Director, see [“Using Error Handling” on page 9-12](#).

All Navigation Control events include parameters that supply additional information to the event recipient. For example, **Table** provides the table name in which a record was created, and **RecNum** provides the number of the current record.

Intercepting Events

To trigger an event procedure in an application, you intercept or trap a specific event and then enter the code that you want to execute on receipt of the event. You can trap any event for the DataGroup specified for a Navigation Control.

To trap an event

1. Choose **View→Code** from the Visual Basic menu bar to open the Code window for a Data Director Navigation Control.
2. Select the event you want to trap from the **Procedures** list box.
3. Add your application code.

Tip: You can programmatically cancel a data operation by using the **Cancel** property (see “[ButtonStyle Property](#)” on page 7-41).



AfterDataChanged

This event occurs after Data Director changes data in a record in its internal buffers.

Syntax

```
Sub NavigationControl_AfterDataChanged( RecNum_as_Long,
    Table_as_String, Column_as_String)
```

Parameters

The parameters for **AfterDataChanged** are listed in the following table.

Parameters	Description
RecNum	The record number of the changed data
Table	The name of the table in which the data was changed
Column	The name of the column in which the data was changed

AfterDelete

This event occurs after Data Director deletes a record.

Syntax

```
Sub NavigationControl_AfterDelete(Table_as_String)
```

Parameters

The parameters for **AfterDelete** are listed in the following table.

Parameters	Description
Table	The name of the table from which Data Director deleted a record

AfterEnterQBE

This event occurs after Data Director enters Query-by-Example mode.

Syntax

```
Sub NavigationControl_AfterEnterQBE(Table_as_String)
```

Parameters

The parameters for **AfterEnterQBE** are listed in the following table.

Parameter	Description
Table	The name of the table on which QBE mode was set

AfterExitQBE

This event occurs after Data Director exits Query-by-Example mode.

Syntax

```
Sub NavigationControl_AfterExitQBE(Table_as_String)
```

Parameters

The parameters for **AfterExitQBE** are listed in the following table.

Parameters	Description
Table	The name of the table on which QBE mode was set

AfterFetchBlob

This event occurs after Data Director fetches a blob value from its internal buffers and displays it in an application window.

Syntax

```
Sub NavigationControl_AfterFetchBlob(RecNum_as_Long,  
Table_as_String, Column_as_String)
```

Parameters

The parameters for **AfterFetchBlob** are listed in the following table.

Parameters	Description
RecNum	The record number of the retrieved blob value
Table	The name of the table on which the blob was retrieved
Column	The name of the column associated with the retrieved blob value

AfterGotoRecord

This event occurs after Data Director navigates to a record in the result set.

Syntax

```
Sub NavigationControl_AfterGotoRecord( RecNum_as_Long,  
Table_as_String)
```

Parameters

The parameters for **AfterGotoRecord** are listed in the following table.

Parameters	Description
RecNum	The number of the record to which Data Director navigated
Table	The name of the table on which Data Director navigated

AfterLogon

This event occurs after Data Director successfully logs on to a database.

Syntax

```
Sub NavigationControl_AfterLogon(UserName_as_String,  
Password_as_String, Server_as_String, Database_as_String,  
DBMS_as_String, DataSource_as_String)
```

Parameters

The parameters for **AfterLogon** are listed in the following table.

Parameters	Description
UserName	The user name used to log on to the database
Password	The password used to log on to the database
Server	The name of the server logged on to
Database	The name of the database logged on to
DBMS	The name of the DBMS logged on to
DataSource	The name of data source associated with the ODBC driver used to connect to the database

AfterNewRecord

This event occurs after Data Director inserts a new record in the result set.

Syntax

```
Sub NavigationControl_AfterNewRecord(RecNum_as_Long,  
                                     Table_as_String)
```

Parameters

The parameters for **AfterNewRecord** are listed in the following table.

Parameters	Description
RecNum	The number of the new record
Table	The name of the table on which the new record was inserted

AfterQuery

This event occurs after Data Director executes a query.

Syntax

```
Sub NavigationControl_AfterQuery(Table_as_String)
```

Parameters

The parameters for **AfterQuery** are listed in the following table.

Parameter	Description
Table	The name of the queried table

AfterSave

This event occurs after Data Director saves data. Saving includes both inserting and updating data.

Syntax

```
Sub NavigationControl_AfterSave(Table_as_String)
```

Parameters

The parameters for **AfterExitQBE** are listed in the following table.

Parameter	Description
Table	The name of the table to which data was saved

BeforeDataChanged

This event occurs before Data Director changes data in a record in its internal buffers. Data Director generates this event in response to a change in data. You can change data by modifying information in an application window, or you can change it programmatically.

Use **BeforeDataChanged** to capture data changes in Data Director’s internal buffers before the data is marked as changed by Data Director. For example, use **BeforeDataChanged** when you want to pop up a message box asking users if they want to save changed data before continuing with another operation.

You can cancel a data-change operation and prevent Data Director from changing data by using the **Cancel** property in response to **BeforeDataChanged**.

After the record is changed, Data Director generates the **AfterDataChanged** event.

Syntax

```
Sub NavigationControl_BeforeDataChanged(RecNum_as_Long,  
Table_as_String, Column_as_String)
```

Parameters

The parameters for **BeforeDataChanged** are listed in the following table.

Parameters	Description
RecNum	The record number of the data being changed
Table	The name of the table on which data was changed
Column	The name of the column on which data was changed

BeforeDelete

This event occurs before Data Director deletes the current record in the result set. Data Director generates **BeforeDelete** in response to a record being deleted. You can delete a record by clicking **Delete** in the Navigation Control, or you can specify that the record be deleted programmatically.

Use **BeforeDelete** to control which records a user can delete. For example, if you do not want to allow users to delete records from specific tables, you can trap this event and cancel the delete operation.

You can cancel a delete operation and prevent Data Director from deleting data by using the **Cancel** property in response to **BeforeDelete**.

After a record is deleted, Data Director generates the **AfterDelete** event.

Syntax

```
Sub NavigationControl_BeforeDelete(Table_as_String)
```

Parameters

The parameters for **BeforeDelete** are listed in the following table.

Parameter	Description
Table	The name of the table containing the record being deleted

BeforeEnterQBE

This event occurs before Data Director enters Query-by-Example (QBE) mode. Data Director generates **BeforeEnterQBE** in response to a QBE request. A QBE request occurs either when you click **QBE** in the Navigation Control or it occurs programmatically.

You can cancel a QBE operation and prevent Data Director from going into QBE mode by using the **Cancel** property in response to **BeforeEnterQBE**.

After Data Director enters QBE mode, Data Director generates the **After-EnterQBE** event.

Syntax

```
Sub NavigationControl_BeforeEnterQBE( Table_as_String)
```

Parameters

The parameters for **BeforeEnterQBE** are listed in the following table.

Parameters	Description
Table	The name of the table entering QBE mode

BeforeExitQBE

This event occurs before Data Director exits Query-by-Example (QBE) mode.

Data Director generates **BeforeExitQBE** in response to a QBE request. A QBE request occurs when you click **QBE** in the Navigation Control, or it occurs programmatically.

You can cancel a QBE operation and prevent Data Director from exiting QBE mode by using the **Cancel** property in response to this event.

After Data Director exits QBE mode, Data Director generates the **After-ExitQBE** event.

Syntax

```
Sub NavigationControl_BeforeExitQBE( Table_as_String)
```

Parameters

The parameters for **BeforeExitQBE** are listed in the following table.

Parameters	Description
Table	The name of the table exiting QBE mode

BeforeFetchBlob

This event occurs before Data Director fetches a blob value. Data Director generates **BeforeFetchBlob** in response to a blob value being fetched. You can specify that a blob value be fetched either programmatically or when your application user queries data in an application window that contains a blob value.

Use **BeforeFetchBlob** when you want to cancel the fetch of a blob value due to time and size constraints when the value has little importance to the data being displayed in an application. For example, if a user is scrolling through employee records, you might not want to display the picture of each employee for each record when the data is fetched.

You can cancel a fetch operation and prevent Data Director from fetching a blob value by using the **Cancel** property in response to **BeforeFetchBlob**.

After the blob value is fetched, Data Director generates the **AfterFetchBlob** event.

Syntax

```
Sub NavigationControl_BeforeFetchBlob( RecNum_as_Long,  
Table_as_String, Column_as_String)
```

Parameters

The parameters for **BeforeFetchBlob** are listed in the following table.

Parameters	Description
RecNum	The record number for the blob value being fetched
Table	The name of the table on which the blob value is being fetched
Column	The name of the column associated with the blob value being fetched

BeforeGotoRecord

This event occurs before Data Director navigates to a record in result set. Data Director generates **BeforeGotoRecord** in response to a request to navigate to a record. You can specify that a record be navigated programmatically or by using the navigation buttons in the Navigation Control.

This event is useful when you want to cancel the navigation operation when a user is about to navigate through a large result set. For example, suppose a query on a table returns a large result set of 10,000 records and a user requests to navigate to the last record in the result set. Since it would take a long time to navigate to the last record in the result set, you might wish to cancel the operation.

You can cancel a record navigation operation and prevent Data Director from navigating to a record by using the **Cancel** property in response to **BeforeGotoRecord**.

After Data Director navigates to a record, it generates the **AfterGotoRecord** event.

Syntax

```
Sub NavigationControl_BeforeGotoRecord( RecNum_as_Long,  
    Table_as_String)
```

Parameters

The parameters for **BeforeGotoRecord** are listed in the following table.

Parameters	Description
RecNum	The record number to which Data Director navigates
Table	The name of the table for which the navigation is taking place

BeforeLogon

This event occurs before Data Director logs on to a database. Data Director generates **BeforeLogon** in response to a logon request. Users can log on to a database when you initialize an application by using the Navigation Control, or you can specify that the logon occurs programmatically.

Use **BeforeLogon** when you want to use your own Logon dialog box in place of the standard Data Director Logon dialog box. For example, when a user queries data, Data Director automatically logs on to the database. You can cancel **BeforeLogon** and pop up your own Logon dialog box in place of the Data Director Logon dialog box.

You can cancel a logon operation and prevent Data Director from logging on to a database by using the **Cancel** property in response to **BeforeLogon**.

After successfully logging on to the database, Data Director generates the **AfterLogon** event.

Syntax

```
Sub NavigationControl_BeforeLogon(UserName_as_String,  
    Password_as_String, Server_as_String, Database_as_String,  
    DBMS_as_String, DataSource_as_String)
```

Parameters

The parameters for **BeforeLogon** are listed in the following table.

Parameters	Description
UserName	The user name being used to log on to the database
Password	The password being used to log on to the database
Server	The name of the server being logged on to
Database	The name of the database being logged on to
DBMS	The name of the DBMS being logged on to
DataSource	The name of data source associated with the ODBC driver used to connect to the database

BeforeNewRecord

This event occurs before Data Director creates a new record. Data Director generates **BeforeNewRecord** in response to a request to create a new record. You can create a record by clicking **New Record** in the Navigation Control, or you can specify that the log on be done programmatically.

You can cancel a new record operation and prevent Data Director from creating a record by using the **Cancel** property in response to **BeforeNewRecord**.

After the record is created, Data Director generates the **AfterNewRecord** event.

Syntax

```
Sub NavigationControl_BeforeNewRecord(RecNum_as_Long,  
Table_as_String)
```

Parameters

The parameters for **BeforeNewRecord** are listed in the following table.

Parameters	Description
RecNum	The number of the record being created
Table	The name of the table that contains the record being created

BeforeQuery

This event occurs before Data Director executes a query. Data Director generates **BeforeQuery** in response to a query request. A query request occurs either when you click **Query** in the Navigation Control, or you can specify that the query request occurs programmatically.

You can cancel a query operation by using the **Cancel** property in response to **BeforeQuery**.

After Data Director executes a query, Data Director generates the **AfterQuery** event.

Syntax

```
Sub NavigationControl_BeforeQuery(Table_as_String)
```

Parameters

The parameters for **BeforeQuery** are listed in the following table.

Parameters	Description
Table	The name of the table being queried

BeforeSave

This event occurs before Data Director saves data. You can save data by clicking **Save** in the Navigation Control, or you can specify that the data be saved programmatically.

Use **BeforeSave** when you want to capture data changes and validate the data before the data is saved to the database. You can also control which records a user can save. For example, if you do not want to allow users to save data to specific tables, you can trap this event and cancel the save operation.

You can cancel a save operation and prevent Data Director from saving data by using the **Cancel** property in response to **BeforeSave**.

After successfully saving data, Data Director generates the **AfterSave** event.

Syntax

```
Sub NavigationControl_BeforeSave(Table_as_String)
```

Parameters

The parameters for **BeforeSave** are listed in the following table.

Parameters	Description
Table	The table to which Data Director is saving data

CustomEvent

This event occurs when a user clicks a custom button on a Navigation Control at runtime. This custom button is associated with a specific table.

Syntax

```
Sub NavigationControl_CustomEvent(Table_as_String)
```

Parameters

The parameters for **CustomEvent** are listed in the following table.

Parameters	Description
Table	The table with which the custom event is associated

Example

This event is useful when you want to add a custom button to a Navigation Control and perform a special operation when a user clicks the custom button. For example, if you want to launch a calculator program when a user clicks a **Calculator** button on a Navigation Control, your code for the **CustomEvent** would look as follows:

```
Private Sub NavControl1_CustomEvent(Table_as_String)  
    Shell "CALC.EXE", 1 ' Run Calculator.  
End Sub
```

Error

This event occurs after Data Director generates an error.

Syntax

```
Sub NavigationControl_Error(Text_as_String, Number_as_Integer,  
    Time_as_String, Severity_as_Integer, Category_as_Integer,  
    HelpFile_as_String, HelpContext_as_Integer)
```

Parameters

The parameters for **Error** are listed in the following table.

Parameters	Description
Text	Returns the error text string
Number	Returns the Data Director error number or the database error number
Time	Returns the time the object was created
Severity	Returns the error severity level: FATAL INFORMATIONAL WARNING
Category	Returns the type of error generated: DATA DIRECTOR DATABASE SYSTEM
HelpFile	Returns a fully qualified path to the help file
HelpContext	Returns a context ID for a topic in a Microsoft Windows help file

OptimisticDetectChanged

This event occurs when a concurrent change by another user is detected during a save operation, if the application is running in optimistic detect concurrency mode.

Syntax

```
Sub NavigationControl_OptimisticDetectChanged(RecNum_as_Long,  
Table_as_String)
```

Parameters

The parameters for **BeforeFetchBlob** are listed in the following table.

Parameters	Description
RecNum	The number of the record in which the changed was detected
Table	The name of the table that contains the changed record

Navigation Control Events Quick Reference

The following table provides you with a quick reference to each event, including a brief description and information about the parameters available to each event.

Event	Occurs	Parameters	Data Type
AfterDataChanged	After Data Director changes data in its internal buffers	RecNum Table Column	Long String String
AfterDelete	After Data Director deletes data	Table	String
AfterEnterQBE	After Data Director enters QBE mode	Table	String
AfterExitQBE	After Data Director exits QBE mode	Table	String
AfterFetchBlob	After Data Director fetches a blob value	RecNum Table Column	Long String String
AfterGotoRecord	After Data Director goes to a record	RecNum Table	Long String
AfterLogon	After Data Director logs on to a database	UserName Password Server Database DBMS DataSource	String String String String String String
AfterNewRecord	After Data Director creates a new record	RecNum Table	Long String
AfterQuery	After Data Director executes a query	Table	String
AfterSave	After Data Director saves data	Table	String
BeforeDataChanged	Before Data Director changes data in its internal buffers	RecNum Table Column	Long String String
BeforeDelete	Before Data Director deletes a record	Table	String

(1 of 2)

Event	Occurs	Parameters	Data Type
BeforeEnterQBE	Before Data Director enters QBE mode	Table	String
BeforeExitQBE	Before Data Director exits QBE mode	Table	String
BeforeFetchBlob	Before Data Director fetches a blob value	RecNum Table Column	Long String String
BeforeGotoRecord	Before Data Director goes to a record	RecNum Table	Long String
BeforeLogon	Before Data Director logs on to a database	UserName Password Server Database DBMS DataSource	String String String String String String
BeforeNewRecord	Before Data Director creates a new record	RecNum Table	Long String
BeforeQuery	Before Data Director executes a query	Table	String
BeforeSave	Before Data Director saves data	Table	String
CustomEvent	When a user clicks a custom button	Table	String
Error	When an error occurs in Data Director	Text Number Time Severity Category HelpFile HelpContext	String Integer String Integer Integer String Integer
OptimisticDetectChanged	When a concurrent change is detected	RecNum Table	Long String

(2 of 2)

Navigation Control Properties

This section describes each custom Navigation Control property. The syntax shown for each property in this section is for setting a property value.

To set a current property value, specify a value for the property variable. The following example sets the current value of the bookmark button **Visible** flag (for a Navigation Control named **NavigationControl**) to the Boolean value **True**:

```
NavigationControl.BookmarkButtonsVisible = True
```

To return a current property value, specify a variable for the return value and set it equal to the property name. The following example returns the current value of the **Cancel** flag (for a Navigation Control named **NavigationControl**) to the variable **iCancel**:

```
iCancel=NavigationControl.Cancel
```

For information about the following standard Visual Basic objects and properties, refer to your Visual Basic documentation.

DragIcon	Index	Tag
DragMode	Left	Top
Font	Name	Visible
Height	TabIndex	WhatsThisHelpID
HelpContextID	TabStop	Width

BookmarkButtonsVisible Property

Sets and returns the visible status of the bookmark button set.

Syntax

```
[ form. ]NavigationControl.BookmarkButtonsVisible  
[ = Boolean% ]
```

Data Type

Integer (Boolean)

Usage

Read/write at design time and runtime.

Settings

The settings for **BookmarkButtonsVisible** are as follows.

Setting	Description
True	Display the bookmark button set
False	Do not display the bookmark button set

ButtonStyle Property

Sets and returns the style of the buttons on the Navigation Control.

Syntax

```
[form.]NavigationControl.ButtonStyle [= Integer%]
```

Data Type

Integer

Usage

Read/write at design time and runtime.

Constants

The constants for **ButtonStyle** are as follows.

Constant	Description
cncRegular	Display the regular button style for buttons on the Navigation Control
cncFlat	Display a flat button style for buttons on the Navigation Control
cncRound	Display a round button style for buttons on the Navigation Control

Cancel Property

Sets and returns the value of the cancel flag. This flag determines whether the current **Before** event for the Navigation Control will generate a corresponding **After** event. Use this property to cancel operations that trigger Navigation Control events.

Syntax

```
[form.]NavigationControl.Cancel [= Boolean%]
```

Data Type

Boolean

Usage

Read/write at runtime.

Settings

The settings for **Cancel** are as follows.

Setting	Description
True	Cancel the event; do not generate an After event for the corresponding Before event
False	Do not cancel the event; proceed with the operation; generate an After event for the corresponding Before event

Example

This is an example of using the **Cancel** property for a Navigation Control named **Nav1**.

```
'Before you save in Data Director, check to see what the table
'name is. If the table name is ORDERS, cancel the save.
Sub Nav1_BeforeSave (Table_as_String)
    If (Table = "ORDERS") Then
        Nav1.Cancel = True
    End If
End Sub
```

DataGroup Property

Sets and returns the name of the DataGroup associated with the Navigation Control.

Syntax

```
[form.]NavigationControl.DataGroup [=String$]
```

Data Type

String

Usage

Read/write at design time. Read-only at runtime.

DefaultButtonSize Property

Sets and returns the default button size.

Syntax

```
[form.]NavigationControl.DefaultButtonSize [= Integer%]
```

Data Type

Integer

Usage

Read/write at design time and runtime

Constants

The constants for **DefaultButtonSize** are as follows.

Constant	Description
cncSmall	Display small-sized buttons on the Navigation Control
cncMedium	Display medium-sized buttons on the Navigation Control
cncLarge	Display large-sized buttons on the Navigation Control

FrameStyle Property

Sets and returns the style of the frame surrounding the Navigation Control.

Syntax

```
[form.]NavigationControl.FrameStyle [= Integer%]
```

Data Type

Integer

Usage

Read/write at design time and runtime.

Constants

The constants for **FrameStyle** are as follows.

Constant	Description
cncNone	Do not display a frame style surrounding the Navigation Control
cncEmbossed	Display an embossed frame style surrounding the Navigation Control
cncRecessed	Display a recessed frame style surrounding the Navigation Control

LayoutMode Property

Sets and returns how the Navigation Control responds to changes in dimensions.

Syntax

```
[ form. ]NavigationControl.LayoutMode [= Integer%]
```

Data Type

Integer

Usage

Read/write at design time and runtime.

Constants

The constants for **LayoutMode** are as follows.

Constant	Description
cncAutomatic	Automatically lay out the Navigation Control to optimize the space used by the control buttons
cncStretchStatus	Stretch the status panel while maintaining the button size when resizing the Navigation Control

ModelFile Property

Sets and returns the filename and pathname of the Model file associated with the Navigation Control. The Model file must contain a table of the same name as set in the **Table** property. This property is set by Data Director when the Navigation Control is bound.

Syntax

```
[form.]NavigationControl.ModelFile [= String$]
```

Data Type

String

Usage

Read/write at design time. Read-only at runtime.

NavigationButtonsVisible Property

Sets and returns the visible status of the navigation button set on the Navigation Control.

Syntax

```
[form.]NavigationControl.NavigationButtonsVisible  
[= Boolean%]
```

Data Type

Integer (Boolean)

Usage

Read/write at design time and runtime.

Settings

The settings for **NavigationButtonsVisible** are as follows.

Setting	Description
True	Display the navigation button set
False	Do not display the navigation button set

OptimisticDetect Property

Sets and returns a value that determines how to proceed when you trap the **OptimisticDetectChanged** event.

Syntax

```
[form.]NavigationControl.OptimisticDetect [= Integer%]
```

Data Type

Integer

Usage

Read/write at design time and runtime. Use this property with the **OptimisticDetectChanged** event.

Settings

The settings for **OptimisticDetect** are as follows.

Setting	Description
CancelUpdate	Cancel the update (default)
Proceed	Proceed with the update
CancelMarkSaved	Do not proceed with the update, but mark the cached record as saved

PreferSingleGroupPerLine Property

Sets and returns a value indicating how the Navigation Control lays out groups of controls.

The **PreferSingleGroupPerLine** property lays out groups of controls in separate rows of the Navigation Control; each group occupies a single row. Controls that can be grouped are navigation buttons, bookmark buttons, the status panel, and action buttons.

Syntax

```
[form.]NavigationControl.PreferSingleGroupPerLine  
[= Boolean%]
```

Data Type

Integer (Boolean)

Usage

Read/write at design time and runtime.

Settings

The settings for **PreferSingleGroupPerLine** are as follows.

Setting	Description
True	Group like controls on separate lines of the Navigation Control
False	Do not group like controls on separate lines of the Navigation Control

SpacingHorizontal Property

Sets and returns the horizontal spacing between the buttons on the Navigation Control.

Syntax

```
[form.]NavigationControl.SpacingHorizontal [= Integer%]
```

Data Type

Integer

Usage

Read/write at design time and runtime.

Settings

You can specify a value between 0 and 10 for the **SpacingHorizontal** property.

SpacingVertical Property

Sets and returns the vertical spacing between the buttons on the Navigation Control.

Syntax

```
[form.]NavigationControl.SpacingVertical [= Integer%]
```

Data Type

Integer

Usage

Read/write at design time and runtime.

Settings

You can specify a value between 0 and 10 for the **SpacingVertical** property.

StatusBackColor Property

Sets and returns the color used to paint the background of the status panel.

Syntax

```
[form.]NavigationControl.StatusBackColor [= Long&]
```

Data Type

Long (color value)

Usage

Read/write at design time and runtime.

Settings

- You can specify normal RGB colors or use system default colors for the **StatusBackColor** setting. Normal RGB colors are colors that are specified using the Color palette, or by using the **RGB** or **QBColor** functions in code.
- You can specify colors using a decimal value (0) or a hexadecimal value (&H00FFFFFF). The valid range for a normal 16-bit RGB color is 0 to 16,777,215 (&H0000000 to &HFFFFFFF).
- The system default colors are those that are defined by the user's Control Panel settings or by constants listed in the Visual Basic object library in the Object Browser.

StatusCaption Property

Sets and returns the format string used to generate the text on the status panel.

The format string will replace the first occurrence of *T* with the current table name and the first occurrence of *#* with the current record number. Any other text might be mixed with either parameter, both parameters, or neither parameter.

Syntax

```
[ form. ]NavigationControl.StatusCaption [=String$]
```

Data Type

String

Usage

Read/write at design time and runtime.

Example

T denotes the current table name.

denotes the current record number.

You can customize the caption by adding text before and after the table and record symbols. You can also omit these symbols in your customized caption if you do not want to include the table name or record number on the status panel.

For example, if the current table name is **CUSTOMERS** and the current record number is 15, you can achieve a variety of results based on what you specify as the syntax for this property. The following tables shows a few examples.

Status Caption	Displays
T - #	CUSTOMERS - 15
Table: T	Table: customers
Record: #	Record: 15
Updating Database	Updating Database

StatusForeColor Property

Sets and returns the color used to display the text on the status panel.

Syntax

```
[form.]NavigationControl.StatusForeColor [= Long&]
```

Data Type

Long (color value)

Usage

Read/write at design time and runtime.

Settings

- You can specify normal RGB colors or use system default colors for the **StatusBackColor** setting. Normal RGB colors are colors that are specified using the Color palette or by using the **RGB** or **QBColor** functions in code.
- You can specify colors using a decimal value (0) or a hexadecimal value (&H00FFFFFF). The valid range for a normal 16-bit RGB color is 0 to 16,777,215 (&H00000000 to &HFFFFFFF).
- The system default colors are those that are defined by the user's Control Panel settings or by constants listed in the Visual Basic object library in the Object Browser.

StatusVisible Property

Sets and returns the visible status of the status panel.

Syntax

```
[form.]NavigationControl.StatusVisible [= Boolean%]
```

Data Type

Integer (Boolean)

Usage

Read/write at design time and runtime

Settings

The settings for **StatusVisible** are as follows.

Setting	Description
True	Display the status panel
False	Do not display the status panel

Table Property

Sets and returns the name of the table associated with the Navigation Control.

Syntax

```
[form.]NavigationControl.Table [= String$]
```

Data Type

String

Usage

Read/write at design time. Read-only at runtime.

Navigation Control Properties Quick Reference

The following table provides you with a quick reference to each Navigation Control property, including the design-time and runtime attributes of each property.

Navigation Control Property Categories	Navigation Control Properties	Design Time		Runtime	
		DP*	D*	RW*	RO*
Control appearance	ButtonStyle	■	■	■	■
	DefaultButtonSize	■	■	■	
	FrameStyle	■	■	■	
	PreferSingleGroup- PerLine	■	■	■	
	SpacingHorizontal	■	■	■	
	SpacingVertical	■	■	■	
Control selection	BookmarkButtons- Visible	■	■	■	■
	NavigationButtons- Visible	■	■	■	
Data operations	Cancel	■	■	■	■
	OptimisticDetect			■	■

*** KEY:**

DP Set at design time using the Data Director Navigation Control Properties dialog box.

D Set at design time using the Visual Basic Properties dialog box.

RW Read/write at runtime through Visual Basic code using object properties.

RO Read-only at runtime.

Navigation Control Property Categories	Navigation Control Properties	Design Time		Runtime	
		DP*	D*	RW*	RO*
DataGroup and table	DataGroup	■	■	■	■
	ModelFile	■	■		■
	Table	■	■		■
Layout mode	LayoutMode	■	■	■	■
Status panel	StatusBackColor	■	■	■	■
	StatusCaption	■	■	■	
	StatusForeColor	■	■	■	
	StatusVisible	■	■	■	

*** KEY:**

DP Set at design time using the Data Director Navigation Control Properties dialog box.

D Set at design time using the Visual Basic Properties dialog box.

RW Read/write at runtime through Visual Basic code using object properties.

RO Read-only at runtime.

Navigation Control Objects

This section describes each Navigation Control object. You can set properties for the objects using the Data Director Navigation Control Properties dialog box, or you can set properties programmatically.

When using the objects, be sure to include the parent object (the Navigation Control) in your syntax.

To set a current property value, specify a value for the property variable. The following example sets the current value of the **New** button **Visible** flag (for a Navigation Control named **NavigationControl**) to the Boolean value **True**:

```
NavigationControl.NewButton.Visible = True
```

To return a current property value, specify a variable for the return value and set it equal to the property name. The following example returns the current value of the **Visible** flag for a **Delete** button (for a Navigation Control named **NavigationControl**) to the variable **iVisible**:

```
iVisible=NavigationControl.DeleteButton.Visible
```

Each of the Navigation Control objects uses the following standard Visual Basic properties; for more information about these properties, refer to your Visual Basic documentation:

- **Caption**
- **Enabled**
- **Visible**

CustomButton1 Object

Provides object-oriented access to the custom button and enables you to set properties for the button. The custom button enables you to programmatically trap an event and perform a custom data operation.

These properties are not available in the Visual Basic Properties dialog box. Set properties for the **CustomButton1** object programmatically or through the Navigation Control Properties dialog box.

Syntax

```
[form.]NavigationControl.CustomButton1.Visible [= Boolean%]  
[form.]NavigationControl.CustomButton1.Enabled [= Boolean%]  
[form.]NavigationControl.CustomButton1.Caption [= String$]
```

Data Types

Integer (Boolean), String

Usage

Read/write at design time and runtime.

Settings

The settings for **Boolean** are as follows.

Setting	Description
True	Display/enable the custom button
False	Hide/disable the custom button

DeleteButton Object

Provides object-oriented access to the **Delete** button and enables you to set properties for the button.

These properties are not available in the Visual Basic Properties dialog box. Set properties for the **DeleteButton** object programmatically or through the Data Director Navigation Control Properties dialog box.

Syntax

```
[form.]NavigationControl.DeleteButton.Visible [= Boolean%]  
[form.]NavigationControl.DeleteButton.Enabled [= Boolean%]  
[form.]NavigationControl.DeleteButton.Caption [= String$]  
[form.]NavigationControl.DeleteButton.StyleFlags [= Integer%]
```

Data Type

Integer (Boolean), String

Usage

Read/write at design time and runtime.

Settings

The settings for **Boolean** are as follows.

Setting	Description
True	Display/enable the Delete button
False	Hide/disable the Delete button

Constants

The constants for **StyleFlags** are as follows.

Constant	Value	Description
cncIcon	1	Display a bitmap or icon on the Delete button
cncText	2	Display a caption on the Delete button

NewButton Object

Provides object-oriented access to the **New** button and enables you to set properties for the button.

These properties are not available in the Visual Basic Properties dialog box. Set properties for the **NewButton** object programmatically or through the Data Director Navigation Control Properties dialog box.

Syntax

```
[form.]NavigationControl.NewButton.Visible [= Boolean%]  
[form.]NavigationControl.NewButton.Enabled [= Boolean%]  
[form.]NavigationControl.NewButton.Caption [= String$]  
[form.]NavigationControl.NewButton.StyleFlags [= Integer%]
```

Data Type

Integer (Boolean), String

Usage

Read/write at design time and runtime.

Settings

The settings for **Boolean** are as follows.

Setting	Description
True	Display/enable the New button
False	Hide/disable the New button

Constants

The constants for **StyleFlags** are as follows.

Constant	Value	Description
cncIcon	1	Display a bitmap or icon on the New button
cncText	2	Display a caption on the New button

QBEButton Object

Provides object-oriented access to the **QBE** button and enables you to set properties for the button.

These properties are not available in the Visual Basic Properties dialog box. Set properties for the **QBEButton** object programmatically or through the Data Director Navigation Control Properties dialog box.

Syntax

```
[form.]NavigationControl.QBEButton.Visible [= Boolean%]  
[form.]NavigationControl.QBEButton.Enabled [= Boolean%]  
[form.]NavigationControl.QBEButton.Caption [= String$]  
[form.]NavigationControl.QBEButton.StyleFlags [= Integer%]
```

Data Type

Integer (Boolean), String

Usage

Read/write at design time and runtime.

Settings

The settings for **Boolean** are as follows.

Setting	Description
True	Display/enable the QBE button
False	Hide/disable the QBE button

Constants

The constants for **StyleFlags** are as follows.

Constant	Value	Description
cncIcon	1	Display a bitmap or icon on the QBE button
cncText	2	Display a caption on the QBE button

QueryButton Object

Provides object-oriented access to the **Query** button and enables you to set properties for the button.

These properties are not available in the Visual Basic Properties dialog box. Set properties for the **QueryButton** object programmatically or through the Data Director Navigation Control Properties dialog box.

Syntax

```
[ form. ]NavigationControl.QueryButton.Visible [= Boolean%]  
[ form. ]NavigationControl.QueryButton.Enabled [= Boolean%]  
[ form. ]NavigationControl.QueryButton.Caption [= String$]  
[ form. ]NavigationControl.QueryButton.StyleFlags [= Integer%]
```

Data Type

Integer (Boolean), String

Usage

Read/write at design time and runtime.

Settings

The settings for **Boolean** are as follows.

Setting	Description
True	Display/enable the Query button
False	Hide/disable the Query button

Constants

The constants for **StyleFlags** are as follows.

Constant	Value	Description
cncIcon	1	Display a bitmap or icon on the Query button
cncText	2	Display a caption on the Query button

SaveButton Object

Provides object-oriented access to the **Save** button and enables you to set properties for the button.

These properties are not available in the Visual Basic Properties dialog box. Set properties for the **SaveButton** object programmatically or through the Data Director Navigation Control Properties dialog box.

Syntax

```
[form.]NavigationControl.SaveButton.Visible [= Boolean%]  
[form.]NavigationControl.SaveButton.Enabled [= Boolean%]  
[form.]NavigationControl.SaveButton.Caption [= String$]  
[form.]NavigationControl.SaveButton.StyleFlags [= Integer%]
```

Data Type

Integer (Boolean), String

Usage

Read/write at design time and runtime.

Settings

The settings for **Boolean** are as follows.

Setting	Description
True	Display/enable the Save button
False	Hide/disable the Save button

Constants

The constants for **StyleFlags** are as follows.

Constant	Value	Description
cncIcon	1	Display a bitmap or icon on the Save button
cncText	2	Display a caption on the Save button

Navigation Control Objects Quick Reference

The following table provides you with a quick reference to each Navigation Control object, including the design-time and runtime attributes of each object.

Navigation Control Objects	Design Time		Runtime	
	DP*	D*	RW*	RO*
CustomButton1	■		■	
DeleteButton	■		■	
NewButton	■		■	
QBEBButton	■		■	
QueryButton	■		■	
SaveButton	■		■	

*** KEY:**

DP Set at design time using the Data Director Navigation Control Properties dialog box.

D Set at design time using the Visual Basic Properties dialog box.

RW Read/write at runtime through Visual Basic code using object properties.

RO Read-only at runtime.

The Navigation Control Objects have the properties **Visible**, **Enabled**, **Caption**, and **StyleFlags**. The only exception to this is for the **CustomButton1** object, which does not use the **StyleFlags** property.

Multiple-Developer and Multiuser Support

Building Multiple-Developer Projects	8-3
Using a Source Code Control System	8-4
Data Director Works with Any Source Code Control System	8-4
About Visual Basic and Data Director Project Files	8-5
Checking Out and Checking In Files	8-6
Checking Out Files	8-6
Checking In Files	8-7
Working with Project Files	8-8
Working with Form Files	8-9
Working with Model Files	8-10
Working with DataGroup Files	8-11
Working with Data Director Project Files	8-11
Applying Changes to Your Project	8-12
Compiling Your Project	8-12
Building Multiuser Applications	8-13
Multiple-Developer File Status Indicators	8-14

This chapter discusses multiple-developer and multiuser support in Data Director.

Building Multiple-Developer Projects

Data Director enables you to build multiple-developer projects, allowing you to share project files across development teams for the simultaneous development of Data Director applications.

Using a source code control system and the multiple-developer feature in Data Director, you can:

- share project files with other developers to concurrently develop a Data Director application.
- track different versions of project files and maintain a history of file versions.
- merge changes in project files with the project source code.

Each Visual Basic project contains multiple files that can be easily managed using a source code control system. Similarly, each Visual Basic project developed using Data Director contains Data Director–specific files that hold different types of information specific to the Data Director application. These Data Director files can also be easily managed by your source code control system.

Using a Source Code Control System

Using a source code control system, you can check files out and make changes to the files in the Visual Basic and Data Director development environment. When you check in a file, the source code control system ensures that the changes are integrated into your project and tracks revisions to the project code.

For example, your Visual Basic project environment might be housed in a central repository on a server. Each developer might have a personal version of the project under development on a client PC. The central repository contains the final version of the code, while the client PCs contain each developer's personal build environment.

Using the source code control system, developers can check out files from the central repository and work in their own build environment. When developers are finished making changes to the files, they check in the files to the central repository development environment, which also tracks any revisions made to the project code.

Data Director Works with Any Source Code Control System

Data Director integrates with any commercially available source code control system, including the Microsoft Visual Source Safe environment. As long as you have a source code control system or Source Safe installed and are using it in your development environment, you can control the access and use of Data Director project files.

Some source code control systems allow you to check out only one file at a time. Most source code control systems, however, allow multiple developers to simultaneously check out the same file, detect changes in files when they are checked in, and alert you to any conflicts that might occur when merging files that are checked in by multiple developers at the same time.

Depending on whether the Data Director files that you are working with are text files or binary files, you can check out multiple files at the same time if your source code control system supports this functionality.

About Visual Basic and Data Director Project Files

Each Visual Basic project that you create contains the following files:

- A project (**.vbp**) file
- One or more form (**.frm**) files
- One or more module (**.bas**) files
- One or more class (**.cls**) files

Because Visual Basic stores information about a project in separate files, multiple developers can work on the same project at the same time, as long as they are not working on the same form or code module at the same time.

Data Director provides additional files that integrate with the Visual Basic environment; each file contains information specific to a Data Director application. The following additional Data Director files are included:

- One or more Model (**.mlt** and **.mlx**) files
- One or more DataGroup (**.dgp**) files
- A compiled Data Director (**.ddx**) project file

Similar to Visual Basic files, developers can work on the same Data Director project files at the same time. Data Director also supports the ability to work on the same Model and DataGroup files at the same time. If you are working with text files, such as Model (**.mlt**) and DataGroup (**.dgp**) files, you can merge changes to the files when you check them in. Binary files, such as Model (**.mlx**) or Data Director project (**.ddx**) files, can be checked out simultaneously, but because they are binary files, you cannot merge multiple changes made to these files with your source code.

Data Director also writes information specific to a Data Director application to the Visual Basic project and form files. The sections that follow discuss each of the Data Director project files, including the Visual Basic project and form files.

Checking Out and Checking In Files

Data Director integrates with any commercially available source code control system. However, if you use the Microsoft Visual Source Safe development environment, Data Director provides additional functionality to your source code control system. For example, if you use Source Safe, Data Director automatically prompts you to check out Data Director files when you make changes to your project.

Source Safe is available with the Enterprise edition of Visual Basic and is an add-in to Visual Basic. As long as you have Source Safe installed and have added your Visual Basic project into the Source Safe environment, you can check in and check out Data Director files using the Visual Basic **Tools** menu or by running Source Safe.

Source code control systems operate under the concepts of checking out files and checking in files. Checking out files from the source code control system marks the file as being checked out by your user name and makes the file writable so that you can make changes to it. Checking in a file integrates your work into the source code in the project environment.

If your source code control system supports the ability to check out and check in the same file by multiple developers, you can merge changes from multiple developers into your source code.

Checking Out Files

Checking out a file enables you to work on the file locally and then merge your changes into the project source code. Depending on the type of source code control system you use, multiple developers can check out the same file at the same time, or only one developer can check out a file at a time.

To check out a file using Source Safe

1. Start Microsoft Visual Source Safe.
2. Select the project that contains the file you want to check out.
3. Select the file that you want to check out.
4. Choose **SourceSafe→CheckOut** from the Source Safe menu bar.

For more information about using Source Safe to check out files, refer to your Microsoft Visual Source Safe documentation.

Automatically Checking Out Files

If you use the Source Safe development environment, Data Director automatically prompts you to check out the appropriate Data Director files when you attempt to change a project's attributes using the user interface. For example, if you attempt to make a change to a DataGroup property, Data Director prompts you to check out the DataGroup file associated with the DataGroup that you are changing.

Checking In Files

After you make changes to a file, you check it back in to the source code control system. When you save a project or form and check in the file that you were working on, your changes are integrated into the project environment.

If your source code control system supports the ability to check in the same file by multiple developers, you can merge changes from multiple developers into the source file when they are checked in.

For example, Data Director DataGroup (.dgp) files are saved in a text format, enabling multiple developers to simultaneously check in DataGroup files. Some source code control systems are able to read the text format and detect merge conflicts. These systems will notify you of any conflicts in the merged file and alert you to these differences. You can then examine the differences and correct them using a text editor.

To check in a file using Source Safe

1. Start Microsoft Visual Source Safe.
2. Select the project that contains the file that you want to check in.
3. Select the file that you want to check in.
4. Choose **SourceSafe→CheckIn** from the Source Safe menu bar.

For more information about using Source Safe to check out files, refer to your Microsoft Visual Source Safe documentation.

Working with Project Files

Visual Basic project (**.vbproj**) files contain information specific to a Visual Basic project, such as the name of the project, the forms contained in the project, and the name of any custom controls added to the Visual Basic Toolbox for the project.

In addition to the information that Visual Basic stores in a project file, Data Director writes information that pertains to a Data Director project to the project file. The following additional information is written to the project file:

- Project-level information such as the file version
- Mapping from the DataGroups used in the project to the specific DataGroup filenames

Data Director modifies a project file under the following conditions:

- When the Data Director software has been updated
- When a DataGroup is added to a project
- When a DataGroup is removed from a project
- When a DataGroup filename has changed

You need to check out a Visual Basic project file using your source code control system when you want to perform the following Data Director operations:

- Create a DataGroup
- Delete a DataGroup
- Rename a DataGroup
- Change project trace file settings
- Save a project created using a previous version of Data Director

If you attempt to perform any of these operations when the project file is not checked out, Data Director prompts you to check out the project file.

Refer to your Visual Basic documentation for any additional considerations regarding using Visual Basic project files with a source code control system.

Working with Form Files

Visual Basic form (.**frm**) files contain information specific to a Visual Basic form, such as the name of the form and the attributes of the form.

In addition to the information that Visual Basic stores in a form file, Data Director writes Data Director–specific information to each form file in a project. The following additional information is written to the form file:

- The table and column of each DataLink that is linked to controls on the form
- The data path of each DataLink
- DataLink properties

Data Director modifies a form file when you save the Visual Basic project.

You need to check out a Visual Basic form file by using your source code control system when you want to perform the following Data Director operations:

- Create new DataLinks to controls on the form
- Update existing DataLinks to controls on the form
- Set DataLink properties
- Delete DataLinks
- Delete a DataGroup that has DataLinks to the form
- Delete a control from a form
- Replace an existing DataLink

If you attempt to perform any of these operations when the form file is not checked out, Data Director prompts you to check out the form file.

Refer to your Visual Basic documentation for any additional considerations regarding using Visual Basic form files with a source code control system.

Working with Model Files

Data Director Model files contain information about a Model; each Model in a project has its own file. Model files contain the following information:

- Information about the tables, views, and routines in the Model
- Information about the columns in each table, including column names and data types
- Information about the relationships between tables in the Model

When you import a database structure to a Model and save the Model, Data Director saves the Model information in two Model files: an **.mlx** binary file that you distribute with an application and an **.mlt** text file that you can check out using your source code control system to make changes to the Model file.

Because Data Director stores Model information in a text file, each Model file can be checked out and maintained by multiple developers using your source code control system.

Data Director modifies a Model file when you save the Visual Basic project.

You need to check out a Model form file using your source code control system when you want to make any changes to your Model.



Important: If you move a compiled Model (**.mlx**) file, when you run your project, you might receive an error message stating that Data Director cannot locate your Model file. To find a Model file in the Visual Basic environment, Data Director looks first in the hard-coded path of the **ModelFile** property of the Navigation Control; second, in the directory containing the Data Director project (**.ddx**) file; and finally, in the current directory. If you receive this error message, either update the path of the Model in the Navigation Control Properties dialog box, or move the **.mlx** file into the same directory as the project's **.ddx** file.

Working with DataGroup Files

Data Director DataGroup (.dgp) files are text files that contain information about a single DataGroup; each DataGroup in a project has its own file. DataGroup files contain the following information:

- The name of the DataGroup
- The name of the master table in the DataGroup
- The name of the Model on which the DataGroup is based
- DataGroup properties
- A list of forms that contain DataLinks to tables and columns in the DataGroup

Because Data Director stores DataGroup information in a text file, each DataGroup file can be checked out and maintained by multiple developers using your source code control system.

Data Director modifies a DataGroup file when you save the Visual Basic project.

You need to check out a Data Director DataGroup file using your source code control system when you want to perform the following Data Director tasks:

- Change the name of the Model file on which the DataGroup is based
- Set DataGroup properties
- Add or delete forms that contain DataLinks to the DataGroup

Working with Data Director Project Files

Data Director project (.ddx) files are compiled files that contain information required to run your project. When you save your project, Data Director reads the Visual Basic project file, the Visual Basic form files, and the Data Director DataGroup files to create the Data Director project file.

Generating a Data Director project file is discussed in more detail in [“Compiling Your Project” on page 8-11](#).

Data Director project files are also required for application deployment (see [“Deploying Your Application” on page 10-3](#)).

Applying Changes to Your Project

To reflect changes in files, you need to save the Visual Basic project file and form files. When you save your project files and form files, Data Director integrates your changes into the appropriate Visual Basic and Data Director files.

Compiling Your Project

In order to distribute your application, you need to compile your project to generate the compiled Data Director project (**.ddx**) file that is distributed with your application.

To generate a Data Director project file

1. Open your project in Visual Basic.
2. Check out the Visual Basic project files that you will be modifying.
For detailed instructions, see [“Checking Out Files” on page 8-6](#).
3. Check out the Data Director (**.ddx**) project file.
For detailed instructions, see [“Checking Out Files” on page 8-6](#).
4. Ensure that the Data Director project file is writable.
5. Save your project.
Data Director saves the **.ddx** file to disk; this file contains a compiled version of all of the Data Director information included in the project.
6. Check in all project files, including the **.ddx** file.

Building Multiuser Applications

One of the primary concerns for a developer building client/server applications is managing the concurrent access to data by multiple application users. Concurrent access to data refers to more than one application user accessing the same data in a database at the same time. The concern on the part of the developer is to provide users with data integrity, that is, to ensure that users have a consistent and valid view of data and do not overwrite the work of others when saving data.

Data Director provides you with concurrency control methods for updating data and also provides settings that you can configure so that your application will have additional control to detect when data in the client-side application has changed relative to the underlying database.

You can add additional code to the **OptimisticDetectChanged** Navigation Control event in order to provide your application with customized warning or error messages.

For more information on concurrency control, see “[Concurrency Control Concepts](#)” on page 5-7, and see the *Informix Guide to SQL: Tutorial*.



Important: *The behavior of the concurrency control methods in Data Director depend on the behavior of the locking methods used in the underlying database and the ODBC driver that you use to access the database. You should be familiar with these concurrency methods for both your database and the ODBC driver.*

Multiple-Developer File Status Indicators

Data Director offers a number of facilities to monitor the status of the files with which you are working. When the status of a file changes (for example, from read-only to read/write) while Data Director is running, that change might not be immediately reflected in the DataLink Manager window.

By default, Data Director checks the status of a file only when the link is first accessed. You can change this setting by using the check box called **On first access, then after delay of 0-10.0 minutes** on the **Source** page of the DataLink Manager Options dialog box. If you set the delay to **0**, the status of the file is checked each time the DataLink Manager needs that information to populate the window.

The easiest way to ensure that all windows reflect a new file status is to select a DataGroup from the **DataGroup** combo box in the DataLink Manager Model pane.

You can also update individual DataLink Manager components without making a new selection. The method is based on the window, as follows:

- For the DataLink Manager **DataGroup** combo box and status bar, click the DataLink Manager window.
- For the DataGroup, Table, and Column Properties dialog boxes, close and reopen the Properties dialog box.
- For the DataLink List pane, change a filter option in the DataLink List Filter Options dialog box.
- For the DataLink Properties dialog box, close and re-open the dialog box after the DataLink List pane is updated.

Running and Testing a Data Director Application

Starting a Data Director Application	9-3
Logging On to Your Database	9-4
Enabling Smart Logon for a DataGroup	9-4
Running Your Application	9-5
Navigating Through a Result Set.	9-5
Setting Record Bookmarks	9-6
Performing Data Operations	9-6
Querying Data.	9-7
Using Query-by-Example	9-7
Creating New Records	9-9
Saving Data.	9-9
Deleting Data	9-9
Master-Detail Coordination	9-10
Using Lookup Data	9-10
Querying Lookup Data.	9-10
Refreshing Reference Data	9-11
Automatic Foreign Key Selection	9-11
Sorting Lookup Data	9-11
Data Loss Alerts	9-12
Using Error Handling	9-12
Error Handling in Data Director	9-12
Customizing the Default Error Handling Behavior	9-13
Using the Data Director Objects.	9-13
Standard Integration with Visual Basic.	9-14
Customizing Runtime Error Messages.	9-14
Using the Trace File.	9-15
Locating and Opening a Trace File	9-15
Trace File Format	9-16
Reusing SQL Statements	9-16

Data Director not only helps you develop applications faster, but also provides your application with a suite of runtime data management features. This chapter describes the default runtime features of a Data Director application.

Starting a Data Director Application

By default, Data Director does not affect your application's start-up. However, under the following circumstances, Data Director will display the Data Director Logon dialog box automatically before any other start-up events occur:

- The **Auto Query** DataGroup option is selected.
If the **Auto Query** option is selected for any DataGroup in the application, Data Director automatically displays the Data Director Logon dialog box when the first application window with DataLinks is opened (see [“Setting Query Properties” on page 5-20](#)).
- Your application contains reference links.
If reference links are present in the application, Data Director exhibits **Auto Query** behavior by displaying the Data Director Logon dialog box and then automatically querying the lookup data (see [“Using Lookup Data” on page 9-9](#)).

Logging On to Your Database

Data Director automatically manages database connections. Before executing a database operation, you must be logged on to your database. If any of the following data operations is attempted before logging on, Data Director automatically displays the default logon dialog box for the DBMS in use:

- Query operation
- Record navigation
- Save operation

Enabling Smart Logon for a DataGroup

Data Director maintains one logon per database for an application per DataGroup. If an application contains multiple DataGroups, multiple logons are established. If an application uses multiple databases and the **Smart Logon** option is selected for a DataGroup, Data Director uses the logon information specified when logging on to the first database to automatically logon to subsequent databases.

If the current logon information is incorrect for a subsequent database, Data Director displays the Data Director Logon dialog box for the appropriate DBMS.

Running Your Application

Data Director provides an application with a number of runtime data management features. The moment you create a DataGroup and add a Navigation Control, your application has the capability to accommodate key user activities:

- Navigating through a result set
- Setting record bookmarks
- Performing data operations
- Querying lookup data

This section discusses how Data Director provides each of these data management features; it assumes that your project contains one or more Navigation Controls.



***Tip:** If you prefer to use your own navigation controls, you can implement the same functionality in a project by using the Data Director Objects.*

Navigating Through a Result Set

Record navigation enables you to easily browse through a result set. For example, an application window with a Navigation Control enables end users to navigate through records in a result set, including quickly navigating to the first or last record in the result set.




If you are not already logged on to your database, clicking any of the Navigation Control buttons or calling an equivalent Data Director Object causes the default Data Director Logon dialog box to appear.

The navigation buttons are described in [“Adding Navigation Buttons” on page 7-10](#).

Setting Record Bookmarks

Record bookmarks allow you to navigate rapidly through a subset of a result set. Bookmarks do not change your data and they persist only until you query your data again. As soon as you query again, the bookmarks are removed.

Each bookmark button is described in the following table.

Icon	Button	Description
	Set Bookmark	Sets a bookmark on the current record. This button appears pressed in each time you navigate to a bookmarked record. The Set Bookmark button is a toggle switch; clicking this button when the current record is already bookmarked removes the bookmark.
	Previous Bookmark	Navigates to the previous bookmarked record relative to the current record.
	Next Bookmark	Navigates to the next bookmarked record relative to the current record.

Performing Data Operations

By default, a Navigation Control provides the following data operations:

- Query
- Query-by-Example (QBE)
- New
- Save
- Delete

This section explains each data operation.

Querying Data

Data Director automatically queries data when a control on a form requests data from the database. This action generally occurs when an application form is first displayed in a running application.



Query button

If you click the **Query** button in the Navigation Control, Data Director clears the current result set and queries all data for the Navigation Control's selected DataGroup and table. Any related tables are also queried, if a link to those tables exists.

If you have not already logged on to your database, the Data Director Logon dialog box appears after you click the **Query** button.

Using Query-by-Example

Query-by-Example (QBE) enables you to query a specific subset of the data in your database, based on search criteria you enter into specific fields of an application window.



QBE button

When you click **QBE**, Data Director switches an application to QBE mode to allow you to enter search criteria. You can resume standard run mode at any time by completing your query or cancelling QBE mode.

The status display on a Navigation Control automatically indicates when an application is in QBE mode. QBE applies only to the Navigation Control's selected DataGroup and table.

When you switch to QBE mode, Data Director clears the data in all fields, starting with the Navigation Control's selected table, as well as all related data.

If you cancel QBE mode, the cleared data is automatically restored.

Entering a QBE Search Value

Search criteria can be a specific data value or can include a wildcard. Data Director supports standard ANSI wildcard symbols.

The following table describes each wildcard symbol.

ANSI	Description	Example
%	Searches for zero or more characters, including spaces and punctuation	To search for all customer names that begin with the letter <i>C</i> , enter a search value of <i>C%</i> . To search for all customer names ending with <i>Inc.</i> , enter a search value of <i>%Inc.</i>
_	Searches for any single character, including spaces and punctuation	To search for a customer name that begins with the letter <i>C</i> and has five letters, enter a search value of <i>C_ _ _ _</i> .



Important: *QBE searches are case sensitive. A search for *C%* will return different results than *c%*.*

You can enter a search value in any editable field of the form window that uses the DataGroup you specified for the Navigation Control. Check boxes, option buttons, and other non-text fields can also be used as search criteria.



Tip: *To allow users to enter search criteria in non-editable fields, trap the **AfterQBE** event for a Navigation Control, and then set the appropriate fields to be editable. For more information about the Navigation Control events, see [“Writing Event Procedures” on page 7-19](#).*

Cancelling QBE Mode

The **QBE** button is a toggle; to cancel QBE mode at any time, click the button again. If QBE is a labelled button, its label changes from **Start QBE** to **Cancel QBE**.

If you cancel QBE mode, any fields that were cleared when you switched to QBE mode are restored to their previous values and related data is queried again.



New button

Creating New Records

Clicking the **New** button creates a new record for the specified table and related tables. Data Director automatically adds the new record to the current result set and navigates to it.

You can insert a new record at any time. New records appear after the last queried record in a result set. After you save a new record, it remains labelled **New Record** and cannot appear in the appropriate sorting order until you query again.

By default, a new record appears when an application starts. If you execute a query, this new record is deleted if **Auto Query** is not active. For more information, see [“Auto Query” on page 5-20](#).



Save button

Saving Data

Clicking the **Save** button saves all modified and new data for any of the tables of the DataGroup specified for a Navigation Control.

If the **Save Confirmation** option is selected for the Navigation Control's specified DataGroup, Data Director asks you to confirm that you want to save your data before continuing with the operation.



Delete button

Deleting Data

Clicking the **Delete** button deletes the current record for a Navigation Control's DataGroup and table from the database.

If the current record:

- does not include detail records, then Data Director deletes only the current record.
- includes detail records and cascade is specified in the underlying database, then Data Director deletes the current record and all related detail records.
- includes detail records and cascade is not specified in the underlying database, then Data Director displays a dialog box informing you that the delete operation cannot be completed and will be cancelled.

If the **Delete Confirmation** option is selected for the Navigation Control's specified DataGroup, Data Director asks you to confirm the deletion of data before actually deleting the data.

Master-Detail Coordination

Data Director automatically performs master-detail coordination in a running application. You need not write any additional code to implement master-detail coordination.

When tables are related to one another through a one-to-many relationship, the “one” table represents a master table, and the “many” table represents a detail table.

When an end user queries data for tables that are related to each other through a one-to-many relationship, Data Director automatically queries all detail records for the master table.

Using Lookup Data

In a running application, lookup data enables end users to select a value from a displayed list. Examples of lookup data include lists of countries, employees, or any other list of information that appears in a combo or list box.

Reference links are special DataLinks that perform two distinct functions: they look up the referenced value for the current record and, in the case of a combo box, display a complete list of values when the end user selects the control.

For example, suppose your application associates every title in the **TITLES** table with a publisher. When a user queries title information, the associated publisher is also queried and displayed in a combo box reference link. An end user can modify the publisher associated with a title by selecting a publisher from the displayed list.

Querying Lookup Data

Data Director automatically queries lookup data for all DataGroups that contain reference links in an application. Lookup data is not queried until the first application window with DataLinks is opened.

When Data Director queries lookup data, the Data Director Logon dialog box appears, prompting the user to log on to the appropriate DBMS. If the logon is successful, lookup data is queried and displayed.

After you query, the appropriate value for the current record appears as the selected item in the linked combo box.

If the end user clicks **Cancel** in the Data Director Logon dialog box, or if the database logon is unsuccessful, lookup data that the user would expect to see displayed in the linked combo box is neither queried nor displayed.

Refreshing Reference Data

Once Data Director queries reference data, that data is not queried again unless you programmatically refresh it.

If reference data is not queried, use the Data Director Objects to programmatically refresh queried data for the reference link. You can use the **DataGroups** collection to query the specific table in the DataGroup that is contained in the reference link. For more information, see the [Data Director Objects Guide](#).

Automatic Foreign Key Selection

When you query a column's parent table, the foreign key from the reference link is used to select the current record in the linked combo box.

When an end user selects a value from a combo box reference link, Data Director saves the primary key of the reference table in the foreign key column of the parent table.

For example, suppose an application uses a reference link between a combo box and the NAME column of the **PUBLISHERS** table. When an end user selects a publisher name, Data Director stores the primary key (**PUBLISHERS.PUB_ID**) in the foreign key column (**TITLES.PUB_ID**).

Sorting Lookup Data

By default, queried lookup data is not sorted unless the Visual Basic **Sorted** property was set to **True** when you created the reference link.

To sort lookup data, set the **Sorting** option for the table that contains the reference link using the Table Properties dialog box (see [“Setting DataGroup Table Properties” on page 5-31](#)).

Data Loss Alerts

Data loss alerts ensure that end users are always notified of runtime data operations that, if immediately executed, would result in the loss of any unsaved changes.

For example, if a user modifies a record and then attempts to navigate to another record before saving, Data Director displays a data loss alert to ask if the user would like to save changes before proceeding. Data Director does not display an alert if a record change only applies to the detail table of a master-detail relationship.

At runtime, if a user initiates a data operation before saving changes, Data Director automatically displays a warning message box.

Using Error Handling

Error handling enables you to systematically identify and resolve problems that occur between an application and a database while using Data Director. Using error handling, you can troubleshoot problems that occur in an application by analyzing the cause of an error and then handling the problem in your application code.

For example, you can trap errors that occur in an application and provide logic in your application code to handle the error situation. You can also choose to simply display the error message in a message box.

Error Handling in Data Director

By default, Data Director automatically passes information about errors that occur in an application to Visual Basic. For example, Data Director passes the error number and error text to Visual Basic each time an error occurs. You do not need to write any additional code to handle errors that occur in a Data Director application.

This section provides an overview of the ways in which you can implement error handling in Data Director.

Customizing the Default Error Handling Behavior

There might be times when you want to customize the default error-handling behavior of Data Director to provide additional control over your error-handling functionality:

- You can customize the error display (see [“Setting Error-Handling Properties” on page 5-22](#)).
- You can trap error events using the Data Director Navigation Control (see [“Intercepting Events” on page 7-21](#)).
- You can track error messages using the Data Director trace file (see [“Using the Trace File” on page 9-13](#)).

Using the Data Director Objects

Error handling in Data Director is implemented primarily through the use of the Data Director Objects. You can customize error handling in Data Director using the **Error** object, methods, and properties for the following parent objects:

- **Project** object
- **DataGroup** object
- **Model** object

By default, error handling is enabled for each object. In addition, each error that populates the **Error** object for a DataGroup also populates the **Error** object associated with the DataGroup’s project.

Refer to the [Data Director Objects Guide](#) for detailed information about the Data Director Objects and the methods and properties available to each object.

Standard Integration with Visual Basic

Data Director integrates seamlessly with the error-handling capabilities of Visual Basic. In fact, you do not need to write any additional code to implement error handling in a Data Director application.

When an error occurs in an application, Data Director creates an internal **Error** object and automatically populates the Visual Basic **Err** object with error information. For each error that occurs, Data Director passes the following information to the Visual Basic **Err** object:

- Error number and error text
- Context ID and filename for a Data Director on-line help file
- The source of the error (**cvddo.dll**)

You can use this information in your application code to debug the error situation or you can display the error message in a message box for an application user.

You can identify error messages originating from a Data Director application by the **Source** property of the Visual Basic **Err** object. Based on additional properties that Data Director sends to the **Err** object, you can use Visual Basic **On Error** statements to handle specific error situations. You can also use the Visual Basic **On Error** statements to implement your own application-specific error handler.

Refer to your Visual Basic documentation for more information about using the **Err** object and **On Error** statements.

Customizing Runtime Error Messages

If you need to customize Data Director error messages for an application that you are deploying, open the **ddmsg32.dll** file in a resource editor to edit error messages.

Using the Trace File

To assist you in debugging your applications, Data Director creates and maintains a trace file of operations and database commands that are generated for a project at runtime. For example, you can view the SQL statements that Data Director generates and sends to your database, as well as any errors received from the database.

Use the trace file for the following purposes:

- Track communication with your database.
- Confirm that Data Director is doing what you want it to do. You can review your SQL statements to ensure that you are getting the data you expect.
- Debug your application by reviewing the trace file if you are not returning the correct data result set or are receiving a database error.
- Cut and paste SQL statements to construct virtual tables (see [“Creating Virtual Tables” on page 4-27](#)).

Refer to the [Data Director Objects Guide](#) for more information about using the **Trace** object.

Locating and Opening a Trace File

Data Director names a project's trace file using the corresponding project name and saves the file with a **.trc** extension. For example, a project file called **books.vbp** will have a trace file named **books.trc**.

Trace files are by default placed in same directory as your application.

If you run an untitled project, Data Director creates a trace file with the default file name **projectx.trc**. If you save an untitled project that you have previously run, Data Director renames its trace file from **projectx.trc** to the name of the new project.

To open a trace file

1. Choose **View→Data Director Trace File** from the Visual Basic menu bar, or click the **Trace File** button in the DataLink Manager status bar. You can also open a trace file using any text editor or word processor.



Trace button

Trace File Format

The trace file includes entries for all database commands generated by Data Director. Each entry begins with the time of the entry in square brackets. Following the time, an SQL statement, error message text, or a message appears.

Reusing SQL Statements

You can copy any SQL statement displayed in a trace file and then execute the statement by pasting it into a different tool. For example, you can copy an SQL statement from a trace file and paste it into the **Commands** page of the Table Properties dialog box to create a virtual table. You can also run an SQL statement copied from a trace file in a DBMS-specific database access utility such as INFORMIX-SQL Editor.

Distributing an Application

Deploying Your Application	10-3
Including Data Director and Other Required Files	10-3
Data Director Project Files.	10-4
Data Director Supporting Files	10-4
Visual Basic Files	10-5
Other Microsoft Files	10-5
Using the Visual Basic Application Setup Wizard	10-5
Installing the Application	10-7

Once you design and build a Data Director application, you can distribute it to application end users.

This chapter assumes that you are familiar with the application distribution functionality available in Visual Basic and discusses some of the Data Director-specific issues of distributing a Data Director application.

Deploying Your Application

Use the Visual Basic Application SetupWizard or Setup Toolkit to distribute your Data Director application. These tools enable you to create a setup program for your application, as well as create distribution media or distribution server directories for your application.

Refer to your Visual Basic documentation for more information about using the Visual Basic Application SetupWizard or Setup Toolkit to distribute your application.

Including Data Director and Other Required Files

To distribute an application developed with Data Director, your application must be distributed with the appropriate Data Director, Visual Basic, and Microsoft files.

Data Director Project Files

The following table lists the Data Director project files necessary to deploy your application.

File Type	Description
.exe file	Your Visual Basic application's executable file containing Data Director information
.mlx files	The Data Director Model files used in your application
.ddx file	The Data Director project file



Important: The **.ddx** filename must match your application's name. For example, if your application is named **MyApp.exe**, the file **MyApp.ddx** must be included.

Data Director Supporting Files

The **deploy32\dd** directory located on the Data Director CD contains the following required Data Director files. These files can also be found in the directory where Data Director is installed (for example, **C:\Program Files\Informix\Data Director**).

- **choreo32.dll**
- **cvlib32.dll**
- **ddmsg32.dll**
- **ddodbc32.dll**
- **vbase32.dll**

The following required Data Director files can be found in the directory where Data Director is installed (for example, **C:\Program Files\Informix\Data Director**). These files are automatically included in your application's installation when using the Visual Basic Application Setup Wizard.

- **navctl32.ocx**
- **cvddo.dll**

Visual Basic Files

Please see your Visual Basic Documentation for additional files needed. The Visual Basic Application SetupWizard automatically includes most of the files required by Visual Basic.

Other Microsoft Files

The **deploy\share** directory located on the Data Director CD contains the following required Microsoft Foundation Classes (MFC) DLLs. These files can also be found in your Windows system directory.

- **mfc42.dll**
- **msvcrt.dll**
- **msvcirt.dll**

Using the Visual Basic Application SetupWizard

The SetupWizard enables you to create distribution diskettes or a distribution server directory for your Visual Basic application. The SetupWizard compresses your project files, creates an executable (**setup.exe** file), copies the distribution files to the distribution diskettes, and performs all other steps necessary to distribute an application.

To distribute an application using the SetupWizard

1. Create and save your project with Visual Basic and Data Director to generate a final Data Director project (**.ddx**) file.

Important: *Do not rename your project after saving your project for the final time. If you rename your project, save the entire project again.*

2. Launch the Visual Basic Application SetupWizard.



VB 4

VB 5

VB 5

VB 4

3. In Step 1 of the SetupWizard, perform the following tasks:
 - a. Enter or select the filename of your Visual Basic project (.vbp) in the **Project File** text box.
 - b. Rebuild the project:
Check the **Rebuild the Project's EXE File** check box. ♦
Check the **Rebuild the Project** check box. ♦
 - c. Click **Next** to proceed.
If you are using Visual Basic Version 4.0a, the SetupWizard proceeds to Step 3 (5 in this procedure list).
4. In Step 2, specify the distribution method (diskettes, one directory, and so on), and then click **Next** to proceed. ♦
5. In Step 3, select the disk and directory where you want to place the application setup files, and then click **Next** to proceed.
6. In Step 4, add any ActiveX (OLE) server components that your application requires, and then click **Next** to proceed.
The Data Director file **cvddo.dll** should be available and selected. If it is not, locate and select it before clicking **Next**.
7. In Step 5, the SetupWizard displays any dependent files that are found for your application; click **Next** to proceed.
If you use the Data Director Navigation Control in your application, the **navctl32.ocx** file appears in the list.
If you are using Visual Basic Version 5.0, the SetupWizard proceeds to Step 7 (9 in this procedure list).
8. In Step 6, specify whether your application is an OLE automation shared component or server. ♦

9. In Step 7, you can view, add, and delete files from the distribution list that will be used with your application's installation program; include all required Data Director, Visual Basic, and Microsoft files.

See [“Including Data Director and Other Required Files” on page 10-3](#). Ensure that you include your project's **.ddx** and **.mlx** files and that all the Data Director **.dll** files and all the Microsoft **.dll** files are in the list.

You might be prompted to find one of the Data Director **.dll** files. If so, navigate to the directory into which you installed Data Director; for example, **C:\Program Files\Informix\Data Director**. Once the SetupWizard has that path, it will continue to find the Data Director **.dll** files in the specified directory.



Warning: When you add the MFC files, they are configured to automatically install into the Windows system directory. Do not change the default location of these files.

When you add the Data Director supporting files, they are configured to automatically install into the Windows system directory. The only exception is the **cvddo.dll** file, which installs into a Windows shared OLE server directory. Use the default installation destinations for all files. If you choose to specify different destination directories for any files, these directories must be specified in the application user's path for proper registration of the **navctl32.ocx** and **cvddo.dll** files.

10. Click **Finish** to complete the setup process.

Installing the Application

Before installing your deployed application, ensure that the 32-bit ODBC driver is installed and the corresponding ODBC data source is configured on the user's PC.

ODBC DataDriver Support

The ODBC DataDriver enables your database to interface with Data Director. This appendix describes the ODBC DataDriver and explains how to use it.

Open Database Connectivity (ODBC)

The DataDriver for Open Database Connectivity works in conjunction with an ODBC driver that you provide. ODBC drivers are available through many database and third-party vendors.

As long as you have previously installed a database-specific ODBC driver, the Data Director DataDriver for ODBC enables you to:

- connect to most database products on the market.
- access flat-file data sources such as Access and Paradox the same way that SQL databases such as INFORMIX-OnLine databases are accessed.
- take advantage of functionality available in an underlying database using ODBC, such as working with system-generated keys, blob values, and stored procedures.

The ODBC DataDriver has an entry in the Data Director section of the system registry and enables database connectivity. For example, specifying ODBC when you import a data source or create a Model tells Data Director to connect to the specified data source through ODBC.

ODBC Compliance Level

The Data Director ODBC DataDriver takes advantage of the functionality available in the ODBC Level One and Level Two APIs.

Software Requirements

To use the Data Director DataDriver for ODBC, you must have the following software programs:

- **Microsoft ODBC Driver Manager.** Available from Microsoft and other sources, the ODBC Driver Manager acts as an intermediary between your application and your ODBC driver. The ODBC Driver Manager routes calls from your application to your ODBC driver, and then returns results from your ODBC driver to your application.
- **Third-party ODBC driver.** Database-specific drivers are available through a variety of sources, or you can write your own. ODBC drivers support the ODBC API and ODBC SQL data types.

Although Data Director supports any ODBC-compliant database, INFORMIX-OnLine databases are the default. To use Data Director with a non-Informix database, contact your ODBC or database vendor for the appropriate drivers. Note that ODBC drivers from various vendors might vary in terms of their coverage and conformance to the ODBC specification.

Glossary

bookmarks	Bookmarks allow you to navigate rapidly through a subset of a result set. Bookmarks do not change your data and they persist only until you query your data again.
cached data	To expedite data access, Data Director caches all data on the client side. Data Director can cache to file or memory.
collection	Some Data Director Objects, like Engine , Model , and Project objects, own sets or <i>collections</i> of other objects. For example, the Engine object contains a <i>collection</i> of Project objects and a <i>collection</i> of Model objects.
concurrency	Concurrency concerns the problem of multiple applications accessing the same data at the same time. A concurrency scheme addresses this problem by allowing you to set a lock on the data (the pessimistic scheme), do nothing (optimistic no detect) or detect if this data has changed upon update (optimistic detect).
CRC	Cyclical Redundancy Code (CRC) is a highly reliable error-detecting code. Data Director uses CRCs to detect if cached client rows have become outdated in comparison to the data source.
custom SQL	Custom SQL allows you to execute custom SQL statements directly through Data Director to your database.
Data Director Objects (DDO)	Data Director Objects are a set of programmable objects that encapsulate the specifics of back-end databases. They enable Visual Basic programmers to develop object-oriented database applications seamlessly within the Visual Basic development environment.
data path	Data paths specify a unique path from the master table to the individual columns to be queried.

DataGroup	The DDO DataGroup object is a collection of DataLinks . The DataGroup controls all of the data access options, including caching, concurrency, query options, logon options, error handling at the DataGroup level, and transaction management. All DataGroups have master tables. The master table is the table in the database that contains the primary information for a particular DataGroup .
DataLink	The DDO DataLink object identifies which columns Data Director should query. All DataLinks have an associated data path. Data paths specify a unique path from the master table to the individual columns to be queried. When you create a DataLink , you must also specify the data path that starts with the master table and ends with the column you want to query.
detail data	Detail data is data related to the master table and linked to the master table by DataLinks .
detection granularity	Detection granularity concerns the amount of data that Data Director compares in the cached result set to a database to determine if the underlying data has changed when updating data. This setting can only be used when concurrency control is set to optimistic detect.
Engine	The DDO Engine object is the root of the DDO hierarchy. It contains a collection of Project and Model objects.
instantiate	To create an instance of.
log off	To terminate the current user session on the database, use the oDataGroup.Logoff method. This logoff method also terminates all database connections and accepts no parameters.
log on	Before you can query data, you must first log on to the database. The logon method also accepts varUser , varPassword , varServer , varDatabase , varDBMSName , and varDataSource .
lookup data	In a running application, lookup data enables end users to select a value from a displayed list.
master table	The master table is the table in the database that contains the primary information for a particular DataGroup . Data Director starts all queries for a particular DataGroup at the master table.

Model	Models are the starting point for Data Director user interface (not DDO) applications. Models contain all the database structure information used by Data Director to implement data access and data management. Specifically, a Model stores information about table names, column names, relationships, table owners, column sizes, column types, primary key attributes, and foreign keys.
navigation	There are four main methods for navigating through a result set: <ul style="list-style-type: none"> ■ Table.NextRecord ■ Table.PreviousRecord ■ Table.FirstRecord ■ Table.LastRecord
Project	The DDO Project object exists primarily to organize a collection of DataGroups. It can also be used to control error reporting at the project level.
query data	To transfer data from the database server to your local cache, “query,” use the oDataGroup.QueryData method.
ReferenceGroup	A ReferenceGroup is a DataGroup that contains all of the records associated with a many-to-one relationship. Data Director automatically creates a ReferenceGroup when you create a many-to-one DataLink.
synchronization	Because Data Director caches all data, multiple DataLinks to the same columns point to the same cached data value. Therefore, changes to the cached data value are propagated throughout all DataLinks that refer to it.
virtual link	A virtual link is a DataLink to a column rather than a specific link between a column and control. The data is retrieved directly from the Column object property on the DataLink and does not appear in an application.
virtual table	A virtual table is a table that contains the result set for a custom SQL statement.

Index

A

Ad hoc queries, using 4-27
 Aggregate columns, creating 4-39
 Aliases
 defining for columns 4-34
 defining for tables 4-27
 Applications
 compiling 2-12
 designing 2-6
 designing multiuser
 applications 8-13
 distributing Data Director 10-3
 example Intro-8
 initializing 7-18
 running 2-12, 9-3
 starting 9-3
 testing 2-12
 Auto query 5-20
 Auto size options 3-4
 AVG, calculated columns 4-40

B

Blob properties 5-29
 Blob values 5-29
 Bookmark buttons
 adding to a Navigation
 Control 7-9
 using 9-6
 Bound controls 6-6
 Business processes, defining 5-14

C

C++ applications, using Data
 Director with Intro-8

Calculated columns

AVG 4-40
 count 4-40
 creating 4-39
 date 4-40
 date functions 4-40
 entering SQL text 4-40
 math operators 4-40
 MAX 4-40
 MIN 4-40
 SUM 4-40

Calculation functions

count 4-40
 date and time values 4-36
 types 4-40

Changed row indicator,
 specifying 4-37, 5-12

Changing tables 4-32

Check boxes

DataLink properties 6-21
 using with Data Director 6-5

Checking in files 8-7

Checking out files 8-6

Colors, changing on the Navigation
 Control 7-17

Column attributes, updating 4-15

Column Properties dialog box 4-34,
 5-37

Columns

choosing a data type 4-35
 creating 4-33
 creating aggregate columns 4-39
 creating calculated columns 4-39
 creating non-aggregate
 columns 4-39
 defining default values 4-35
 dropping old columns 4-11, 4-15
 filtering 3-16
 modifying 4-41
 overview 4-32

- specifying a column size 4-35
- specifying a primary key 4-37
- specifying an alias 4-34
- updating column attributes 4-15
- Combo boxes 6-5
- Command buttons
 - DataLink properties 6-23
 - using with Data Director 6-5
- Concurrency control
 - changed row indicator 4-37
 - changed row indicator
 - method 5-12
 - CRC method 5-12
 - granularity detection
 - methods 5-13
 - optimistic 5-9
 - overview 5-7
 - pessimistic 5-14
 - queried/cached columns
 - method 5-13
 - read consistency 5-9
 - setting detection methods 5-27
 - setting granularity methods 5-27
 - setting up 5-26
 - update columns only
 - method 5-13
 - value method 5-12
- Concurrency properties
 - detection granularity 5-26
 - detection method 5-26
 - optimistic, detect 5-26
 - optimistic, no detect 5-26
- Conditions
 - condition operators 5-32
 - setting 5-31
- Confirmation options 3-10
- Connecting to databases 4-21
- Connection properties 5-25
- Constants, default values for 4-36
- Control arrays, creating DataLinks to 6-10
- Controls
 - check boxes 6-5
 - combo boxes 6-5
 - command buttons 6-5
 - DB combo 6-5
 - DB grid 6-5
 - DB list 6-5
 - filtering 3-16

- list boxes 6-5
- option buttons 6-5
- scroll bars 6-5
- text boxes 6-5
- Count, calculated columns 4-40
- CRC method, setting 5-12, 5-27
- Create Relationship dialog
 - box 4-46
- Creating
 - calculated columns 4-39, 4-40
 - columns. *See* Creating columns.
 - DataGroups 5-14
 - DataLinks 6-6
 - Models 4-3
 - Navigation Controls 7-7
 - new records 9-9
 - relationships. *See* Creating relationships.
 - tables 4-23, 4-26
 - virtual tables. *See* Creating virtual tables.
- Creating columns
 - choosing a data type 4-35
 - creating mandatory columns 4-33
 - entering a name 4-33
 - overview 4-33
 - selecting a primary key
 - column 4-37
 - specifying a changed row indicator 4-37
 - specifying a size 4-35
 - specifying a system generated key 4-37
 - specifying an alias 4-34
- Creating relationships
 - overview 4-45
 - selecting a relationship type 4-47
 - specifying destination
 - columns 4-46
 - specifying origin columns 4-46
- Creating virtual tables
 - checking your syntax 4-30
 - overview 4-27
 - specifying SQL commands 4-29
- Custom button, adding to a Navigation Control 7-9

D

- Data
 - automatically committing 5-24
 - creating new records 9-9
 - data loss alerts 9-12
 - deleting 9-9
 - master-detail coordination 9-10
 - navigating 9-5
 - querying 9-7
 - saving 9-9
 - setting bookmarks 9-6
 - using lookup data 9-10
- Data Director
 - components 2-5
 - data access 2-4
 - data management 2-4
 - installing 1-4
 - Logon dialog box 9-3
 - project files 8-11
 - starting 1-10
 - system requirements 1-3
 - Tutorials 1-20
- Data paths
 - modifying 6-16
 - multiple paths 6-13
 - overview 6-13
 - selecting 6-15
 - syntax 6-19
- Data source, selecting 4-12
- Data types
 - defining for columns 4-35
 - updating 4-11, 4-15
- Data value synchronization, setting up 6-19
- Database drivers, selecting 4-12
- Database properties 5-24
- Databases
 - automatically logging on 5-25
 - connecting 4-21
 - disconnecting 4-21
 - example, used in tutorials Intro-5
 - importing 2-7, 4-6
 - importing foreign keys 4-10
 - importing primary keys 4-10
 - importing system objects 4-11
 - importing tables 4-10
 - importing views 4-10
 - logging on 4-13, 9-3

- selecting 4-13
- setting DataGroup
 - properties 5-24
- updating 4-11, 4-22
- updating Models 4-5
- DataDrivers, Open Database
 - Connectivity (ODBC) A-1
- DataField property 6-6
- DataGroup combo box 1-15
- DataGroup files, checking out 8-11
- DataGroup pane
 - filtering 3-16
 - overview 1-15
- DataGroup properties
 - blob support properties 5-29
 - concurrency properties 5-26
 - connection properties 5-25
 - database properties 5-24
 - error handling properties 5-22
 - join properties 5-29
 - query properties 5-20
- DataGroup Properties dialog
 - box 5-19
- DataGroups
 - assigning to a Navigation
 - Control 7-8
 - business processes 5-14
 - creating 5-14
 - DataGroup files 5-6, 8-11
 - deleting 5-16
 - error handling 9-13
 - master table overview 5-5
 - master-detail coordination 9-10
 - organizing data 5-5
 - overview 5-3
 - recovering 5-18
 - referencing Models 5-4
 - setting column properties 5-37
 - setting properties 5-19
 - setting table properties 5-31
- DataLink Manager
 - DataGroup combo box 1-15
 - DataGroup pane 1-15
 - DataLink pane 1-16
 - options. *See* DataLink Manager
 - options.
 - starting 1-10
 - status bar 1-18
 - toolbar 1-16
 - user interface 1-13

- window overview 1-13
- DataLink Manager options
 - auto size options 3-4
 - confirmation options 3-10
 - display options 3-8
 - source options 3-11
 - status bar options 3-6
- DataLink pane
 - filtering 3-16
 - overview 1-16
- DataLink properties
 - check boxes 6-21
 - command buttons 6-23
 - option buttons 6-20
 - setting 6-19
- DataLinks
 - creating 2-9
 - creating reference links 6-11
 - creating to control arrays 6-10
 - creating to option buttons 6-10
 - deleting 6-12
 - filtering 3-16
 - overview 6-3
 - replacing 6-12
 - supported controls 6-5
- DataSource property 6-6
- Date, calculated columns 4-40
- DB combo boxes 6-5
- DB grids 6-5
- DB lists 6-5
- DBMS
 - selecting 4-13
 - setting 4-20
- .ddx files 8-11, 8-12, 10-4
- Decimal, numeric size 4-35
- Default values
 - overriding Model settings 5-37
 - setting 5-37
 - setting for columns 4-35
 - using calculation functions 4-36, 5-38
 - using constants 4-36, 5-38
- Delete button, adding to a
 - Navigation Control 7-9
- Delete confirmation 5-20
- Deleting data
 - overview 9-9
 - primary keys 4-16
- Deleting DataGroups 5-16

- Deploying Data Director
 - applications 10-3
- Designing an application 2-6
- Destination columns,
 - relationships 4-42
- Detection granularity, setting 5-27
- .dgp files 5-6, 8-11
- Disconnecting from databases 4-21
- Disk space requirements 1-3
- Display options 3-8, 3-14
- Distributing Data Director
 - applications 10-3
- Drivers, selecting 4-12

E

- Error handling
 - DataGroup level 9-13
 - default behavior 9-14
 - fatal errors 5-23
 - informational errors 5-23
 - Model level 9-13
 - overview 9-12
 - project level 9-13
 - properties 5-22, 5-23
 - using the Err object 9-14
 - warning errors 5-23
- Error messages
 - customizing 5-22, 9-14
 - displaying 5-22
- Event parameters 7-38
- Examples directory Intro-8
- Existing Relationships dialog
 - box 4-18

F

- Fetching rows 5-21
- Files
 - checking in 8-6, 8-7
 - checking out 8-6
 - required for distributing Data
 - Director applications 10-4
- Filtering
 - DataGroup information 3-16
 - DataLink information 3-16
 - options 3-16
- Fonts, changing on the Navigation
 - Control 7-16

Foreign keys
 importing 4-10
 inferred relationships 4-17
Form files
 checking out 8-9
 storing Data Director
 information 8-9
Forms, filtering 3-16
.frm files 8-9
Full outer joins 5-29

G

Getting on-line help 1-22
Grid controls, third-party Intro-8
Grids, creating DataLinks to 6-5
Group-by clauses, using 4-27

H

Hardware requirements 1-3
Help, on-line 1-22
Horizontal scroll bars 6-5

I

Import items, selecting 4-14
Import Options page 4-10
Importing Models
 import options 4-9
 importing foreign keys 4-10
 importing primary keys 4-10
 importing system objects 4-11
 importing tables 4-10
 importing views 4-10
 inferring relationships 4-11
 Model Import Wizard 4-8
Inferred relationships
 overview 4-17
 verifying 4-11, 4-17
Initializing an application 7-18
Installing Data Director
 how to install 1-4
 requirements 1-3
Installing deployed Data Director
 applications 10-7

J

Join properties, outer joins 5-29

L

Launching Data Director 1-10
Layout options, Navigation
 Control 7-8
Left outer joins 5-29
List boxes 6-5
Locking, overview 5-8
Logging on
 auto query 9-3
 automatic logons 5-25
 automatically logging on 9-3
 during import 4-13
 full logon prompting 4-12
 Logon dialog box 5-25
 smart logon 9-4
 timeout values 5-24
 using the Data Director Logon
 dialog box 9-3
Login timeout 5-24
Lookup data
 creating DataLinks 6-11
 using 9-10

M

Many-to-one relationships 4-42
Master tables 5-5
Master-detail coordination 9-10
Math operators, calculation
 columns 4-40
MAX, calculated columns 4-40
.mdt files 8-10
.mdx files
 generating 8-12
 overview 8-10
Memory requirements 1-3
Methods of detection, setting 5-27
Microsoft Foundations Classes used
 for distributing Data Director
 applications 10-5
MIN, calculated columns 4-40
Model files 8-10
Model Import Wizard 4-9

Model pane 1-15
Model View pane 1-11
Model Viewer
 Model View pane 1-11
 starting 1-10
 Table Relationship pane 1-11
 toolbar 1-12
 viewing Models 4-19
 window overview 1-10
Models
 adding tables 4-26
 adding virtual tables 4-27
 creating 2-7
 dropping old table columns 4-15
 error handling 9-13
 overview 4-3
 planning an import 4-7
 properties 4-23
 refreshing 4-5, 4-11, 4-22
 sharing 4-5
 updating 4-5, 4-22
 updating column attributes 4-15
 using multiple 4-7
 viewing 4-19
Modify Relationship dialog
 box 4-46
Modifying data paths 6-16
Multi-developer
 about Visual Basic files 8-5
 building multi-developer
 projects 8-3
 checking in files 8-6, 8-7
 checking out files 8-6
 using source code control
 systems 8-3
 working with Data Director
 project files 8-11
 working with DataGroup
 files 8-11
 working with form files 8-9
 working with Model files 8-10
 working with Visual Basic project
 files 8-8
Multiple column relationships 4-47
Multiple data paths 6-13
Multi-user applications,
 designing 8-13

N

- navctl32.ocx file 7-6
- Navigation buttons
 - adding to a Navigation Control 7-9
 - using 9-5
- Navigation Control
 - adding a Custom button 7-9
 - adding a Delete button 7-9
 - adding a New button 7-9
 - adding a QBE button 7-9
 - adding a Query button 7-9
 - adding a Save button 7-9
 - adding a status panel 7-9
 - adding bookmark buttons 7-9
 - adding navigation buttons 7-9
 - adding to a form 7-7
 - adding to a project 7-6
 - assigning a DataGroup 7-7
 - assigning a table 7-7
 - changing the button size 7-15
 - changing the color 7-17
 - changing the font 7-16
 - creating 2-10, 7-7
 - customizing the appearance 7-15
 - events 7-19
 - grouping like items 7-15
 - initializing an application 7-18
 - objects 7-57
 - overview 7-5
 - performing data operations 9-6
 - properties 7-40
 - specifying a layout option 7-8
 - using bookmark buttons 9-6
 - using multiple controls 7-19
 - using navigation buttons 9-5
- New button, adding to a Navigation Control 7-9
- Non-aggregate columns, creating 4-39
- Number of rows per fetch, setting 5-20
- Numeric sizes 4-35

O

- Objects, Navigation Control 7-57
- One-to-many relationships 4-42
- On-line examples Intro-8
- On-line help Intro-8, 1-22
- On-line manuals Intro-8
- Open Database Connectivity (ODBC), Microsoft Driver Manager A-2
- Optimistic concurrency control 5-9
- Option buttons
 - creating DataLinks to 6-10
 - DataLink properties 6-20
 - using with Data Director 6-5
- Options
 - DataLink Manager options 3-3
 - filtering options 3-16
 - import options 4-9
 - project options 3-12
- Origin columns, relationships 4-42
- Outer joins 5-29

P

- Pessimistic concurrency control 5-14
- Precision, numeric size 4-35
- Primary key column, specifying 4-37
- Primary keys
 - importing 4-10
 - inferred relationships 4-17
 - setting 4-21
 - specifying 4-37
 - used for deletes 4-16
 - used for updates 4-16
 - verifying 4-11, 4-16
- Project files
 - checking out 8-8
 - compiling 8-12
 - storing Data Director information 8-8
- Project trace options 3-12
- Projects
 - compiling 8-12
 - error handling 9-13
- Properties, Navigation Control 7-40

Q

- QBE button, adding to a Navigation Control 7-9
- QBE (Query-by-Example), using 9-7
- Query button, adding to a Navigation Control 7-9
- Query properties 5-20
- Query timeout 5-24
- Querying data
 - lookup data 9-10
 - overview 9-7
 - using auto query 5-20
 - using smart query 5-20

R

- Read consistency 5-9
- Record numbers, unique 4-37
- Records
 - creating 9-9
 - deleting data 9-9
 - locking 8-13
 - navigating 9-5
 - saving data 9-9
 - setting bookmarks 9-6
- Recovering deleted DataGroups 5-18
- Reference data, refreshing 9-11
- Reference links
 - creating 6-11
 - using 9-10
- Refreshing Models 4-15, 4-22
- Relationship types
 - importing 4-17
 - many-to-one 4-42
 - one-to-many 4-42
 - one-to-one 4-42
- Relationships
 - creating 4-45
 - destination columns 4-42
 - implied relationships 4-44
 - importing inferred relationships 4-11
 - inferred relationships 4-17
 - many-to-one defined 4-43
 - modifying 4-48

- modifying the key order 4-48
- multiple column
 - relationships 4-47
- one-to-many defined 4-43
- origin columns 4-42
- overview 4-41
- selecting a relationship type 4-46
- selecting destination
 - columns 4-46
- selecting origin columns 4-46
- types 4-42
- verifying inferred
 - relationships 4-17
- Result sets
 - cached on client side 5-9
 - comparing to database data 5-9
 - fetching 9-7
 - navigating 9-5
 - querying 9-7
- Right outer joins 5-29
- Routines, in virtual tables 4-31
- Row set size, setting 5-21
- Running Data Director
 - applications 9-3
- Runtime features
 - cached data 2-15
 - data loss alerts 2-16
 - data value synchronization 2-15
 - look up data 2-16
 - master-detail coordination 9-10
 - running an application 9-3
 - using lookup data 9-10

S

- Save button, adding to a Navigation
 - Control 7-9
- Save confirmation 5-20
- Saving data 9-9
- Scale, numeric size 4-35
- Select Data Path dialog box 6-17
- Selecting a data path 6-15
- Selecting import items 4-14
- Setting conditions 5-31
- SetupWizard 10-5
- Smart login 5-25
- Smart query 5-20
- Software requirements Intro-5, 1-3

- Sorting data 5-34
- Source code control systems
 - checking in files 8-6
 - checking out files 8-6
 - using 8-3
- Source options 3-11
- Source Safe
 - checking in files 8-7
 - checking out files 8-6
 - using 8-6
- SQL commands
 - checking your syntax 4-30
 - entering for virtual tables 4-29
 - executing 4-28
 - using custom commands 4-28
 - using from trace files 9-16
- SQL text, entering for calculated
 - columns 4-40
- Starting Data Director 1-10
- Status bar 1-18
- Status bar options 3-6
- Status panel
 - adding a custom caption 7-11
 - adding to a Navigation
 - Control 7-9
- Stored procedures, using in virtual
 - tables 4-31
- Sub-queries, using 4-27
- SUM, calculated columns 4-40
- Supported controls 6-5
- Syntax, checking SQL
 - commands 4-30
- System generated key,
 - specifying 4-37
- System objects, importing 4-11
- System requirements 1-3

T

- Table properties
 - conditions 5-31
 - sorting 5-34
- Table Properties dialog box 4-27
- Table Relationship pane 1-11
- Tables
 - assigning an owner 4-27
 - assigning to a Navigation
 - Control 7-8

- creating 4-26
- entering a description 4-27
- filtering 3-16
- importing 4-7, 4-10
- modifying 4-32
- naming 4-27
- overview 4-26
- setting conditions 5-31
- sorting data 5-34
- specifying an alias 4-27
- Text boxes
 - DataLink properties 6-19
 - using with Data Director 6-5
- Timeouts 5-24
- Toolbars
 - DataLink Manager 1-16
 - Model Viewer 1-12
- Trace files
 - enabling 3-12
 - locating 9-15
 - opening 9-15
 - specifying a file location 3-12
 - using 9-15
- Trace options 3-12
- Tutorial
 - Data Director tutorial 1-19
 - opening lessons 1-21
 - summary of lessons 1-20

U

- Unique record numbers,
 - generating 4-37
- Updating
 - concurrency control data 8-13
 - Model elements 4-15
 - Models 4-11
 - primary keys 4-16

V

- Value method, setting 5-12, 5-27
- .vbp files 8-8
- Verify Primary Keys dialog
 - box 4-16
- Verifying
 - inferred relationships 4-11
 - primary keys 4-16

- Vertical scroll bars 6-5
- Views, importing 4-10
- Virtual tables
 - assigning an owner 4-28
 - checking syntax 4-29
 - creating 4-27
 - creating columns 4-40
 - entering a description 4-28
 - generated columns 4-40
 - modifying SQL text 4-40
 - naming 4-28
 - overview 4-27
 - using stored procedures 4-31
 - viewing SQL text 4-40
- Visual Basic controls 6-5
- Visual Basic files, storing Data
 - Director information 8-8, 8-9
- Visual Basic SetupWizard 10-5
- Visual Source Safe
 - checking in files 8-7
 - checking out files 8-6
 - using 8-6

