User's Guide

dBASE SE

release **2.0**

for Windows® 95, 98, 2000 NT, ME and XP

dBASE, Inc. Vestal, NY
http://www.dbase.com news://news.dbase.com

Printed in the U.S.A.

Contents

Chapter 1
Introduction to *dBASE SE* **1-1**

Welcome to *dBASE SE* ! 1-1
What is *dBASE SE*? 1-1
dBASE Newsgroups 1-2
The *dBASE SE* Knowledgebase: 1-2
Overview of dBASE SE 1-2
Report objects and the integrated Report designer 1-3
Project Explorer 1-3
Data objects 1-3
Visual designers 1-3
ActiveX integration 1-3
The Inspector 1-3
Full-featured Source editor 1-4
SQL designer 1-4
BDE Administrator and database support 1-4
DBF7 file format features 1-4
Samples and sample viewer 1-4
Source Aliasing 1-5
Dynamic External Objects (DEO) 1-5
dBASE SE documentation 1-5
Typographical conventions 1-6
Documentation updates and additional information resources 1-6
Software registration 1-6

Chapter 2
Installing *dBASE SE* and connecting to an SQL database server **2-1**

What you need to run *dBASE SE* 2-1
HARDWARE 2-1
OPERATING SYSTEM 2-1
NETWORKS 2-1
Products and programs in your *dBASE SE* package 2-1
Installing *dBASE SE* 2-2
What happens during installation 2-2
Un-installing *dBASE SE* 2-3
How to connect to an SQL database server 2-3
Install and configure the server software. 2-3
Configure the Borland Database Engine (BDE) 2-4
Listing SQL tables in the Navigator 2-4

Chapter 3
dBASE dQuery/Web **3-1**

What is dQuery/Web? 3-1
Basic functionality of dQuery/Web: 3-1
dQuery Components and Navigation 3-2
The Live Data Area 3-2
The EasyStart Menu 3-2
The Navigator 3-3
Menus and Toolbars 3-3
Opening, Saving and Creating a New dataModule 3-4
Creating a Database object 3-4
Aliases 3-4
Creating a Database object 3-5
Saving a Database object 3-7
Creating Query objects 3-7
Setting Active Indexes 3-10
Creating a Parent-Child Link 3-11
Entering Data 3-12
Filtering Data 3-14
Working With Custom Views 3-16
No-Click Reports 3-17
One-Click Windows 3-19

Chapter 4
Introduction to programming in dBL4-1

"Hard coding" vs. visual programming 4-1
Advantages of event-driven programs 4-1
How event-driven programs work 4-2
Developing event-driven programs 4-4

Chapter 5
Creating an application **5-1**

Creating an application (basic steps) 5-1
Project files (overview) 5-2
Creating a project file 5-2
Adding files to a project 5-3
Notes on the Project Explorer 5-3
Building the user interface 5-3
Form design guidelines 5-4
Goal of form design 5-4
Purpose of a form 5-4
Some guidelines for data entry forms 5-4
Designing the form layout 5-5
Guidelines for using the z-order 5-5
Creating a form 5-6
Using the Form wizard 5-6
Using the Form designer 5-6
.WFM file structure 5-7
Form class definition 5-8
How the contents are generated 5-8
Editing a .WFM file. 5-8
Editing the header and bootstrap 5-8
Editing properties in the .WFM file 5-9
Types of form windows 5-9
MDI and SDI applications 5-9
Modal and modeless windows 5-10
Customizing the MDI form window 5-10
Standard features of MDI windows: 5-10
Using multi-page forms 5-10
Global page (forms) 5-11
Navigation buttons (form pages) 5-11
Creating a custom form, report, or data module class 5-11
Using a custom class 5-12
Creating custom components 5-13
Creating custom components 5-13
Adding custom components to the Component palette . . . 5-14
Removing custom components from the Component palette 5-14

Chapter 6
Accessing and linking tables **6-1**

The dBASE data model 6-1
Query objects 6-1
SQL property 6-2

rowset property 6-2
Rowset objects 6-2
The row cursor and navigation 6-2
Rowset modes 6-3
Rowset events 6-3
Row buffer 6-3
Field objects 6-3
value property 6-3
Using dataLinks 6-4
Database objects 6-4
Accessing a database 6-4
Database-level security 6-4
Database-level methods 6-4
Default Database object 6-4
Session objects 6-4
StoredProc objects 6-5
DataModRef objects 6-5
Linking a form or report to tables 6-5
Linking to a table automatically 6-6
Linking to a table manually 6-6
Procedure for using a Session object 6-7
Calling a stored procedure 6-7
Using local and remote tables together 6-7
Creating master-detail relationships (overview) 6-7
Using an SQL JOIN statement 6-8
Linking master-detail in local tables 6-8
Using the masterSource property 6-9
What is a dataModule? 6-9
Creating a dataModule 6-10
Creating business rules in a dataModule 6-10
Using a dataModule 6-10

Chapter 7
Using the Form and Report designers 7-1

The designer windows 7-1
Design and Run modes 7-2
The Form Design Window 7-2
The Report Design window 7-3
The visual design is reflected in your code 7-3
Component palette 7-3
Standard page 7-4
Data Access page 7-6
Data Buttons page (forms) 7-7
Report page 7-7
Custom page 7-8
Using ActiveX (*.OCX) controls 7-8
The Field palette 7-8
The Inspector 7-9
Properties page of the Inspector 7-10
Events page of the Inspector 7-11
Methods page of the Inspector 7-11
The Method menu 7-12
Manipulating components 7-12
Placing components on a form or report 7-12
Special case: container components 7-13
Selecting components 7-13
Moving components 7-13
Cutting, copying, pasting, deleting components 7-14
Undoing and redoing in the designers 7-14
Aligning components 7-14
Resizing components 7-15
Spacing components 7-16
Setting a scheme (Form designer) 7-16
Editing a Text object 7-17
Saving, running, and printing forms and reports 7-17
Opening a form or report in Run mode 7-18
Printing a form or report 7-18

Chapter 8
Creating menus and toolbars 8-1

Attaching pulldown menus to forms 8-1
Attaching popup menus to forms 8-1
Creating toolbars and attaching them to forms 8-2
Creating a reusable toolbar 8-2
Attaching a reusable toolbar 8-2
Creating a custom toolbar 8-3
Creating menus with the designers 8-4
The designer menu 8-4
Building blocks 8-4
Adding, editing and navigating 8-5
Features demonstration 8-5
Examining menu file code 8-6
Changing menu properties on the fly 8-7
Menu and menu item properties, events and methods 8-8
Toolbar and toolbutton properties, events and methods 8-9

Chapter 9
Using the Source editor and other code tools 9-1

Using the Source editor 9-1
Two-pane window with tree view 9-2
Notes on the Source editor 9-2
Creating a new method 9-3
The Code Block Builder for editing code blocks 9-3
To create or edit a codeblock 9-3
Editing an existing code block 9-4
The Command window 9-4
Typing and executing commands 9-5
Executing a block of commands 9-5
Reusing commands 9-5
Editing in the Command window 9-5
Saving commands into programs 9-6

Chapter 10
Debugging applications 10-1

Types of bugs 10-1
Using the Debugger to monitor execution 10-1
General debugging procedure 10-2
Debugging runtime applications 10-3
The Source window 10-3
To locate and move to a line number in the Source window 10-4
To find a text string in the current program file 10-4
The Debugger tool windows 10-4
Variables 10-4
Watches 10-4
Call Stack 10-4
Trace 10-4
Docking the Debugger tool windows 10-4
Excluding variable types 10-4
Controlling program execution 10-5
Stepping in the Debugger 10-6

Using breakpoints . . . 10-6
Setting and removing breakpoints . . . 10-6
Working with breakpoints . . . 10-7
Running a program at full speed from the Debugger . . . 10-7
Running to cursor position . . . 10-8
Stopping program execution . . . 10-8
Debugging event handlers . . . 10-8
Viewing and using the Call Stack . . . 10-8
Watching expressions . . . 10-9
Adding watch points . . . 10-9
Editing watch points . . . 10-9
Changing watchpoint values . . . 10-9

Chapter 11
SQL designer 11-1
Opening the SQL designer . . . 11-1
For new queries . . . 11-1
SQL designer elements . . . 11-1
Interacting with the Source editor . . . 11-2
Entering data in the SQL designer . . . 11-2
Running a query from the SQL designer . . . 11-2
Putting your queries to work . . . 11-2
Using your .SQL files with the SQL Property Builder . . . 11-3
Looking at the table pane . . . 11-3
About the table boxes . . . 11-3
Adding tables in the SQL designer . . . 11-3
Renaming a table . . . 11-3
Removing a table . . . 11-3
Selecting fields in the SQL designer . . . 11-4
Selecting all fields in a table . . . 11-4
Selecting individual fields in a table . . . 11-4
Reordering selected fields . . . 11-4
Criteria page (SQL designer) . . . 11-4
Deleting a row . . . 11-5
Adding selection criteria in the SQL designer . . . 11-5
Specifying selection criteria . . . 11-5
Simple Equation . . . 11-5
SQL Expression . . . 11-6
EXISTS Clause . . . 11-6
Combining selection criteria . . . 11-6
Row info . . . 11-6
Criteria combo box . . . 11-6
Grouping selection criteria in the SQL designer . . . 11-7
Drill-down column . . . 11-7
Query operators . . . 11-8
Selection page of the SQL designer . . . 11-8
Selecting a field . . . 11-9
Specifying an output name . . . 11-9
Producing summary data . . . 11-9
Removing duplicate rows . . . 11-9
Deleting a row . . . 11-9
Grouping page of the SQL designer . . . 11-9
Creating a grouped query . . . 11-9
Group criteria page of the SQL designer . . . 11-9
Adding group selection criteria . . . 11-10
SQL Expression . . . 11-10
Simple Having Summary Expression . . . 11-10
Two Summary Expression . . . 11-10
Combining group criteria . . . 11-10
Deleting a row . . . 11-11
Sorting page of the SQL designer . . . 11-11
Joins page of the SQL designer . . . 11-11
Including Unmatched Rows . . . 11-11
Join list box . . . 11-11
Joins grid . . . 11-12
Deleting a join . . . 11-12
Deleting a row . . . 11-12
Creating joins in the SQL designer . . . 11-12

Chapter 12
Designing reports 12-1
Report wizard 12-1
To use the Report wizard 12-2
Example of a report created with the Report wizard 12-3
Wizard-generated Summary Report 12-4
Report designer elements 12-5
The Report and Group panes 12-5
Modifying report in the Report designer 12-6
Deleting columns (fields) from a report 12-6
Adding columns (fields) to a report 12-6
Suppressing duplicate field values 12-7
Displaying default values in a blank report field 12-7
Adding a floating dollar sign to field values in reports 12-7
Adding page numbers 12-8
Drill-down reports 12-8
Adding standard components to a report 12-9
Changing the report's appearance 12-9
Creating report borders 12-9
Setting background color in reports . . . 12-10
Setting background image in reports . . . 12-10
Performing aggregate (summary) calculations . . . 12-10
Designing a report with multiple streamFrames . . . 12-11
Creating printed labels . . . 12-11

Chapter 13
Introduction to designing tables in *dBASE SE* 13-1
Terms and concepts 13-1
Table design guidelines 13-2
Identifying the information to store 13-2
Classifying information 13-3
Determining relationships among tables 13-3
Single versus multiple tables 13-3
One-to-one and one-to-many relationships 13-3
Parent and child tables 13-4
Minimizing redundancy 13-4
Choosing index fields 13-4
Defining individual fields 13-5
Table structure concepts 13-5
Table names 13-5
Table types 13-5
Field types 13-5

Chapter 14
Creating tables 14-1
Supported table types 14-1
Using the Table wizard 14-2
Using the Table designer 14-2
Table designer tips 14-3
User-interface elements in the Table designer 14-3
Resizing columns 14-4
Getting around in the Table designer 14-4
Adding and inserting fields 14-4

Moving fields . . . 14-4
Deleting fields . . . 14-4
Saving the table structure . . . 14-5
Abandoning changes . . . 14-5
Restructuring tables (overview) . . . 14-5
Important guidelines for restructuring . . . 14-5
Modify structure rules . . . 14-6
Changing the structure . . . 14-6
Printing the table structure . . . 14-7
Table access passwords . . . 14-7
Creating custom field attributes . . . 14-7
Specifying data-entry constraints . . . 14-8
Creating and maintaining indexes . . . 14-8
Indexing versus sorting . . . 14-8
Sorting or exporting rows . . . 14-9
dBASE index concepts . . . 14-10
Planning indexes . . . 14-10
Using indexes in data entry . . . 14-10
Using indexes in queries . . . 14-11
Using indexes in reports . . . 14-11
Using indexes to link multiple tables . . . 14-11
Creating a simple index . . . 14-12
Using the Table designer to create a simple index . . . 14-12
Using the Manage Indexes dialog box to create a simple index . . . 14-12
Selecting an index for a rowset . . . 14-12
Index tasks . . . 14-13
Modifying indexes . . . 14-13
Deleting indexes . . . 14-13
Indexing on a subset of rows for dBASE tables . . . 14-13
Hiding duplicate values . . . 14-13
Creating complex indexes for dBASE tables . . . 14-14
Rules for dBASE complex indexes . . . 14-14
Creating the dBASE complex index . . . 14-14
Key expressions . . . 14-15
Primary and secondary indexes . . . 14-15
Unique keys . . . 14-15
Secondary indexes, maintained and non-maintained . . . 14-15
Creating primary indexes . . . 14-16
Creating secondary indexes . . . 14-16
Referential integrity . . . 14-16
Defining referential integrity . . . 14-16
Update and delete behavior . . . 14-17
Changing or deleting referential integrity . . . 14-18

Chapter 15
Editing table data 15-1
A few words of caution . . . 15-1
Running a table . . . 15-1
Protected tables . . . 15-2
Table tools and views . . . 15-2
Table and query views . . . 15-2
Adjusting the view . . . 15-3
Viewing only selected table data . . . 15-3
Table navigation . . . 15-4
Data entry considerations . . . 15-4
Finding and replacing data . . . 15-5
Searching tables . . . 15-5
Replacing data in rows . . . 15-6
Adding rows to a table . . . 15-7
Deleting rows . . . 15-7
Saving or abandoning changes . . . 15-8
Performing operations on a subset of rows . . . 15-8
Selecting rows by setting criteria . . . 15-8
Setting For conditions . . . 15-9
Setting While conditions . . . 15-9
Counting rows . . . 15-9
Performing calculations on a selection of rows . . . 15-9
Viewing and editing special field types . . . 15-10
Viewing the contents of special field types . . . 15-11
Memo fields . . . 15-11
Binary fields . . . 15-11
Importing an image or sound into a binary field . . . 15-11
OLE fields . . . 15-12
Adding an OLE object to an OLE field . . . 15-12
Removing an OLE object from an OLE field . . . 15-12

Chapter 16
Setting up security 16-1
Setting up security strategies . . . 16-1
Individual login via automatic password dialogs . . . 16-2
Preset access via Database and Session objects . . . 16-2
Preset access for Standard table types . . . 16-2
Preset access for SQL and other table types . . . 16-3
Table-level security for DBF tables . . . 16-3
About groups and user access . . . 16-4
Table access . . . 16-4
User profiles and user access levels . . . 16-4
About privilege schemes . . . 16-4
Table privileges . . . 16-5
Field privileges . . . 16-5
About data encryption . . . 16-5
Planning your security system . . . 16-5
Planning user groups . . . 16-6
Planning user access levels . . . 16-6
Planning DBF table privileges . . . 16-6
Planning field privileges . . . 16-7
Setting up your DBF table security system . . . 16-7
Defining the database administrator password . . . 16-7
Creating user profiles . . . 16-8
Changing user profiles . . . 16-8
Deleting user profiles . . . 16-8
Establishing DBF table privileges . . . 16-8
Selecting a table . . . 16-9
Assigning the table to a group . . . 16-9
Setting DBF table privileges . . . 16-9
Setting field privileges . . . 16-9
Setting the security enforcement scheme . . . 16-10
Table-level security for DB tables . . . 16-10
Removing passwords from DB tables . . . 16-11

Chapter 17
Character sets and Language drivers 17-1
Determining the language displayed by the User Interface . . . 17-1
About character sets . . . 17-2
About language drivers . . . 17-3
Performing exact and inexact matches . . . 17-3
Using global language drivers . . . 17-4
To set the ldriver option in dBASE_SE.INI: . . . 17-4
Using table language drivers . . . 17-5
Identifying a table language driver and code page . . . 17-6
Non-English Character Display Issues . . . 17-6
Selecting Specialized Product Fonts . . . 17-6

Table language drivers versus global language drivers 17-7
Handling character incompatibilities in field names 17-7
Converting between OEM and ANSI Text 17-8
Converting from OEM to ANSI 17-8
Converting from ANSI to OEM 17-8
How to convert and view your source code 17-8

Chapter 18
Converting prior version dBASE Applications to dBASE SE 18-1
Converting a dBASE III+/IV Application to Visual dBASE 5.7 18-1
Installing Visual dBASE 5.7 18-1
Overview 18-2
Suggested Steps 18-2
Sample 18-2
Recreating Menus 18-2
Create a sample menu in Visual dBASE 5.7 18-2
Converting .VUE Files 18-3
Converting Forms 18-4
Using The Component Builder to Convert a Form from a .PRG 18-5
Fine-Tuning The Form 18-6
Notes about Memo and Logical fields 18-7
Running the Form 18-8
Using ACCEPT or INPUT? 18-8
Reports and Labels 18-10
Converting dBASE 5.0 for DOS Screens/Menus to dBASE SE 18-12
Setting up for Conversion 18-12
Converting screens or menus to dBASE SE: 18-13
Conversion considerations 18-13
An Option 18-14
Converting dBASE 5.0 for DOS Reports and Labels . . 18-14
Converting Visual dBASE 5.7 Applications to dBASE SE . 18-15
Converting Forms 18-15
Converting Reports and Labels 18-16
Converting QBE Files to Datamodules 18-16
Updating Forms to Use Datamodules 18-17

Tables

7.1 Standard controls 7-4
7.2 Data Access 7-6
7.3 Shading Properties in the Form Designer . . . 7-7
7.4 Components specific to reports 7-7
7.5 Method menu commands 7-12
8.1 Menubar and popup root properties, events and methods 8-8
8.2 Item properties, events and methods 8-8
8.3 Toolbar properties, events and methods 8-10
8.4 Toolbutton properties, events and methods . .8-10
10.1 Methods of controlling execution in the Debugger 10-5
11.1 Query operators 11-8
12.1 Values for the drilldown property 12-9
13.1 dBASE field types for level 7 tables 13-6
14.1 Data-entry constraints 14-8
14.2 Sample dBASE key expressions 14-15
15.1 Navigating rows using the menu, mouse or keyboard 15-4
15.2 Types of calculations 15-10
15.3 Field selection keyboard shortcuts 15-11
16.1 Setting user groups 16-6
16.2 Setting user access levels 16-6
16.3 Setting table privileges 16-7
16.4 Setting field privileges 16-7
16.5 Setting DBF table privileges 16-9
16.6 Setting field privileges 16-10
17.1 European language drivers available in *dBASE SE* 17-3
17.2 Automatic assignment of language drivers by *dBASE SE* 17-5
17.3 Language drivers: Table versus Global. . . . 17-7

Figures

3.1 The dBASE SE dQuery/Web start screen . . . 3-2
3.2 EasyStart Menu . . . 3-3
3.3 The Navigator . . . 3-3
3.4 Speedtip example . . . 3-4
3.5 Database Alias dialog box . . . 3-5
3.6 Select/Add dialog box . . . 3-5
3.7 Create Alias dialog box . . . 3-6
3.8 Create Folder dialog box . . . 3-6
3.9 Displayed dataModule . . . 3-7
3.10 Save dataModule dialog box . . . 3-7
3.11 Create Table dialog . . . 3-8
3.12 The Table Designer . . . 3-8
3.13 Table Designer . . . 3-9
3.14 dQuery/Web design surface . . . 3-9
3.15 Table Designer . . . 3-10
3.16 Set Index dialog box . . . 3-11
3.17 Edit Report dialog box . . . 3-18
3.18 Report Design surface . . . 3-18
3.19 One-Click Windows dialog box . . . 3-19
4.1 Sample event handler for a "Hello world" form . . . 4-3
5.1 Project Explorer displaying a view of a file . . . 5-3
5.2 Sample MDI window . . . 5-10
5.3 Saving a custom class . . . 5-12
5.4 Set Custom form Class Dialog Box . . . 5-13
5.5 Save as Custom dialog box; saving Custom Components . . . 5-14
6.1 Drag & Drop . . . 6-10
7.1 Form Designer with a wizard-created form . . . 7-2
7.2 Report Designer with a wizard-created report. 7-3
7.3 Field Palette . . . 7-8
7.4 Using the Inspector . . . 7-10
7.5 Events page of The Inspector . . . 7-11
7.6 Methods page of The Inspector . . . 7-12
7.7 Layout | Align commands . . . 7-15
7.8 Layout | Size commands . . . 7-16
7.9 Layout | Spacing commands . . . 7-16
7.10 Set Scheme dialog box . . . 7-17
7.11 Format toolbar . . . 7-17
9.1 Code Block Builder . . . 9-3
9.2 The Command window . . . 9-4
10.1 Source Window . . . 10-3
10.2 Debugger tool windows, docked . . . 10-5
10.3 Breakpoint window . . . 10-7
10.4 Breakpoint Condition dialog box . . . 10-7
10.5 Call Stack window . . . 10-9
10.6 Watch window . . . 10-9
11.1 SQL Designer Criteria Page . . . 11-5
11.2 SQL Designer: Group selection . . . 11-7
11.3 SQL Designer: Grouped . . . 11-7
11.4 SQL Designer: Drill-down . . . 11-8
12.1 Wizard-generated report on a GOODS table . 12-3
12.2 Wizard-generated Summary Report . . . 12-4
12.3 Adding grand total in the Report wizard . . . 12-4
12.4 Report in Design mode with Group view displayed . . . 12-5
12.5 Selecting a column to delete from a report . . . 12-6
12.6 Field Palette containing active fields . . . 12-7
12.7 Aggregate calculation on a Report . . . 12-10
13.1 Components of a Table . . . 13-2
13.2 One-to-many relationships . . . 13-4
15.1 Table-editing toolbar . . . 15-2
15.2 Columnar view . . . 15-3
15.3 Navigating rows using the toolbar . . . 15-4
15.4 Find Rows dialog box . . . 15-5
15.5 Replace Rows dialog box . . . 15-7
15.6 Delete Rows dialog box . . . 15-8
15.7 Count Rows dialog box . . . 15-9
15.8 Calculate Aggregates dialog box . . . 15-10
15.9 Calculation Results dialog box . . . 15-10
17.1 Setting LDRIVER in the dBASE_SE.INI . . . 17-5
18.1 Sample Menu . . . 18-3
18.2 Sample Form . . . 18-4
18.3 The Component Builder . . . 18-5
18.4 Component Builder window with displayed source code . . . 18-6
18.5 A Running Form . . . 18-8
18.6 Form with PushButton . . . 18-10
18.7 The dBASE IV for DOS Report designer . . . 18-11
18.8 Crystal Reports for dBASE . . . 18-12

Chapter

1

Introduction to *dBASE SE*

Welcome to *dBASE SE* !

dBASE SE is another milestone in the evolution of dBASE – one led, staffed and supported by users and developers like yourselves. With the advent of the dQuery/Web dataModule designer, combined with the legendary power of dBASE, dBASE SE represents the most state-of-the-art data manipulation tool on the market today. For Legacy users the message is simple: OOP is now, and making the jump just got a whole lot easier.

What is *dBASE SE*?

dBASE SE is a 32-bit rapid application development (RAD) environment for the creation of powerful database applications and data-driven web applications. It features flexible interactive database administration tools, an advanced third-generation object-oriented programming model, and a high level of backward compatibility.

Its rich assortment of powerful Windows tools, including Table, Form, Menu and Report designers makes modeling, managing, retrieving and reporting information easier and faster than ever before. dQuery/Web, *dBASE SE*'s radical new interactive live-data tool makes it extraordinarily easy to visualize, enter, edit and retrieve information, regardless of its source. The Borland Database Engine (BDE) included with your package allows easy connectivity to dBASE tables—including the new DBF7 file format—and provides native support for Paradox, Microsoft Access, and Microsoft FoxPro formats, as well as any 32-bit ODBC-supported data source. A set of high performance SQL Links drivers extend support to the most popular enterprise database formats, including Oracle, Sybase, InterBase, MS SQL Server, IBM DB2, and Informix. *dBASE SE* also allows you to create links to other data sources through custom data objects.

With *dBASE SE* you can:

- Work in SQL Server data and save it as Informix
- Work in DB2 and save it as dBASE
- Import data from other applications
- Run reports against almost any database
- Automatically generate applications that work with multiple sources simultaneously.

All this is possible because *dBASE SE* is totally object-oriented. Information is treated as fully inheritable, reusable objects, not as separate, incompatible, difficult-to-convert databases and tables. Want to link a form or a report to your data? Just drop a data object on the appropriate designer and *dBASE SE* handles the rest.

The expert developer will love *dBASE SE*'s object-oriented dBL programming language. Sporting full inheritance for an incredible level of reusability, *dBASE SE* also provides the first drag-and-drop distributed object model with full inheritance. Never has it been easier to update and upgrade. Never has it been more efficient to provide remote technical support.

dBASE SE is also a great second tool for developers working in other languages, environments and databases. From its ad-hoc data-query tools to its built-in Report Classes, *dBASE SE* provides the functionality missing from other, popular, single-purpose tools. Writing an application in Delphi or Visual Basic? Need to see the results immediately? Just fire up *dBASE SE* and browse the data in real-time. Need to kick out a report in minutes? dQuery/Web's No-Click reports require virtually no work at all. Need to model your data, view relationships, check out the results of a SQL Query? Just a few mouse clicks and you've got a real-time result.

This section introduces you to the *dBASE SE* development environment and provides examples and tips that will help you get started quickly. It also describes features introduced in versions prior to *dBASE SE* as well as those new to this release.

dBASE Newsgroups

The dBASE Newsgroups, located at news://news.dBase.com, are a place where dBASE users and developers can obtain peer support and exchange information, tips and techniques. We encourage members of the dBASE community to assist each other with technical questions. Please read the Newsgroup Guidelines before participating.

For more information about Newsgroup Guidelines and configuring your newsreader, visit the dBASE website at www.dBase.com.

The *dBASE SE* Knowledgebase:

Your *dBASE SE* package also contains a full copy of the new Knowledgebase in HTML format. To use the Knowledgebase, double-click on the file "kbmenu.htm" in the \KB folder on your installation CD.

Topics and information include:

- Newsgroups Support
- FAQs
- Programming how-to articles
- The dBASE User's Function Library Project files (dUFLP)
- A complete list of changes and bug-fixes

The *dBASE SE* Knowledgebase is also available on the dBASE Inc. website; **http://www.dbase.com**. The Knowledgebase is an ever expanding repository of all things dBASE. Check our website frequently for updates!

Note This site can also be accessed through the *dBASE SE* Help menu.

Overview of dBASE SE

dBASE SE provides dozens of new features and language elements to provide you with a more productive, efficient work environment. Among these enhancements:

- Report objects and the integrated Report designer
- Project Explorer
- Data objects
- ActiveX integration
- Visual designers
- The Inspector
- Full-featured Source editor
- SQL designer
- BDE utility and database support
- DBF7 file format
- Dynamic External Objects (DEO)
- Source Aliasing

Report objects and the integrated Report designer

You can create reports and labels using native Report objects and the Report designer (similar to the Form designer). The classes use the full power of the object-oriented programming model, with objects that offer such sophisticated features as:

- Complex expression support
- Conditional rendering
- Flexible grouping
- Inheritance
- Report viewing within a form

Project Explorer

A few of the features The Project Explorer provides include:

- Automatic file viewers
- Instant switching between visual preview and source code views
- Project-based compiler directives

Note that the Project Explorer replaces the Catalog functionality available in earlier versions. A conversion utility, CAT2PRJ.PRG, is available in your \Bin directory to let you easily convert Catalogs to Projects.

Data objects

Data access classes merge the SQL and object-oriented paradigms. You can use queries within databases within sessions. Sessions provide independent connections to tables. Each database can then connect to a different data source. The queries connect to one or more tables and provide table navigation capability. The objects let you

- Use Query, Rowset, Field, Database, Session, StoredProc, and other classes to access tables and stored procedures. (Data objects use only SQL to gather data.)
- Use DataModule and DataModRef objects to represent multiple data objects and their relationships. These objects take the place of form.view and old QBE files.
- Create custom data objects to access specialized, third party, and future data formats.

Visual designers

The visual designers feature numerous usability and productivity enhancements, including

- Win32 controls
- Grid and Browse controls, which are faster and offer more functionality than the table browsing functionality in earlier versions
- Complete ActiveX control integration
- Live Two-Way-Tool[a] editing: changes made in visual designers are immediately reflected in the source code and vice versa; to switch between the two, press F12
- File drag-and-drop capability lets you easily link tables and pull files onto a form or report from the Navigator, Project Explorer, Windows Explorer, or even the Windows Find results window
- Dockable toolbars
- A text-formatting palette, featuring industry-standard HTML tags
- In-place editing for Text components
- Automatic labeling for fields dragged from the Field palette
- Versatile form/report metrics (chars, twips, pixels, millimeters and more)
- Expanded image format support. The list now includes support for BMP, GIF (including animated GIF), ICO, JPEG, PNG, XBM, WMF, EMF, TIFF, PCX, and EPS formats

ActiveX integration

You may add ActiveX (OCX) controls directly into your forms and reports. You can either inspect ActiveX controls directly, or right-click a control to access its internal configuration dialog.

The Inspector

A few of the features offered by The Inspector include:

- Single-click list expansion
- Advancement or toggling of a selection with Ctrl-Enter (an alternative to double-clicking)
- A tool button on the Methods page to let you write method overrides
- Bold highlighting on changed and non-default values
- A property type chooser and history list
- Access to a codeblock builder and long string editor tool

Full-featured Source editor

A fully-customizable ASCII Source editor is available for writing programs and methods. It features

- Instant access (F12) from visual designers, with immediate updating between source and design modes
- A tree view of classes, objects, and methods
- Tabbed pages
- Syntax highlighting and a customizable color scheme
- Drag-and-drop editing, including the ability to drag code snippets onto the desktop or into another editor page, and drag them back into any page
- Multiple-level grouped undo
- Keystroke macro recording and playback
- An automatic file-open feature: If another file name appears in your source code, you can set your cursor on the name and press Ctrl-Enter to open the file
- Easy commenting and comment removal on selected blocks

SQL designer

The SQL designer lets you create, edit, and execute SQL queries. You can use the tool to test and apply the simplest SELECT statements to the most advanced queries on data in any supported data source. You can then view the results and save your query for inclusion in your programs.

BDE Administrator and database support

The BDE Administrator utility helps you create aliases and test and configure your database connections and settings. The BDE engine also offers a significant number of database objects that can be opened in each BDE session.

The SQL Links high-performance drivers for *dBASE SE* support most popular enterprise database formats including Oracle, Sybase, InterBase, MS SQL Server, IBM DB2 and Informix.

dBASE SE includes native (non-ODBC) Microsoft Access and Microsoft FoxPro table support, along with Local SQL (for DBF and DB tables) and support for the DBF7 table format.

DBF7 file format features

A few of the features included are:

- Long field names
- New field types: TimeStamp, Double, AutoIncrement, Long
- Field constraints: minimum, maximum, required, and default
- Null character fields
- Distinct indexes; user gets a key violation when attempting to add a duplicate
- Primary distinct index
- Referential integrity
- Table constraints: an array of strings containing logical *dBASE SE* expressions that act as row-level constraints when attempting to save a row
- Custom field attributes that comprise an active data dictionary that works at runtime as well as design time. These attributes are named properties with string values and are created in the Table designer.

Samples and sample viewer

Sample forms, reports, menus and other files are located in the SAMPLES directory of your main *dBASE SE* directory.

The directory features a form called SAMPLE GUIDE.WFM, which gives you a visual preview and description of the purpose of each sample form or applet.

Source Aliasing

What is Source Aliasing?

Source Aliasing is a new feature in *dBASE SE* that provides true source-code portability by referencing files indirectly - through an Alias. Just as the BDE allows you to define an Alias to represent a database or a collection of tables, Source Aliases let you define locations for your various files without using explicit paths - which often differ from machine to machine.

Note Source Aliasing works only in the *dBASE SE* design environment or when running programs from within the *dBASE SE* shell. It is not a runtime feature. To access files indirectly in deployed applications, use Dynamic External Objects (DEO) instead of Source Aliasing.

Dynamic External Objects (DEO)

dBASE SE features a brand new external object model that, if used consistently, promises the lowest total-cost-of-ownership in the industry.

DEO is a unique technology that allows not just users, but applications, to share classes across a network (and soon, across the Web). Instead of linking your forms, programs, classes and reports into a single executable that has to be manually installed on each workstation, you deploy a shell - a simple *dBASE SE* executable that calls an initial form, or provides a starting menu from which you can access your forms and other *dBASE SE* objects.

Dynamic Objects can be visual, or they can be classes containing just "business rules", that process and post transactions, or save and retrieve data. Each of these objects may be shared across your network by all users, and all applications that call them.

Tips You'll have to experiment with DEO to discover the best approach for the way you write and deploy applications. However, here are some interesting subtleties you might leverage to your benefit:

Unanticipated updates: Assume you already shipped a *dBASE SE* application as a full-blown executable. Now you want to make a change to one module. No problem, just copy the object file to the home directory of the application and it'll be used instead of the one built in to the executable. You don't need to redeploy the full application the way you do in most other application development products. Just the changed object.

Reports: You can deploy reports or even let your users create reports (using dQuery/Web) and add them to their applications by designing a report menu that checks the disk for files with an .reo extension. Let the menu build itself from the file list. Here we have true dynamic objects - the application doesn't even know they exist until runtime. DEO supports real-time dynamic applications.

Dynamic Update Support: Want to try out some code or deploy a fix to a customer site or a remote branch office? No problem, just FTP the object file to the remote server and the update is complete.

Remote Applications: If you have VPN support (or any method of mapping an Internet connection to a drive letter), you can run *dBASE SE* DEO applications remotely over the Internet. A future version of *dBASE SE* will include resolution of URLs and IP addresses so you can access remote objects directly through TCP/IP without middleware support.

Distributed Objects: Objects can be in a single folder on your server, in various folders around your network, or duplicated in up to ten folders for fail-over. If one of your servers is down, and an object is unavailable, *dBASE SE* will search the next locations on the list until it finds one it can load. Objects can be located anywhere they can be found by the workstation.

dBASE SE documentation

Your *dBASE SE* Help system offers full context sensitive help, examples, expanded and updated conceptual and training material, plus a full *Language Reference* with code samples you can cut and paste directly from the Help window.

Typographical conventions

The following typographical conventions used in this Help system will help you distinguish among various language and syntax elements.

Convention	Applies to	Examples
Italic/Camel cap	Property names, events, methods, arguments	*length* property, *lockRow*( ) method, <*start expN*> argument
ALL CAPS	Legacy dBASE commands and other language elements from previous versions. Also used in file and directory references.	APPEND BLANK, CUSTOMER.DBF
Roman/Initial cap/ Camel cap	Class names (including legacy classes), table names, field names, menu commands	class File, class OleAutoClient, Members table, Price field
Monospaced font	Code examples	a = new Array(5, 6)

Documentation updates and additional information resources

The dBASE Inc. home page on the World Wide Web, at **http://www.dbase.com**, helps you find the most current information about *dBASE SE*. Periodic updates to the *dBASE SE* Help system, as well as technical notes, tips, and other materials that will further your understanding of the program, will be posted on the dBASE Inc. website.

Your *dBASE SE* CD also contains a full copy of the new Knowledgebase in HTML format. To use the Knowledgebase, double-click on the file "kbmenu.htm" in the \KB folder on your installation CD. The *dBASE SE* Knowledgebase is also available on the dBASE Inc. website, and through the Help menu. The Knowledgebase is an ever expanding repository of all things dBASE. Check our website frequently for updates!

The BDE Administrator, and other included applications and controls, offer their own Help systems. They can be run from disk, from within the applications, or by pressing F1 while an application is open or the control is selected.

For tips on using Windows Help, choose Help | How To Use Help from the main *dBASE SE* menu.

Software registration

To register your product with dBASE Inc. and qualify for support, use the registration card included in your *dBASE SE* package or register at our Web site: **http://www.dbase.com**

dBASE Inc. offers developers high-quality support options. These include free services on the Internet, where you can search our extensive information base and connect with other users of dBASE products. In addition to this basic level of support, you can choose from several categories of telephone support, ranging from support on installation of your dBASE product to fee-based consultant-level support and detailed assistance. To obtain pricing information for dBASE technical support services, please visit our Web site at **http://www.dbase.com**.

To request assistance, call:

dBASE Call Center	Toll free (USA and Canada only)
607-729-0960	888-dBASE-32
	(888-322-7332)

The call center is open from 9:00 AM to 5:00 PM eastern time USA.

Chapter

2

Installing *dBASE SE* and connecting to an SQL database server

This chapter tells you what you need to run *dBASE SE* and lists the products and programs in you *dBASE SE* package. Then it shows

- How to install and uninstall *dBASE SE*
- What happens during installation
- How to connect to an SQL server

What you need to run *dBASE SE*

To run *dBASE SE* you need the following:

HARDWARE

All of the following are required

- Intel 486DX2 or higher
- CD-ROM drive
- 16MB RAM
- 35 MB hard disk space
- VGA or higher resolution (SVGA recommended)
- Microsoft mouse or compatible pointing device

OPERATING SYSTEM

- Microsoft Windows® 95/98/2000
- Microsoft Windows® NT 4.0
- Microsoft Windows® ME
- Microsoft Windows® XP

NETWORKS

- It runs on all Windows-compatible networks, including NT networks, Novell networks and peer-to-peer networks, such as Lantastic and Netbeui.

Products and programs in your *dBASE SE* package

These are the products and programs that come with *dBASE SE*:

- *dBASE SE*

- dQuery dataModule designer.
- The *dBASE SE* debugger.
- The 32-bit Borland Database Engine (BDE) and configuration utility (BDE Administrator), with native drivers for dBASE, Paradox, Microsoft Access 95/97, and Microsoft FoxPro databases.
- Integrated Help system, including a full *Language Reference*.
- The dBASE Knowledgebase.
- Sample tables, forms, reports, and other files you can learn from, use or adapt.
- A selection of custom controls and graphics (backgrounds, cursors, and other images) for use in forms and reports.
- A utility for converting Crystal reports to *dBASE SE* reports.
- A utility for converting Visual dBASE 5.x forms to *dBASE SE* forms.
- A dBASE 5.0 for DOS Form/Menu Converter to assist with the conversion of object-based forms and menus created in dBASE 5.0 for DOS.
- ODBC connectivity and Local InterBase.
- A full set of high performance SQL Links drivers for connecting to: Oracle, Sybase, InterBase, MS SQL Server, IBM DB2 and Informix databases.

Installing *dBASE SE*

The target for the installation of *dBASE SE* will default to a folder where *dBASE SE* may already be installed. You can install *dBASE SE* to a folder location of your own choosing, however installing to the same folder maintains your current dBASE_SE.INI settings. If *dBASE SE* is not presently installed, the target folder will be c:\Program Files\dBASE\SE.

If the BDE is already installed on your machine, the existing folder location will be selected by default. It is recommended that you continue using this location. Current BDE settings and any new BDE settings are merged during the install, so you don’t lose any prior BDE configuration.

To install *dBASE SE*,

1. Insert your *dBASE SE* CD into your CD-ROM drive.

   If Autorun fails to start:

   1. Choose Run from the Start menu.
   2. In the Run box, type the letter of your CD ROM drive followed by a colon and the word setup. For example,

      ```
      D:\dBASE_SE\setup
      ```

2. Follow the directions that appear onscreen.

At the end of the installation process, you are given the option to read the "readme" file. You can read it then or open it later from your *dBASE SE* root directory or program group.

What happens during installation

In addition to installing the options you selected, the following occurs during setup:

- dBASE System-Level files (.DLL, .OCX, etc.) are installed and registered as required in the Windows, or Windows\System, directory.
- The dBASE registry settings are written to HKEY_LOCAL_MACHINE\SOFTWARE\dBASE\SE.
- In addition to the usual subdirectories, like BIN, a subdirectory named My Projects is created off the *dBASE SE* home directory, by default C:\Program Files\dBASE\SE\My Projects.
- The Language of the installer will attempt to match the Windows system language setting. This can be set via the Control Panel | Regional Setting. During the install process, you are given the option of selecting

additional languages. For example, if you select English and German, the User Interface resources and documentation (as available) will be installed for both languages

- The User Interface language resources installed for the BDE Administrator, the BDE Online Help, and the User Interface resources for the Project Explorer, will match the language of the Installer itself. Multi-language installs are not supported for these components.

Un-installing *dBASE SE*

To un-install *dBASE SE*, use the Add/Remove Programs dialog box in the Windows Control Panel.

Note During un-installation, you also have the option of keeping any shared program libraries on your disk that may be needed by other programs. Even if you choose to remove the shared files, other files and directories may remain on your disk after un-installation. These remaining files are usually forms, applications, directories or other items you created while using *dBASE SE*.

For other issues that may affect *dBASE SE* removal, see README.TXT (normally stored in your *dBASE SE* root directory or program group).

How to connect to an SQL database server

If you are connecting your *dBASE SE* application to an SQL database, you need to configure your SQL Links Driver and the BDE to access your SQL database. In this procedure, you create an alias that BDE uses to locate the SQL database. You then add this alias to the Database object on your *dBASE SE* form or report.

Note The following instructions apply to users of the high performance SQL Links drivers for *dBASE SE*.

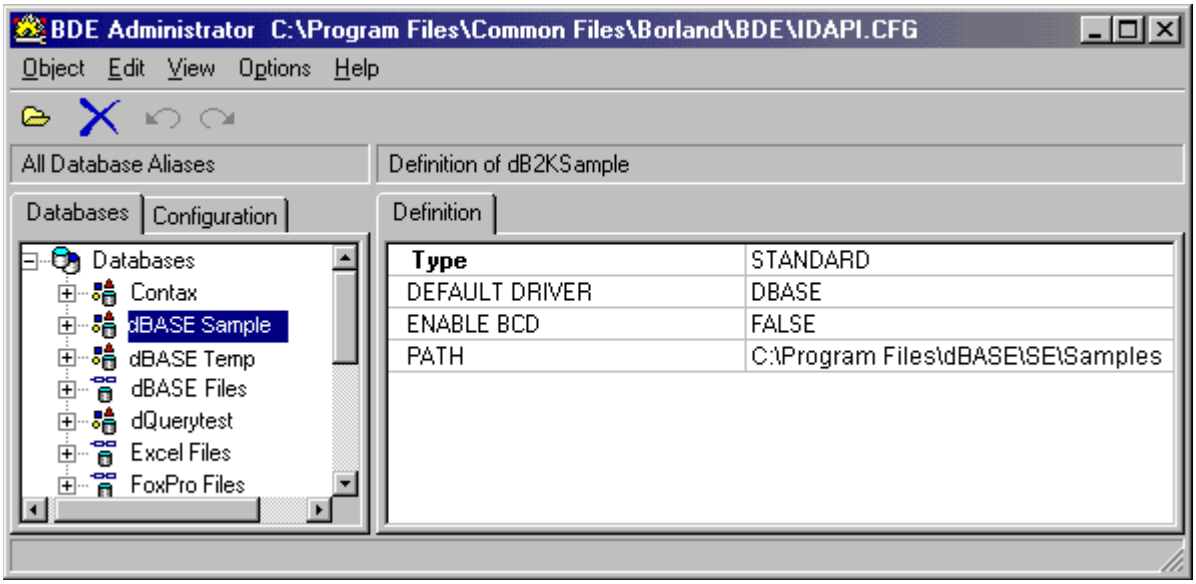
Install and configure the server software

Consult the documentation for your SQL database management system product for specific guidance on the initial steps of the following general procedure (specific product requirements may differ).

1. Make sure you have properly installed the client software for the database management system product to which you want to connect (Oracle, Sybase, InterBase, MS SQL Server, IBM DB2 or Informix).
2. Define server names or other connection strings in the product's required configuration files. For example, in Oracle, TNSNAMES.ORA, or in Sybase, SQL.INI, and so on.
3. Test the connection by using the database vendor's connection utility (such as Sybase's SYBPING.EXE). If you cannot "ping" the server with this utility, BDE and *dBASE SE* will probably not be able to access it either.
4. Make sure both the BDE and the SQL Links drivers are properly installed. If properly installed, the SQL Links drivers for Oracle, Sybase, Interbase, MS SQL Server, IBM DB2 and Informix appear on the Configuration page of the BDE Administrator (available from the *dBASE SE* program group off the Start menu).

Configure the Borland Database Engine (BDE)

The Borland Database Engine (formerly called IDAPI) allows *dBASE SE* to share data with supported SQL databases, Access 95/97, and FoxPro. If you'll be connecting to any of these databases, you must assign them aliases and otherwise configure the BDE with the parameters of the database.



To create an alias and configure the BDE,

1. Open the BDE Administrator (BDEADMIN.EXE), available from your *dBASE SE* program group.
2. Click the Databases tab.
3. Right-click and choose New to create a new alias
4. Enter the full path to the database, including the file name when appropriate.
5. Click the Configuration tab and set the appropriate parameters in the Definition panel:
   - Parameters may vary according to vendor.
   - Parameters in bold cannot be changed.
   - To accommodate record locking in a Windows NT server environment, it is necessary to set the BDE's *localShare* parameter to "true".

Note If you're creating a new ODBC alias, you must define its DSN before you can connect to that database.

You'll find complete instructions in the BDE Administrator Help system. Press F1 with the cursor in any parameter for information on that parameter.

Listing SQL tables in the Navigator

To see SQL tables listed in the Navigator,

1. Open the Navigator, and click the Tables tab.
2. Select the SQL server database alias from the Look In drop-down list (at the top of the Navigator).
3. At the prompt, enter your login name and password to connect to that SQL server database.
4. Once you're connected, you will see the tables in that database in the Navigator.

Chapter

3

dBASE dQuery/Web

What is dQuery/Web?

dQuery/Web is the data center of *dBASE SE*. It's a drag-and-drop tool that provides easy, intuitive data-modeling, filtering, data-entry, reporting and automatic application generation. It's both an interactive tool and a development tool. Want to enter, retrieve, or edit data? Drag tables to the design surface. Want to create a persistent data relationship including multiple tables, joins, filters and relationships? Click Save and dQuery/Web creates a custom, reusable dataModule Class. Want to work both visually and in code? Like all other tools in *dBASE SE*, dQuery/Web is a Round-Trip Tool. This means you can write code, drag-and-drop, or switch back and forth at will.

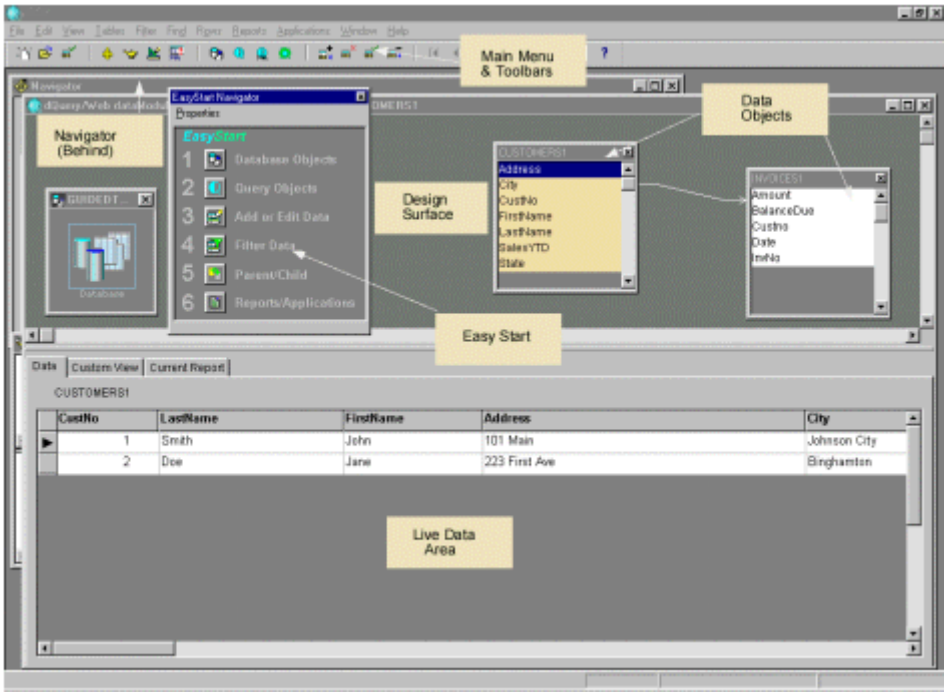
Basic functionality of dQuery/Web:

- Create and manage tables
- Data entry, edit and delete
- Query data
- Data Pump from different formats
- Parent-child relationships
- Filter data
- Search data
- Summarize Data
- Global Search and Replace
- Custom Views of Data
- No-Click Reports
- Customize Reports
- Generate One-Click Windows apps

This list, like most others, doesn't really do justice to dQuery/Web. Take "Filter Data" for example. There are at least six different ways to filter data in dQuery/Web - some persistent, some temporary - for use while doing calculations or exporting data to other formats and tools. In fact, dQuery/Web is particularly notable for the many ways it offers to perform almost any operation. We'll point some of these out as we work our way through this chapter.

Double-Click the *dBASE SE* icon to start *dBASE SE*. Figure 3.1 shows the *dBASE SE* dQuery/Web screen on startup.

Figure 3.1 The *dBASE SE* dQuery/Web start screen



dQuery Components and Navigation

The ***Design Surface*** at the top of the dQuery/Screen is used for modeling data – for creating Query, Database, Session and Stored Procedure objects that describe your databases, tables and queries and the relationships between them.

The Live Data Area

The First Tab of the ***Live Data Area*** is used for entering, editing, and deleting data. The data in this section of dQuery/Web changes in real-time to reflect the currently selected Query object, and any filters or parent-child relationships you've set up for them.

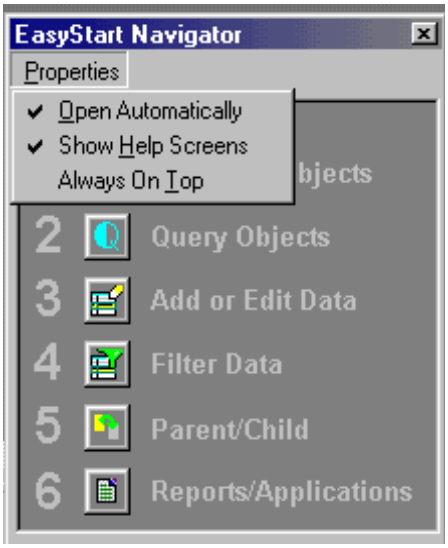
The Second Tab of the Live Data Area is the ***Custom View tab***. dQuery/Web allows you to drag-and-drop fields from any combination of Query objects to create a new, combined View of your data.

The Third Tab is the ***Current Report tab***, which displays either an automatic No-Click report, or any other report you choose to associate with this dataModule.

The EasyStart Menu

EasyStart is a dedicated menu designed to assist you in becoming familiar with dQuery/Web. The six steps go from a Database object, to a complete, fully-functional, Windows application.

Figure 3.2 EasyStart Menu



When installed, EasyStart is “Always on top”, which may become a bit annoying when you’ve got lots of Query objects displayed on the screen.

To change the behavior of EasyStart, click on the “Properties” menu at the top of the EasyStart window. You can change it to appear only when wanted, as well as turn its help screens off and on.

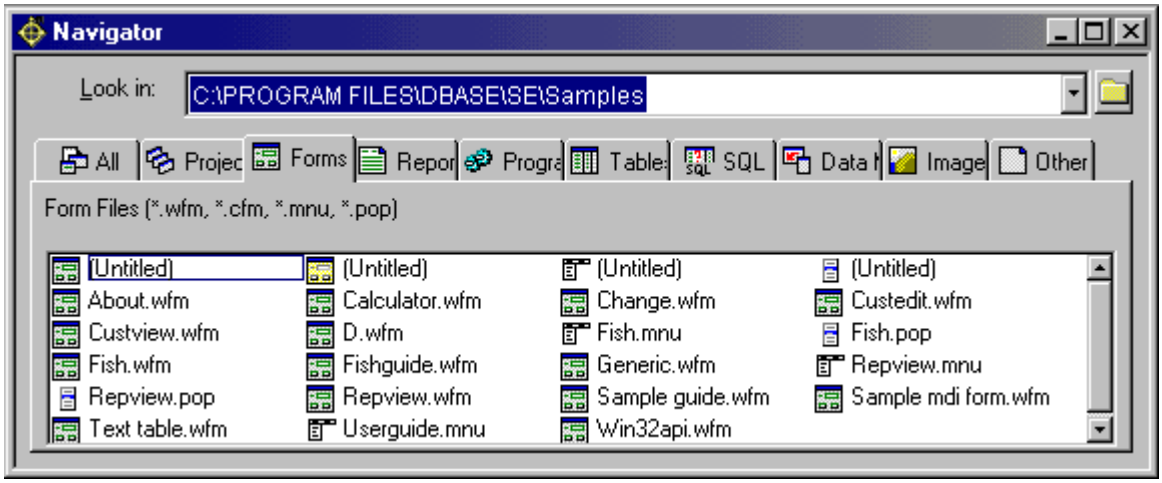
You can turn EasyStart back on at any time by right-clicking anywhere on dQuery/Web and selecting “EasyStart” from the popup menu.

The Navigator

There are many ways of getting around *dBASE SE*. The easiest is drag-and-drop - using your mouse to grab a component or file and drop it on the Design Surface - which is used when you’re working with existing data. When you’re creating new tables from scratch, dQuery/Web provides a set of dialog boxes and tools that help you through each process.

Take a minute to familiarize yourself with the Navigator. This is an extremely useful tool for opening files (including dataModules), and serves as the primary source for dragging existing tables and other files to dQuery/Web.

Figure 3.3 The Navigator



The Navigator can be called up from the “View” menu on most *dBASE SE* screens or by right-clicking on many components and tools throughout the program. If you click on “Untitled”, *dBASE SE* will bring up the appropriate tool and get you started on a new form, report, program or dataModule.

Menus and Toolbars

There are two kinds of menus supported in Windows. Main menus appear at the top of each screen. Pop-up or Context menus are brought up by right-clicking on a component. These menus supply only the options appropriate to the item you clicked on. For example, right-clicking on the Design Surface allows you to add new

components. Right click on the Live Data Area, and you get options to navigate, add, save and delete rows of data. Almost every option that appears on a right-click menu also appears on the Main Menu or the Toolbar. The advantage of right-click menus is that they save you a lot of navigation across the expanse of dQuery/Web and you don't have to look through every single option in the Main menu to find the one you want.

dQuery/Web always offers multiple ways of accomplishing a task. You want to add a new Query data object to your dataModule? Here are some of the possibilities:

- Right-click on the Design Surface
- Drag a table from the Navigator
- Drag a .SQL file from the Navigator
- Double click on "Untitled" on the Navigator Tables Tab
- File | New | Table
- File | New | Query Object from Table
- File | New | Query Object from SQL File
- EasyStart | Query Objects
- Click on a Query Toolbar Button

Opening, Saving and Creating a New dataModule

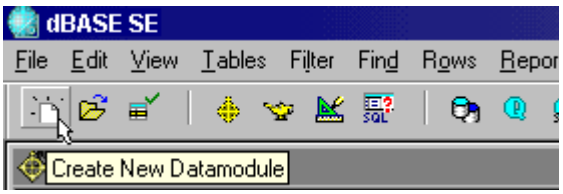
Every time you open dQuery/Web, you've got a brand new dataModule to work with. Alternatively, you can double-click on "Untitled" in the dataModule tab of the Navigator, or select File | New | Datamodule from the dQuery/Web Main Menu.

To open an existing dataModule - select File | Open. Select the desired dataModule and click Open, or Double-click on the desired dataModule in the DataModule tab of the Navigator.

A dataModule can be saved by - Select File | Save, File | Save As, or Click the Save toolbar button.

Tip All toolbars have a "Speedtip" that defines the operation they perform. Leave your mouse over a toolbar button for approximately one half second and the tip will appear.

Figure 3.4 Speedtip example



Creating a Database object

If EasyStart is not currently displayed on your screen, right-click on the Design Surface and then click on EasyStart.

dBASE SE uses Database objects to represent your database – your collection of tables that will define the information you wish to get from your system. In SQL databases, a database is a file. In other database engines, such as dBASE, Paradox, Fox and Advantage, the database is a folder containing related tables of data. dQuery/Web uses the Database object to link your data to applications, reports and dQuery/Web itself. Though not strictly required for non-SQL tables, we highly recommend that you always start with a Database object.

Aliases

The Database object is based on an alias – a specialized Path statement that tells *dBASE SE* where to find your data. Aliases offer many advantages. Their primary benefit is portability. Each machine on the network can use a different drive letter to find shared data. Using an alias allows you to point to the same location regardless of the path required to get there. Another advantage of aliases is they allow you to move your data from one place to

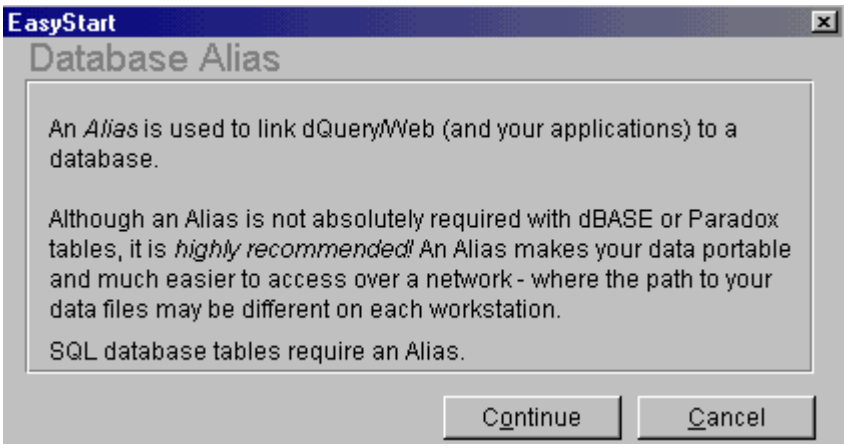
another without having to rebuild your application. If data is moved from drive H to drive L, change your alias to point to the new drive.

Creating a Database object

To create a Database object in dQuery/Web, select an existing alias, or create a new one, and dQuery/Web will generate a Database object automatically. Although we can choose aliases from many other databases such as Paradox, Fox, ODBC, Interbase, SQL Server, Infomix, Oracle, Sybase, DB2, and Access, select one from dBASE.

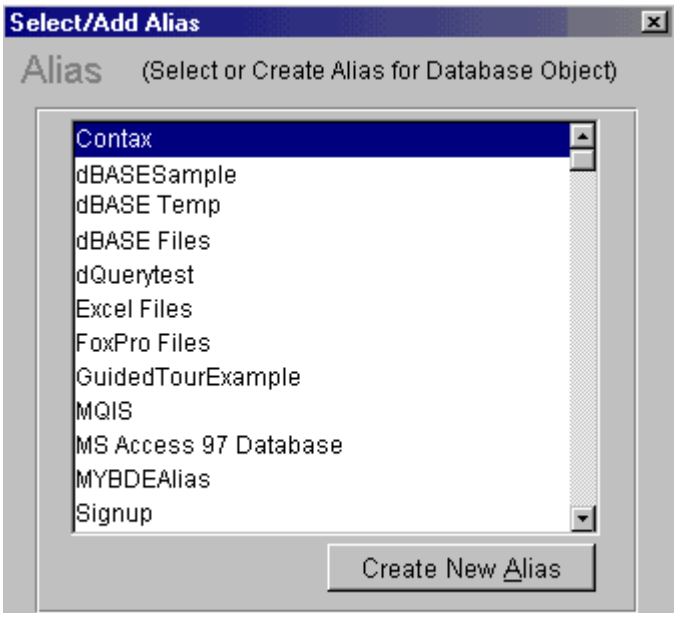
Begin by clicking on the “Database Objects” listed as number 1 on the EasyStart menu. A help dialogue like that shown in the picture below will appear. These informative help dialogues will appear when using various tools in dQuery/Web.

Figure 3.5 Database Alias dialog box



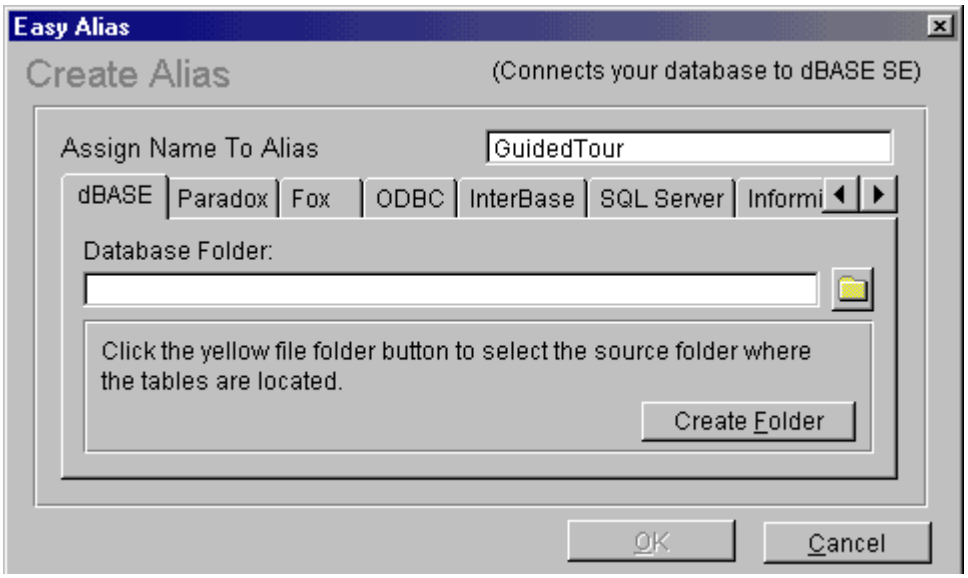
Click "Continue" and the Select/Add Alias dialog box appears . . .

Figure 3.6 Select/Add dialog box



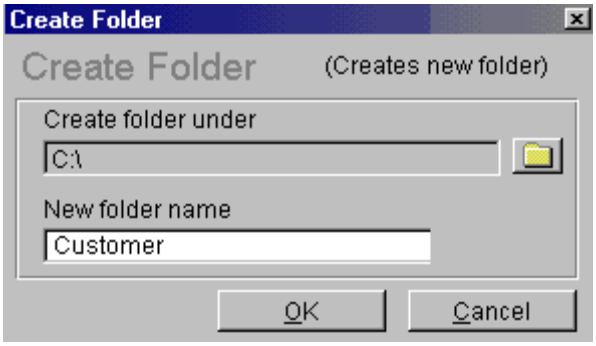
If the database you wish to connect to is not on the Alias list, click the create New Alias button. The Create Alias dialog box will appear.

Figure 3.7 Create Alias dialog box



In the first field enter the name of the Alias you wish to create. Then click “Create Folder”.

Figure 3.8 Create Folder dialog box



Some database engines use a folder to identify the location of your data. Others, such as SQL server, Informix, Oracle, Sybase, and Interbase access your tables through a single file or connection. dQuery/Web will ask for the appropriate information depending upon which database engine you select.

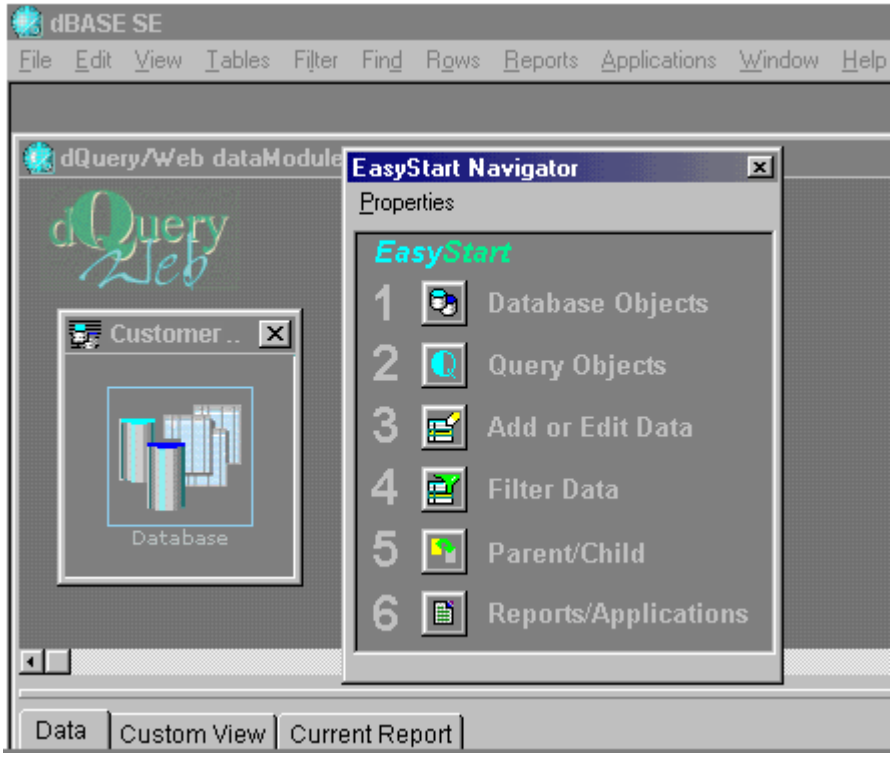
Click on the button with the yellow folder icon to locate the “parent” folder of the one you’re going to create (usually C:\).

Enter “Customer” as the new folder name and click OK. The “Create Alias” dialogue will appear again, displaying the folder you just created. Click OK.

This will bring you back to the “Select/Add Alias” dialogue. Your new alias should be highlighted. To continue, double-click on your alias or click OK.

The new Database object will appear, automatically, on the Design Surface of dQuery/Web.

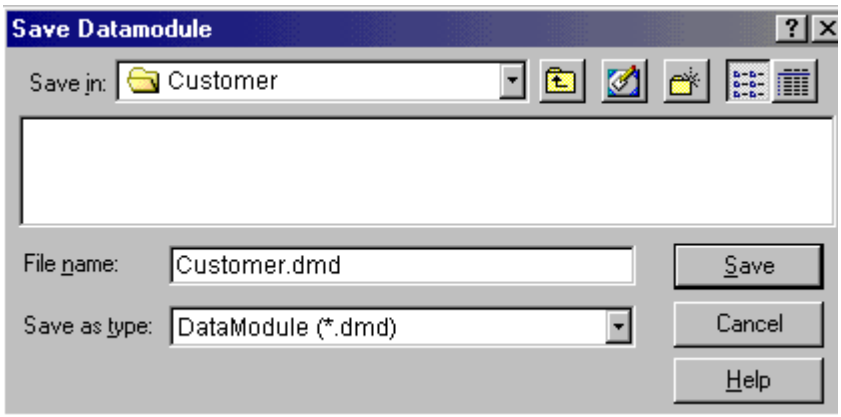
Figure 3.9 Displayed dataModule



Saving a Database object

It is always advisable to save your work as you go along (by clicking File | Save or the Save toolbar button). However, the first time you save, you'll have to tell dQuery/Web where you want to save this dataModule and what you want to call it.

Figure 3.10 Save dataModule dialog



Click File/Save from the Main menu.

Click the "Save in" button to locate your Customer folder. Double click the folder to open it.

Enter "Customer" into the file name field.

Click "Save".

Creating Query objects

A Query object is a representation of your data. It may represent all the rows and columns in a single table; a combination of rows and fields from multiple tables (a "Join") or a subset of rows from a table or combination of tables.

The following example uses Query objects to represent all the columns and rows of the Customer and Invoice tables. A Query is not a table, since it doesn't necessarily reflect everything that's stored in the table. It is an object that represents the selected rows only.

You don't have any tables or data yet. dQuery/Web will generate Query Objects automatically when you create the Customer and Invoices tables in the following steps.

Tip Query objects use SQL Statements to define subsets and Joins. When you select "all" rows and columns, dQuery/Web generates the following statement:

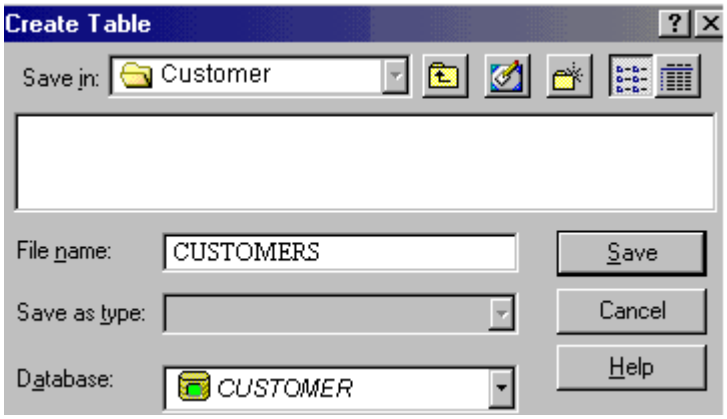
```
Select * from <Tablename>
```

The *dBASE SE* SQL Query Designer as well as the dQuery/Web SQL Statement dialog let you easily and quickly design and implement much more complex SQL statements.

Bring up EasyStart and click on "Query objects" (option 2). The Add Query dialogue box will now appear with the option "Create Query from new table" selected (figure not shown).

Click Continue to bring up the Create Table dialog.

Figure 3.11 Create Table dialog

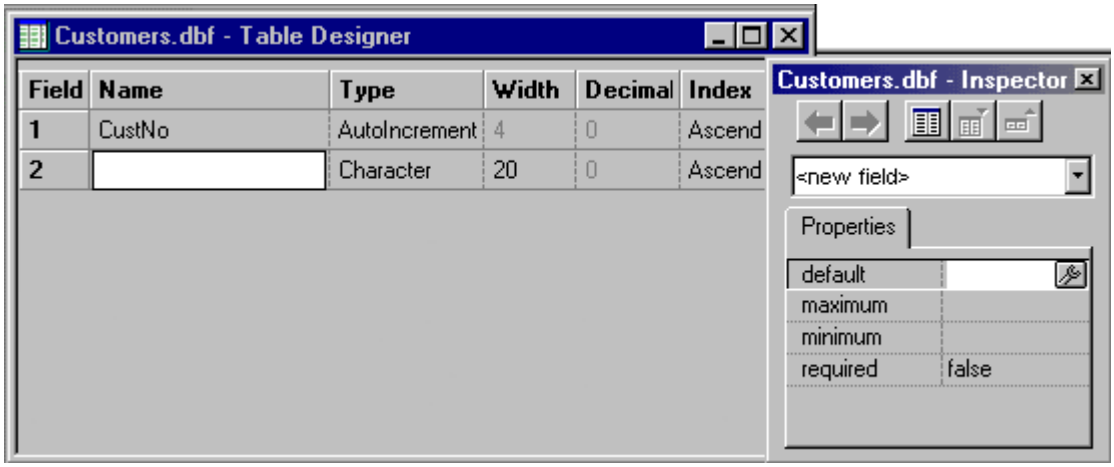


Enter "CUSTOMERS" into the File name field. Note that the dialog already came up with your current database (Customer) selected. Ignore the "Save In" at the top. Whenever a database is selected, the "Save in" is ignored and tables are stored wherever the Database object is set up to look for them.

Click Save to continue. Two new windows will appear: the "Table Designer" and the "Inspector" (If the Inspector does not appear, toggle the Inspector ON and OFF using the F11 key).

Tip You may have any number of Database objects open simultaneously. If you intend to use dissimilar tables (such as combining Access and Oracle tables), you'll need a Database object to represent each database of dissimilar type.

Figure 3.12 The Table Designer



Close the Inspector if it is open, since it will not be used at this time.

In the Table Designer, enter CustNo for the first Field Name. The Field Type will be Autoincrement, which automatically generates the next number in sequence whenever a new record is added.

To select the field type, arrow down and select AutoIncrement. The width will be set to 4. Autoincrements are actually 10 digits long, thought they're represented externally by a width of four positions.

The CustNo field will also have an index to provide fast lookups and allow you to change the search and display order of your query.

Note dBASE, Paradox, Foxpro, and Advantage tables allow you to explicitly select an index. Doing so can improve performance by 1000% or more. SQL database engines such as Oracle, SQL server, and Informix also use indexes, although you may not select them explicitly. The engine itself determines whether a helpful index exists, and then selects it automatically. Therefore it makes sense to add an index on any field on which you expect to search or filter, regardless of database engine type.

Click on the Index column and select “Ascend”, to indicate the order (ascending) in which you will want to see this data.

Figure 3.13 Table Designer

Customers.dbf - Table Designer

Field	Name	Type	Width	Decimal	Index
1	CustNo	AutoIncrement	4	0	Ascend
2	LastName	Character	20	0	Ascend
3	FirstName	Character	15	0	None
4	Address	Character	35	0	None
5	City	Character	20	0	None
6	State	Character	2	0	None
7	Zip	Character	10	0	Ascend
8	SalesYTD	Numeric	10	2	None

Hit Enter and you’re ready to create the next field. Continue on until your Table Designer includes all the fields defined in the illustration below.

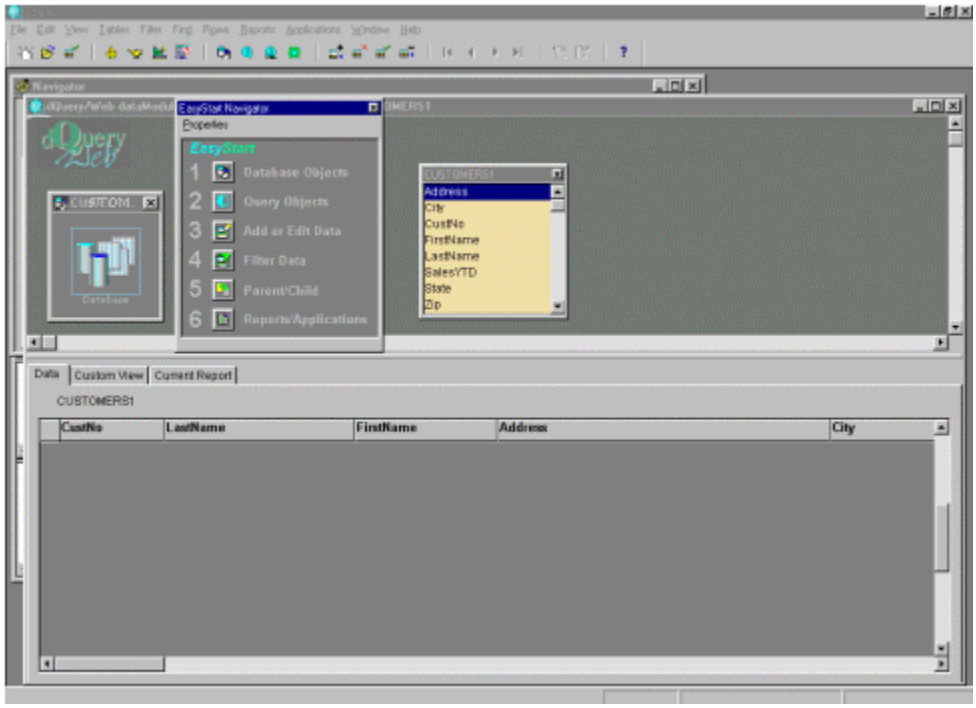
Since the SalesYTD field represents currency, be sure to set the decimal to “2” places. You should end up with three indexes, one each on the CustNo, LastName, and Zip fields. All of these should be set to “Ascend”.

Double-Check your entries and close the Table Designer to continue.

A dialogue box will appear asking you to save the changed information. Click Yes.

You now see a “CUSTOMERS1” Query object on the dQuery/Web Design Surface.

Figure 3.14 dQuery/Web design surface



Use your mouse to slide the new Query object toward the center of the screen. You can move any of the components to any section of the screen you desire. You can also grab the “splitter” - the line that crosses the

middle of the screen - and drag it up or down to change the size of the Design Surface relative to the Live Data Area. This is convenient when you have many Query objects in a dataModule.

Figure 3.15 Table Designer

:GuidedTourExample:invoices.dbf - Table Designer

Field	Name	Type	Width	Decimal	Index
1	CustNo	Numeric	10	0	Ascend
2	InvNo	AutoIncrement	4	0	None
3	Date	Date	8	0	Ascend
4	Amount	Numeric	10	2	None
5	BalanceDue	Numeric	10	2	None

Now that we've created one Query object for our customer data, we can quickly add another for our invoice data. Start with EasyStart, click on Queries and repeat the same process we used to create the Customers table. Only, this time, name the table "Invoices" and use the field definitions in the illustration that follows.

Double-check your entries. The Amount and BalanceDue fields are both for currency, so you need to set the decimal places to "2" for each of them. Both the "CustNo" and "Date" indexes must be set to ascend.

Click the "X" to close the Table Designer.

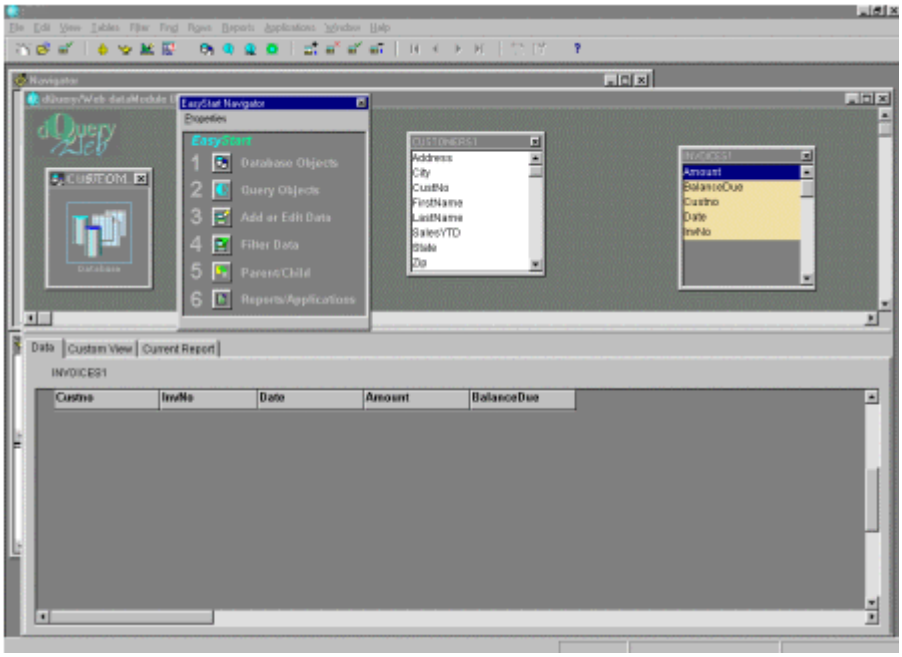
Click Yes when asked to save current changes to the Invoices table.

The second Query object is complete.

The Database object and the two Query objects (CUSTOMERS1 and INVOICES1) will now be displayed on the dQuery/Web design surface.

Use your mouse to slide the new INVOICES1 Query object over to the right of your screen.

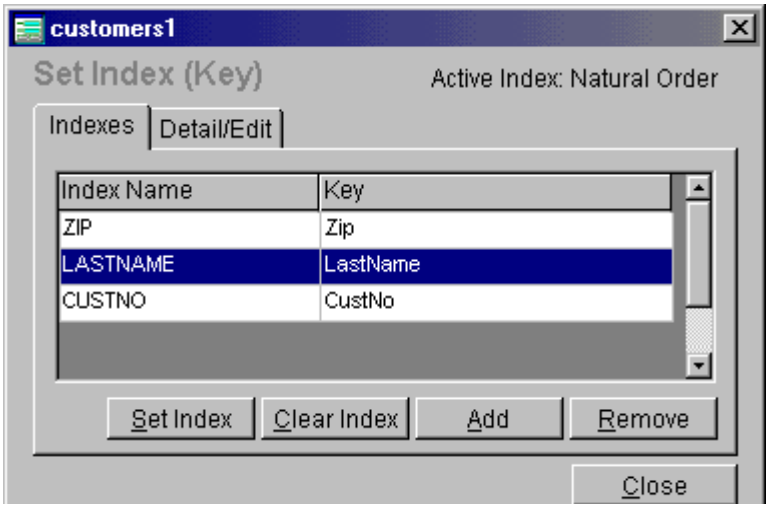
Your dQuery/Web screen should look something like this.



Setting Active Indexes

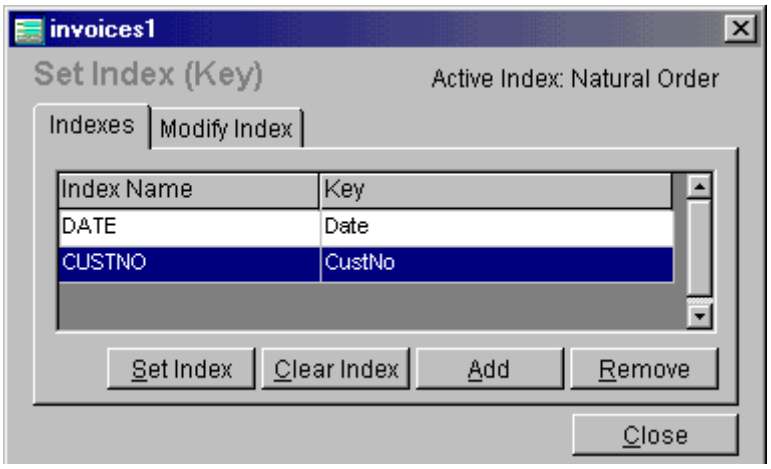
The "active" index determines the display order and search order for the Query. Right-click on the CUSTOMERS1 Query object and select the "Set Index" option from the menu.

Figure 3.16 Set Index dialog box



The Set Index (Key) dialog box will appear listing the available keys. Select LASTNAME, then click Set Index. The Active Index: Natural Order will change to LASTNAME.

Right-click on the INVOICES1 Query object and select the "Set Index" option from the menu. Set the Index to “CUSTNO”. Click Set Index.



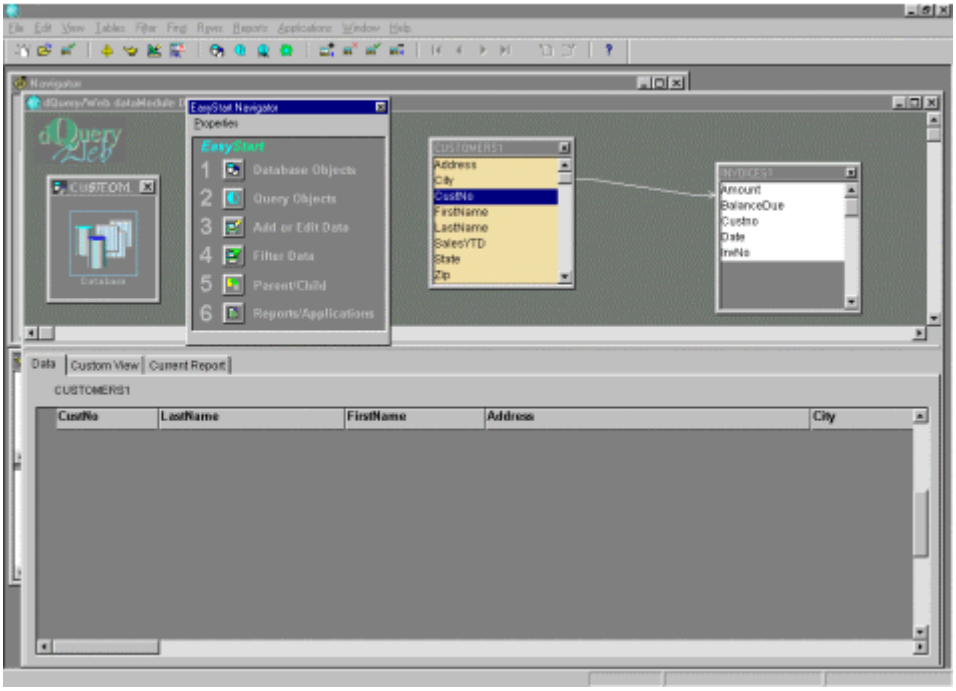
When your data is entered or reported, it will be in the order specified.

Tip When using SQL database engines, setting index may not make any difference. Remember that those database engines usually select the appropriate index files automatically.

Creating a Parent-Child Link

A parent-child link is a way of associating two Queries objects so that, as you move from row to row in the parent query, the child query automatically filters itself to show only the rows that match the current row in the parent query. Parent-child relationships are an extremely useful way to organize information.

Left click on the CustNo field of the CUSTOMERS1 Query object, hold the mouse down, and drag the field to the INVOICES1 Query object. Release the left mouse button and the two tables are linked! This is one of the many drag-and-drop features of dQuery/Web.



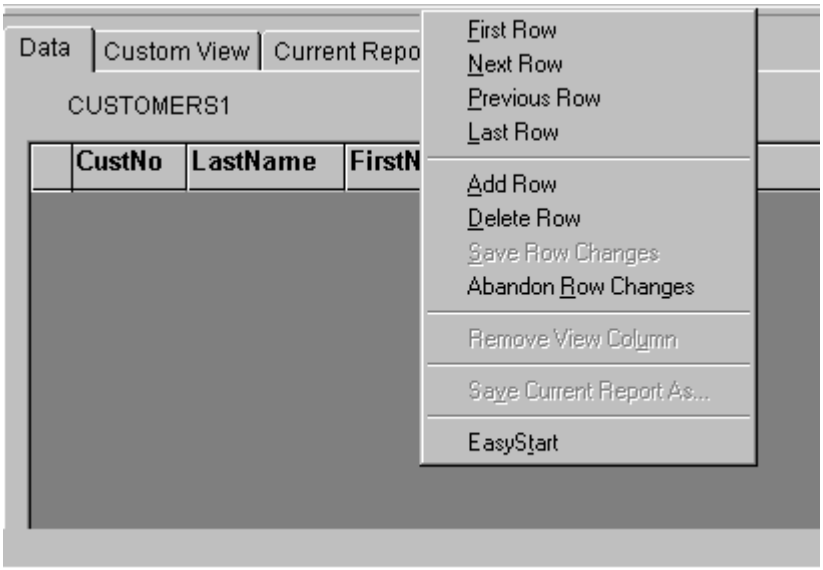
Whenever you navigate through the CUSTOMERS1 Query, the INVOICES1 Query will be filtered automatically to display only invoices belonging to the currently selected Customer!

Entering Data

Look at the bottom half of the dQuery/Web screen. This is the Live-Data Area. There are three tabs: “Data”, “Custom View”, and “Current Report”. First, make sure the Data tab is selected.

Now select the CUSTOMERS1 Query object by clicking anywhere on the body of the Query object.

Tip When you select a Query, by clicking on a Query object on the Design Surface, the data in the Live Data Area below changes to reflect the currently selected Query object.



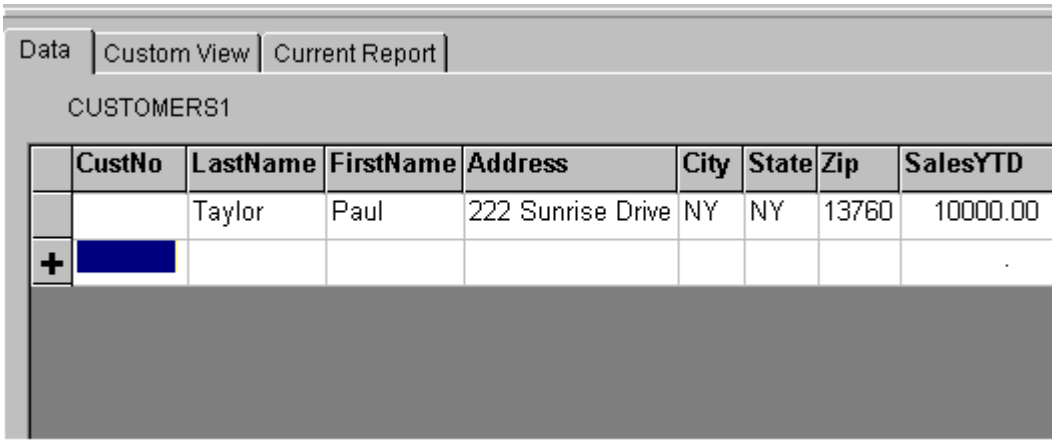
Right-click in the data area (the open area at the bottom of the Data tab). A menu will appear

Select the “Add row”.

A blank row is added to the Customer table. Add the following field information.

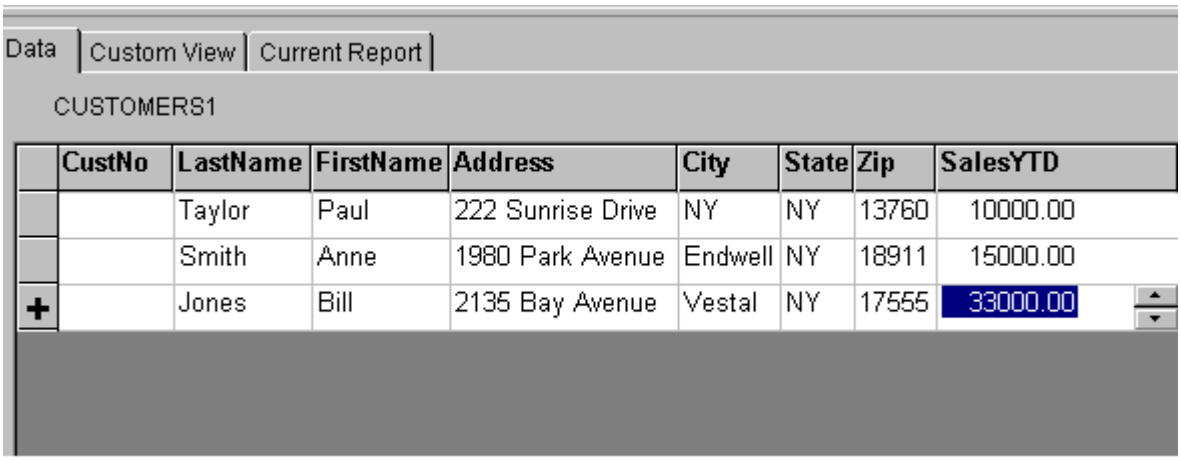
Enter “Taylor” “Paul’ “222 Sunrise Drive” “Endwell” “NY” and “13760” in the appropriate fields. Enter a SalesYTD of “10000.00”.

The CustNo field is left empty. This field is set to AutoIncrement. *dBASE SE* will assign the number automatically.



Hit the <Enter> key and a new blank row will appear. The first customer's data is complete.

Add two more customers to the table.



After the data is all entered, right-click in the data area and select Save Row Changes.

Note The three customers are ordered by last names. The active index key is set to LastName. To display the customers by CustNo order or ZIP order, set the active index to CustNo or ZIP.

Before continuing, make sure that Taylor, or CustNo 1, is the currently selected customer. You can do this by selecting any of the fields in Taylor's row.

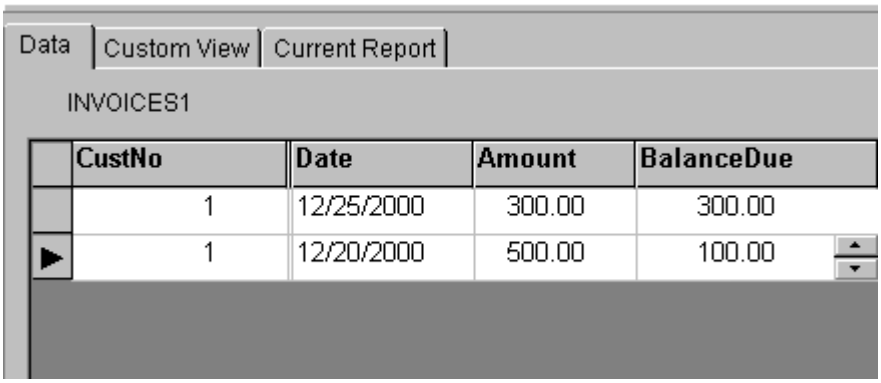
Select the INVOICES1 Query object by right-clicking on the INVOICES1 Query object on the Design Surface. Note that dQuery/Web indicates the current Query by changing its display to a "highlight" color. The name is also changed at the top of the Data tab on the Live Data Area.

Right-click in the Live Data Area. The context menu will appear.

Select the Add Row option.

Note that the first CustNo is already entered as "1". That's because CustNo 1 (Taylor) is the currently selected row in the CUSTOMERS1 table. Because of the parent-child link, any invoices added will automatically be displayed for the current customer.

Create the first invoice.



Enter “12/25/00” as the Date, “300.00” as the Amount, and “300.00” as the BalanceDue. The actual values are not important here, so any variations will work. Add one more invoice for Taylor so that your screen resembles the picture on the left.

Hit the <Enter> key to start a third row.

This time enter “2” as the CustNo value. You’re now entering an invoice for a different customer (in this case, Jones). Enter “12/26/00” as the date, “300.00” as the amount, and “300.00” as the balance due.

When you’re finished adding this invoice, right-click in the live-data area.

Select the Save Row Changes from the menu to save your work.

Notice what has happened in the live-data area. The invoice with CustNo “2” has disappeared. Because of our parent-child link, only invoices for the currently selected customer are displayed. The invoices for CustNo 2 (Jones) will be displayed when CustNo 2 (Jones) is the currently selected customer in the CUSTOMERS1 table.

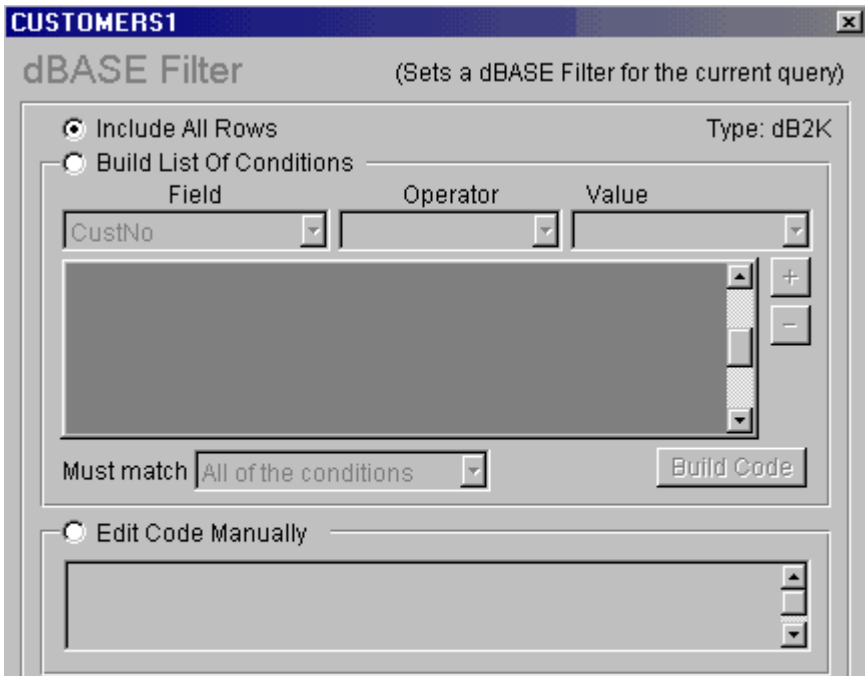
To view Jones’ invoices, select the CUSTOMERS1 Query object and change the currently selected customer to Jones (CustNo 2) by clicking on that row. Then select the INVOICES1 Query object again and note that Jones’ invoice is now showing in the live-data area.

Filtering Data

A filter changes how data is displayed. For example, you can set conditions for a date range (last week, this week, etc.), or for amount ranges on numeric fields like SalesYTD. This allows for full flexibility in viewing your data, live and on-the-fly.

Adding a filter is a simple task in *dBASE SE*. First select the CUSTOMERS1 Query object so that the customer data is showing in the live-data area.

Click on the Main Menu’s filter “dBASE Filter - Non-Indexed” to open the dBASE Filter dialog box.



Note that the Option “Include All Rows” is currently set. Select the option “Build List of Conditions”. This makes the “Field”, “Operator”, and “Value” options available.

The first condition will be where “CustNo = 1”. The Field CustNo is already displayed (it’s the first field on the list). Set the Operator to “=” and enter “1” for the Value.

Click on the “+” (Plus) button to add the condition to the list.

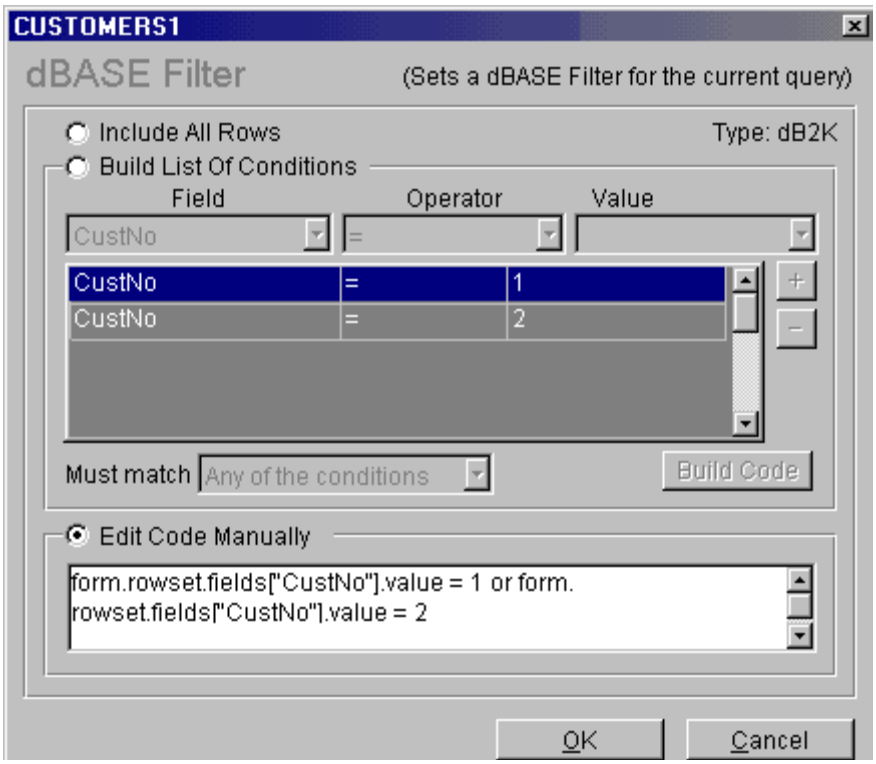
The next condition will be where “CustNo = 2”. Again, the Field is correct as is. Set the Operator to “=” and enter “2” for the Value, then click the “+” button.

To see all records where the value of CustNo is either “1” OR “2”, change the “Must Match” field to “Any of the Conditions”.

Finally, click the "Build Code" button to see the code for your list of conditions in the bottom pane. This last feature is useful for learning how these conditions are represented in the code.

Select the “Edit Code Manually” option. The code area of the screen will now be accessible. This is useful for building or altering code if desired.

The dialogue box will contain the following information. . .



Click OK. Now only customers Jones (CustNo=2) and Taylor (CustNo=1) appear in the data area.

Filters such as the ones we just added are simple, effective methods for reducing large data sets to more meaningful subsets. Filters are just one more way that dQuery/Web allows you to control information.

Select File | Clear All Filters to remove all filters. All the customer records now appear in the data area.

Tip Take a few moments to experiment with the Filter By Grid options under the Filter menu. When you Begin Filter By Grid, a row clears on the data tab of the Live Data Area. Type data into any fields you want to match. Click on Apply Filter By Grid to execute the search. Your live-data grid now displays only rows that exactly match the data you entered. Keep in mind that Filter By Grid only works for exact matches, not partial matches or ranges.

Working With Custom Views

Custom Views are one of dQuery/Web's most innovative features. They allow you to select and display specific fields from Query objects for custom views of your data. Once you've selected the fields you need, you can use the custom view to generate No-Click Reports, which will be illustrated in the next step. You can have any number of custom views (represented by reports) all associated with a single dataModule.

Tip Since we're working with Query objects, and not tables, dQuery/Web offers the unique ability to combine fields from dissimilar database engines into a single view. For example, you can take the first column in your custom view from an Oracle field, the second column from an SQL Server field, and the third column from a dBASE field.

Select the CUSTOMERS1 Query object.

Select the "Custom View" tab on the Live Data Area. It should be empty.

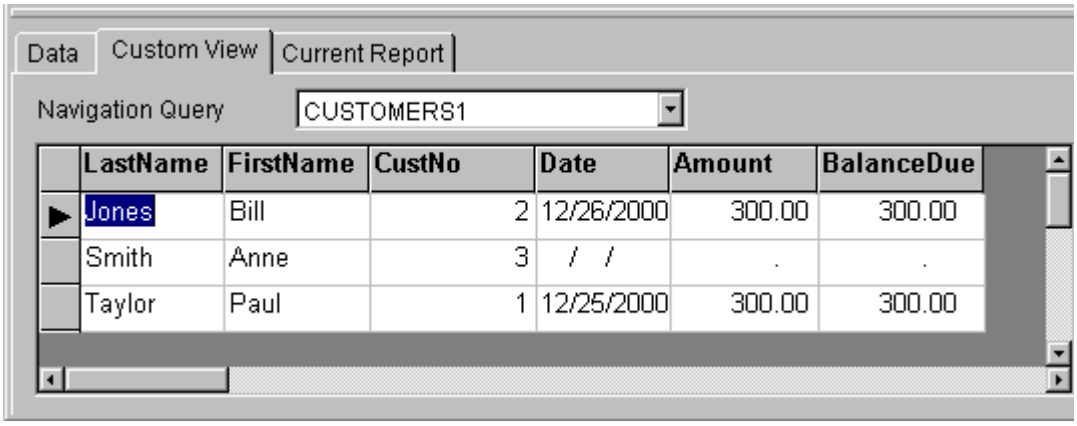
Drag-and-drop the LastName field down to the live-data area. Note that we now have all our customers' last names showing in a single column.

Drag-and-drop the FirstName and CustNo fields from CUSTOMERS1 onto the data area.

Select the INVOICES1 Query object.

Drag-and-drop the Date, Amount, and BalanceDue fields onto the data area.

The Custom View will contain the following information.

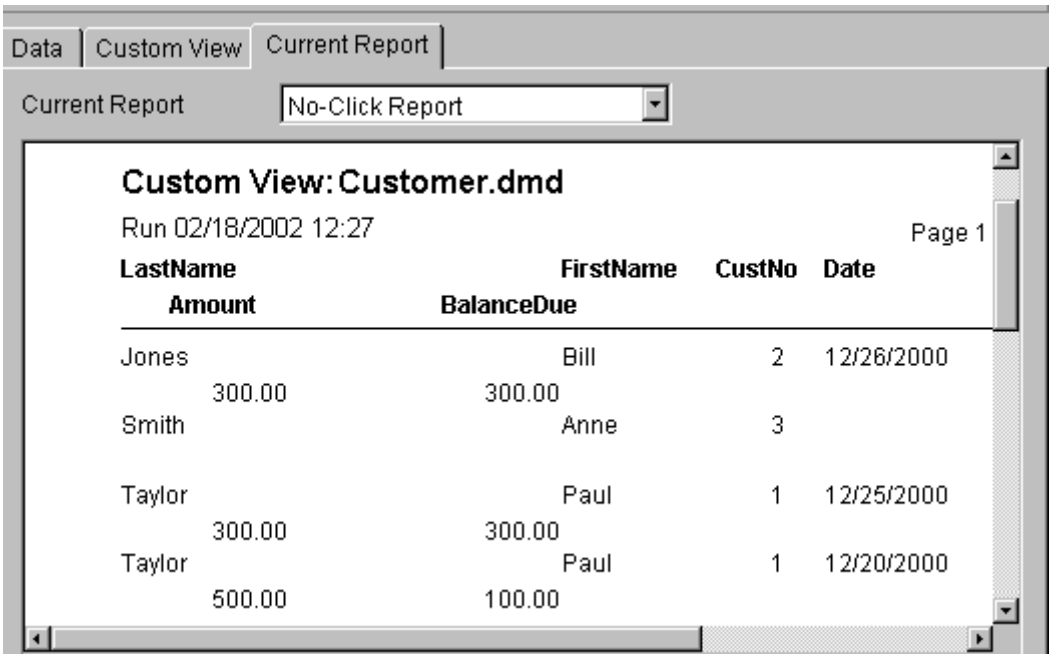


The view you've created will automatically be saved with the dataModule. It contains data from two related tables: CUSTOMERS and INVOICES.

This tool lets you drag-and-drop any of the Query object fields to relate your data in any way you see fit. The real benefit of the Custom View becomes apparent when you need to combine information from a variety of sources in a single report or application.

No-Click Reports

Click the Current Report tab to see the results of the Custom View and the power of dQuery/Web No-Click Reports.



By switching to the Current Report tab, you've caused dQuery/Web to generate a No-Click Report from the data you dragged and dropped into the Custom View. This data reflects all the invoices entered for all the customers. You entered one invoice for Jones, two for Taylor, and none for Smith. Note that if the data had been filtered, the report would have reflected the filters.

In a matter of seconds, your custom view selections have been organized and listed into a report that can be saved, edited, sent to the printer, to an HTML file or directly to the Web!

You can change how you navigate the data in your new report. You're currently navigating by customers. To navigate by invoices, select the Custom View tab.

Change the Navigation Query field (located at the top of the live-data area) from CUSTOMERS1 to INVOICES1.

Select the Current Report tab once more and you'll see only the invoices for Taylor, the currently selected customer.

Before proceeding, go back to the Custom View tab and set the Navigation Query back to CUSTOMERS1.

Click, once again, on the Current Report tab.

To save this report select File | Save Current Report As.

If asked to save the dataModule, select Yes.

Enter "CustomerSales" as the report name and make sure to save it in the Customer folder.

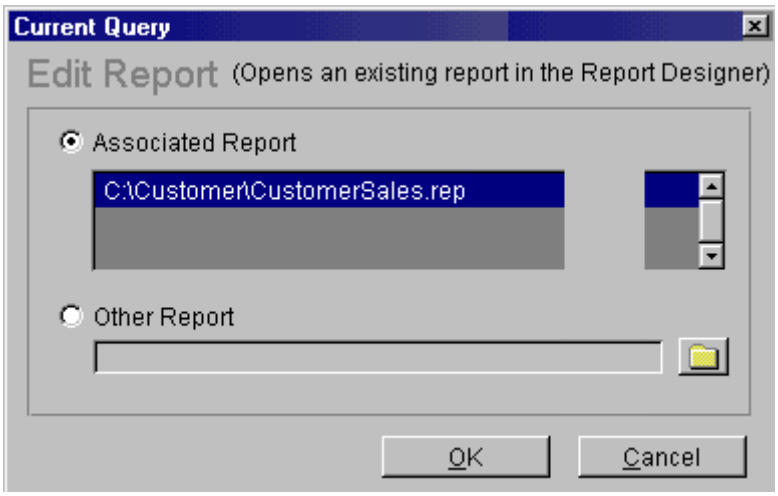
If you drop down the field-list of Current Reports (located at the top of the live-data area) you'll see that you now have two reports: one called No-Click Report and one called CUSTOMERSALES.REP.

Change the CUSTOMERSALES report to illustrate how you can have multiple reports associated with a single dataModule.

Make certain the "No-Click Report" is the currently selected report.

Select Reports | Edit Reports. The following dialogue box will open.

Figure 3.17 Edit Report dialog box



The Edit Report dialog allows you to edit any reports associated with the current dataModule. If you want to edit other reports there's an option near the bottom of the window that allows you to select any *dBASE SE* report on your network.

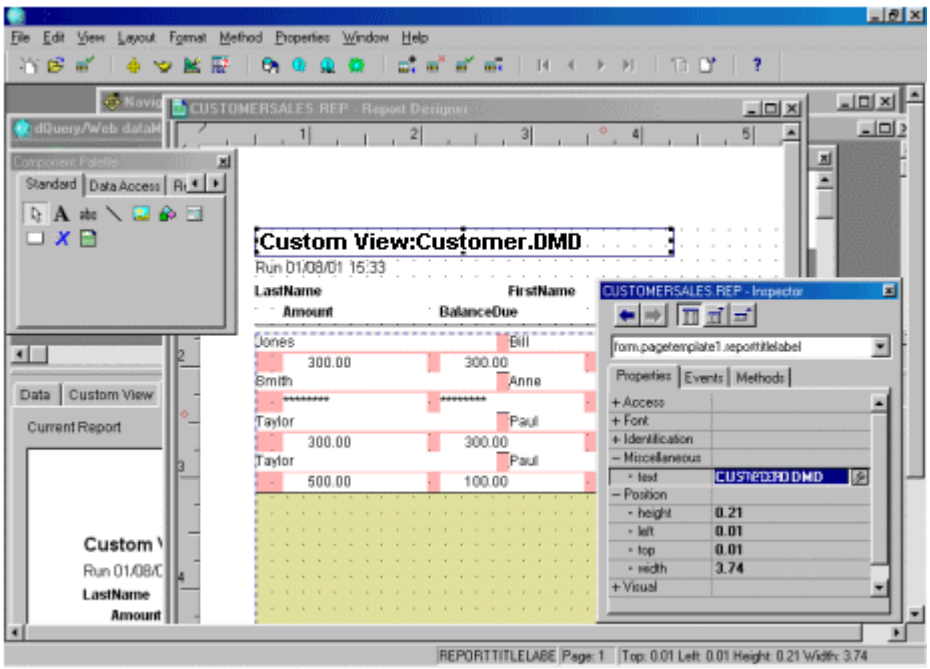
The CustomerSales report is highlighted, so click OK.

The *dBASE SE* Report Designer will now appear with the Component Palette, the Inspector, and the formatting toolbar at the top. Your screen will be similar to Figure 3.18.

The Component Palette provides a full range of objects that may be required to create a report.

Because you closed it earlier, the Inspector may not appear at this point. If it's not on the screen, right-click on the Report Designer and select "Inspector" from the context menu.

Figure 3.18 Report Design surface



The *dBASE SE* Report Designer is a full-blown design tool. With it you can modify all aspects of your reports.

For this example you will make one change, the report title. Select the report title. Click on the Inspector, Properties tab.

Tip To see all available properties, right click on any blank grey area of the Inspector and click on “Expand All Categories”) on the context menu.

Scroll down to the “text” property underneath Miscellaneous and change the value entered to “Customer Sales Report”. As you type, note that the title in the report design surface also changes. Press Enter to save the change.

Close the report.

Click Yes when prompted to save it.

Now drop down the list of reports on the Current Report tab and select CUSTOMERSALES. Note that the title in the displayed report changes. Going back and forth between the two reports demonstrates how dQuery/Web and the *dBASE SE* Information Toolset can quickly and easily create, modify, and display a variety of reports associated with a single dataModule.

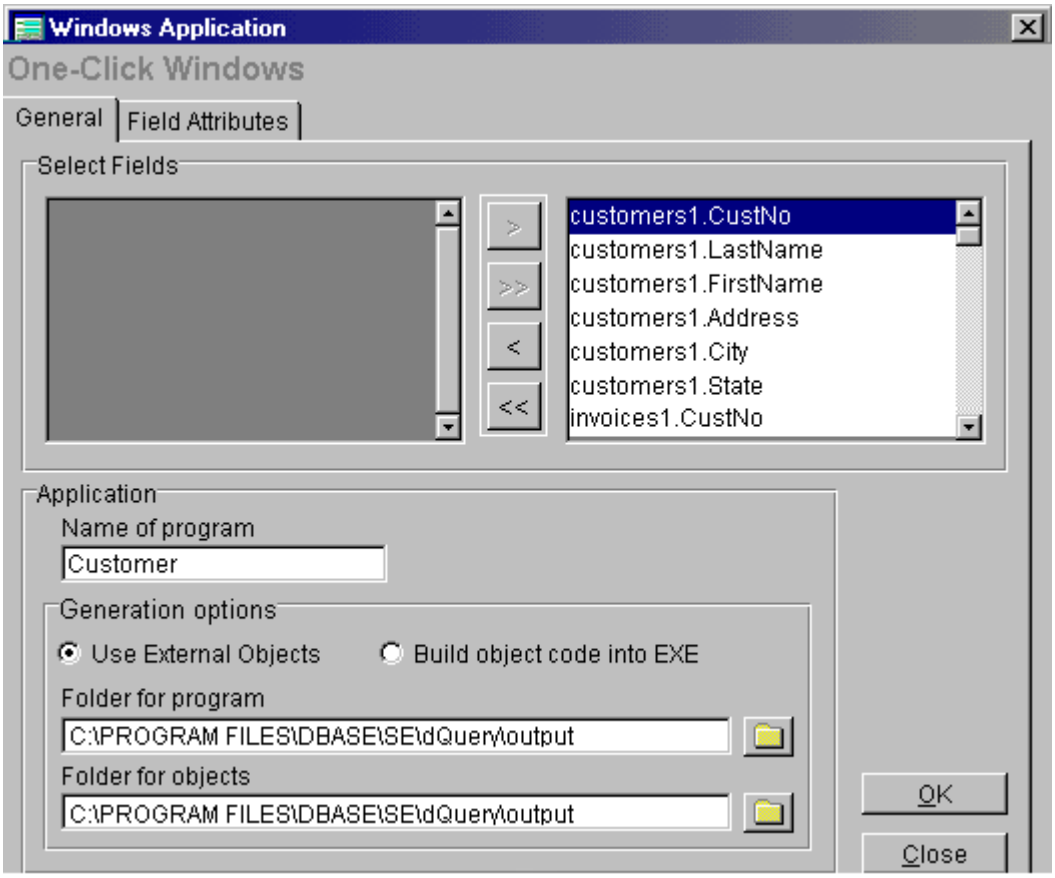
One-Click Windows

You have created a Database object, tables, Query objects, custom views, and custom reports. These are the building blocks of an application. The next logical step is to assemble these components and deploy them as an application. dQuery/Web provides the tools to automatically generate both Windows and Web applications based on your dataModule design. In this step you will quickly create a Windows application with just a single click of the mouse.

Select Applications | One-Click Windows.

Click Yes to save the DataModule. The One-Click Windows dialog box will open.

Figure 3.19 One-Click Windows dialog box

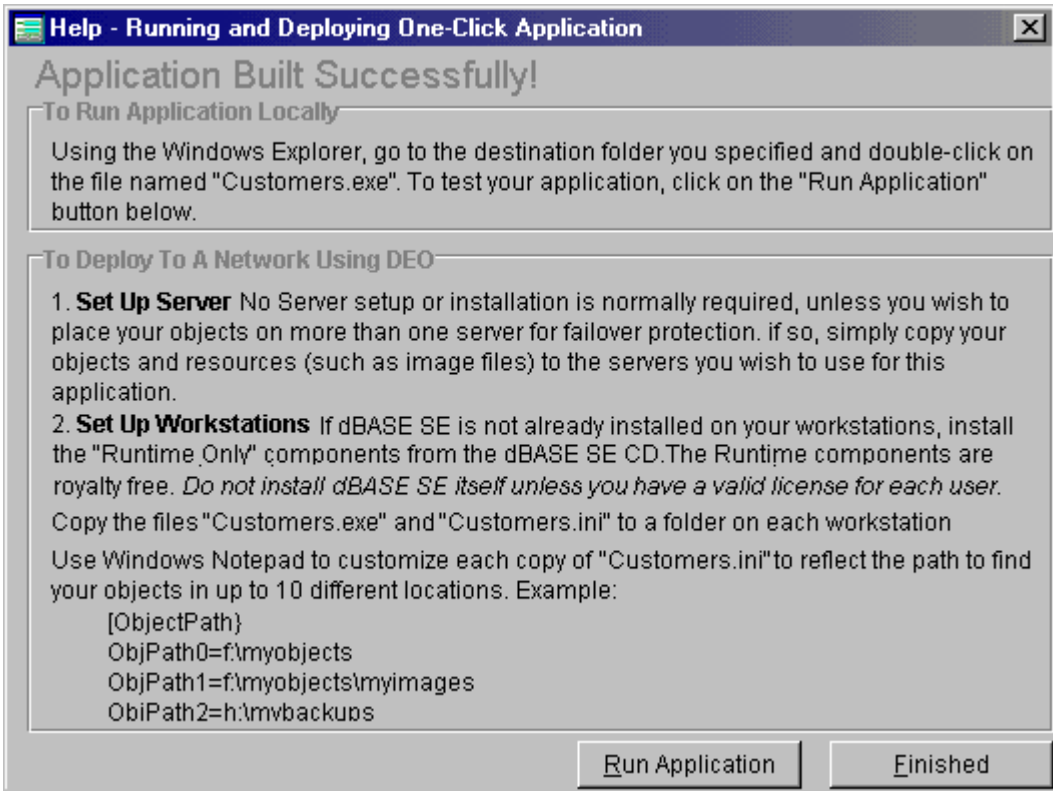


Two tabs are accessible: “General” and “Field Attributes”. The first time this window is opened you'll have to set your folders for both program files and object files on the “General” tab. Use the two folder pushbuttons to change both fields to C:\Customer.

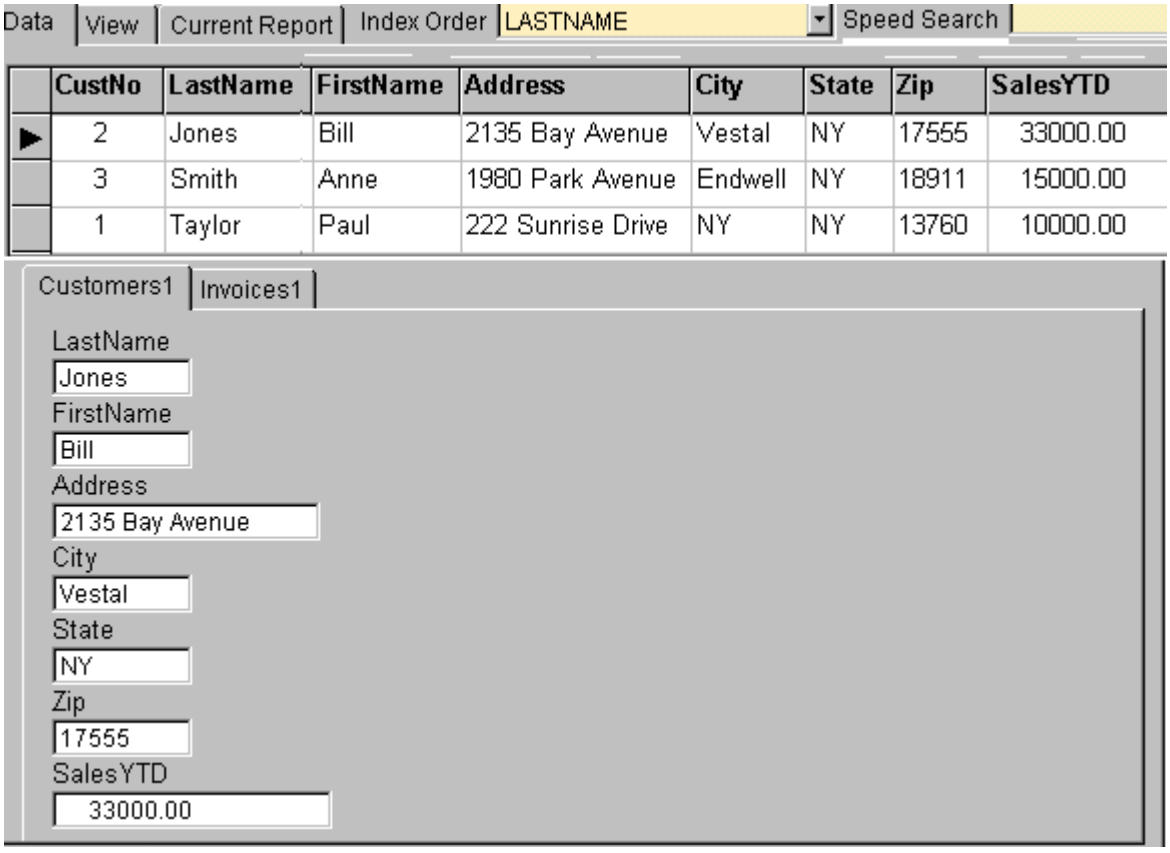
dQuery/Web will remember the settings you've made so in the future you only need to click the OK button to generate your Windows application.

Note that all the fields for your two tables are pre-selected. By selecting the Field Attributes tab you can modify how the field labels appear on data entry screens, and whether or not a given field is required during data entry. Accept the default settings.

Make sure your folder settings point to the Customer folder and click OK. The Application Built Successfully! dialog box opens. Click the Run Application button to see your application in action.



:



Without entering a line of code, you now have a stand-alone, royalty-free, executable application. Do the following to test the capability of your application.

The data-entry screen for the Customers table is displayed. Clicking on the INVOICES1 tab will display the data entry screen for the Invoices table.

If you select Taylor in the Customers table and select the INVOICES1 tab, you'll be looking at the live data—the invoices—for Taylor.

You can edit existing data and enter new data using the Edit menu.

You can locate data by opening the Rowset menu and selecting the Begin Locate by Form option. This option will allow you to locate data by entering data in any field.

Similarly, by opening the Filter menu and selecting the Begin Filter by Form option you can set a variety of filters simply by entering data into the form.

To perform a fast index speed search: Select the customer table in the view tab.

Switch to the Data tab.

With the Index Order set to LASTNAME, type "t" in the Speed Search field.

The first customer that begins with "t" (Taylor) is now highlighted.

Finally, all the reports associated with your dataModule are available in the Current Report tab.

Close the application. Click Finished. The dQuery/Web Design Surface appear.

Chapter

4

Introduction to programming in dBL

dBL is an object-oriented, event-driven programming language that allows developers to create sophisticated, scalable applications using objects and classes.

Objects are a means of encapsulating collections of variables, some containing data, others referencing code. Classes are a mechanism for creating reusable groups of objects. In *dBASE SE*, you can

- Create objects from standard classes or custom classes you declare
- Add custom properties to an object with a simple assignment statement
- Declare custom classes
- Declare classes that are based on other classes and inherit their properties and methods
- Write highly reusable code by building hierarchies of classes

"Hard coding" vs. visual programming

Using a text editor, you can write programs from scratch by typing each command, line after line. That's how most programmers used to write programs: the hard way. With *dBASE SE*, you use design tools to generate the program code for you. The most painstaking requirement of traditional user-interface programming—guessing how fields and menus will appear after positioning them with coordinates—is obsolete. You place objects on a form exactly where you want them, and let *dBASE SE* figure out the coordinates. That's visual programming.

But there's more to visual programming than just laying out forms. The objects you place on your forms have a built-in ability to respond to a user's actions. A pushbutton automatically recognizes a mouse click. A form "knows" when the user moves or resizes it. You don't need to figure out what the user does and how it happens. *dBASE SE* handles that. You just tell the objects how to respond to these events by assigning procedures that will execute when the events occur.

The results of programming visually are applications that are easy to create and easy to use. They're easy to use because they're event-driven.

Advantages of event-driven programs

The three major kinds of user interfaces are

- Command-line, where the user types commands at a prompt. The MS-DOS operating system, the dBASE Dot Prompt (in DOS versions) and Command window (in Windows versions), are examples of command-line interfaces.
- Menu-driven, where the user selects choices from a hierarchy of menu items. Most applications written using prior versions of dBASE provide menu-driven interfaces.

- Event-driven, where the user interacts with visible objects, such as forms containing pushbuttons and list boxes, in any sequence. The user interface is event-driven, and you can create event-driven applications using dBL.

Using traditional programming techniques, you can build menu-driven user interfaces. In these applications, the program dictates the sequence of events. If the user selects Order Entry from a menu, the program typically walks through a series of screens asking the user for information: enter the customer name, enter the date and purchase order number, enter the line items, enter the shipping charge.

These menu-driven techniques are not well-suited for programming in event-driven environments such as Windows. In an event-driven application, the user controls program flow. A user can click a button, activate a window, or select a menu choice at any time, and the program must respond to these events in whatever sequence they occur.

In a well-designed event-driven application,

- The user can focus on the task, not on the application. The user doesn't have to learn a complex hierarchy of menu choices. Rather, when choosing to enter an order, the user sees an order form similar to a familiar paper form.
- The user doesn't need to re-learn how to perform tasks. Because you create an application's interface using familiar objects such as pushbuttons and list boxes, common operations—opening a file, navigating a form, and exiting the application—are more consistent across applications.

Most important, event-driven interfaces reflect the way people work in the real world. When clerks write up orders, they pick up forms and fill them out. When they receive checks for orders, they pick up the invoices and mark them as paid. This natural flow of work follows an object-action pattern. That is, a clerk selects an object (an order form, an invoice) and performs some action with it (fills out the order, posts the check).

Likewise, in an event-driven application, the user selects objects (forms, entry fields, pushbuttons) and performs actions with them.

How event-driven programs work

In an application, the form is the primary user-interface component. Forms contain components, or controls, such as entry fields and pushbuttons, with which the user can interact. The controls recognize events, such as mouse clicks or key presses.

You attach code to event handlers of controls, such as *OnClick* or *OnLeftMouseDown* (most begin with *On*), that correspond to specific events. For instance, when a user clicks a pushbutton, the *OnClick* event handler executes.

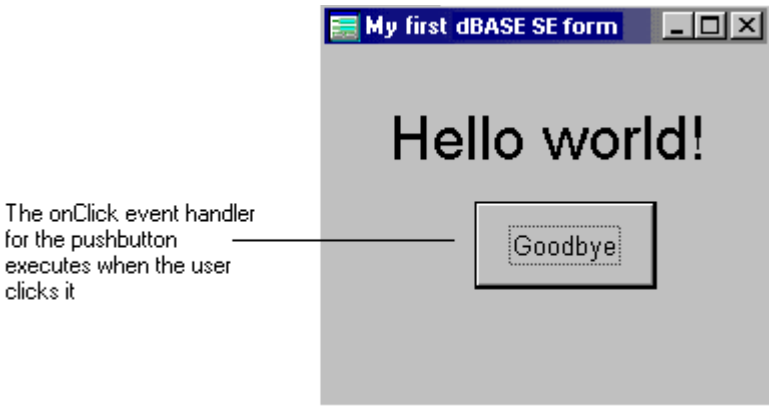
Specifying event handlers for forms is similar to using the ON commands in dBASE DOS, such as ON KEY LABEL or ON ERROR. Like an event handler, the ON command designates some program code to execute when an event, such as a keypress or a run-time error, occurs. However, the events handled by the ON commands are limited and are not associated with user-interface objects.

In a typical event-driven application,

1. The application automatically displays a start-up form.
2. The form, or a control on the form, receives an event, such as a mouse click or keystroke.
3. An event handler associated with the event in step 2 executes.
4. The application waits for the next event.

Figure 4.1 shows a sample "Hello world!" form with one pushbutton on it that is labeled "Goodbye". After displaying the form, *dBASE SE* waits for an event. The user can move, resize, minimize, or maximize the form. When the user clicks the pushbutton, *dBASE SE* executes the *OnClick* event.

Figure 4.1 Sample event handler for a "Hello world" form



The following code, a. WFM file generated by the Form designer, creates the form just described. This code follows the general structure of all forms generated by the Form designer. For now, don't try to understand every line. Just look at the general structure to get a sense of how forms are created, properties are set, and event handlers are assigned to events.

```
parameter bModal
local f
f = new Hello( )
if (bModal)
  f.mdi = .F. && ensure not MDI
  f.ReadModal( )
else
  f.Open( )
endif

CLASS Hello OF FORM
  with (this)
    Height = 16
    Left = 30
    Top = 0
    Width = 40
    Text = "My first dBASE SE form"
  EndWith

  this.TEXT1 = new TEXT(this)
  With (this.TEXT1)
    Height = 3
    Left = 11
    Top = 3
    Width = 33
    Metric = 0
    ColorNormal = "N/W"
    FontSize = 23
    Text = "Hello world!"
  EndWith

  this.BUTTON1 = new PUSHBUTTON(this)
  With (this.BUTTON1)
    onClick = class::BUTTON1_ONCLICK
    Height = 2
    Left = 19
    Top = 9
    Width = 13
    Text = "Goodbye"
    Metric = 0
    StatusMessage = "Click button to exit"
    Group = True
  EndWith

  // {Linked Method} Form.button1.onClick
  Function Button1_OnClick
    DO WHILE (Form.Height > 0) .AND. (Form.Width > 0)
      Form.Text1.Text = "Goodbye"
      Form.Height = Form.Height - 1
      Form.Width  = Form.Width  - 1
      Form.Top    = Form.Top    + .5
      Form.Left   = Form.Left   + .5
    ENDDO
```

```
        Form.Close( )
        return
    ENDCLASS
```

Developing event-driven programs

All you really need to develop event-driven programs is the Form designer and Menu designer. Using the designers and their tools, you can build data-entry forms, dialog boxes, menus—all the visible components of an application. Then use the built-in Source editor to tie the components together by writing procedures to execute when events occur.

Projects that are more complex, however, require planning and a good design. That's where object-orientation helps. Using object-oriented techniques, you can group related information into your own objects, build classes of related objects, and create new objects by making easy modifications to existing ones.

Chapter

5

Creating an application

Using *dBASE SE* design tools, you can create the visual elements of an application quickly, in a manner that promotes reuse of design elements rather than repeated reinvention. Using the Component palette, you simply drag and drop both the visible user-interface components (also known as *controls* or *objects*) and the invisible data objects onto forms. The Inspector gives you an easy way to set an object's attributes, or *properties*, and access its event handlers and methods directly, without hunting.

As you work with the design tools, *dBASE SE* writes the corresponding dBL code for you. If you prefer to do most of your developing in the Source editor, the code you write is reflected in the designer, and you can see immediately what your form looks like and how it runs. In either case, the entire source code for your form is available to you in the Source editor at all times. Press F12 to toggle between the source code and the visual designer.

This section discusses the basic steps of creating an application in *dBASE SE*. It includes information on

- Creating projects and using the Project Explorer to manage them
- Planning and creating forms
- Code generated by the Form designer
- Types of form windows (MDI, SDI, modal, modeless)
- Using ActiveX controls, container components, and multi-page forms
- Creating custom classes, custom components, and data modules and using them in a form or report.

Menus are created separately with the Menu designer. You program a form to display a pull-down menu by referencing it in the form's *menuFile* property; a popup menu you reference manually in the Source editor.

Creating an application (basic steps)

At the simplest level, designing an application in *dBASE SE* consists of these steps:

1. Create a project that will hold all your application's files (and any other files you need to have handy while you're developing). Files you create while the project is open will be added to the project automatically. You may also add to the project any relevant files you've already created.
2. If you're creating tables from scratch, plan your tables so you can link them to one another.
3. Plan your directory structure. For example, you may want to put tables in a DATA subdirectory and images in an IMAGES subdirectory.
4. Use the BDE Administrator to create Borland Database Engine (BDE) aliases for all SQL databases and local tables. You can then access those databases through the BDE and through your application. This procedure is described in "How to connect to an SQL database server" on page 2-3.
5. If several forms (or reports) will be using the same data-source setup (table relationships, SQL statements, methods, and so on) create a data module so you only need to create the data-source setup once. The "dBASE

data model" and how to access tables using Data Access components is described in Chapter 6. "Creating dataModules" is described on page 6-10.

6. Create custom form and report classes to give your application a unified look (see page 5-11).
7. Create the forms (data entry forms, dialog boxes, and so on) that make up the user interface of your application.
8. Create any reports that your forms will link to or run, using the *dBASE SE* Report wizard and integrated Report designer (see Chapter 7, “Using the Form and Report designers”, and Chapter 12, “Designing reports”).
9. Test and debug, using the *dBASE SE* debugger (see Chapter 10, “Debugging applications”).

Project files (overview)

Every application should have a project (.PRJ) file. The project file contains pointers to all your application files (tables, forms, queries, bitmaps, and so on). In addition to keeping things organized, having a project file lets you

- Set properties for the project as a whole; for example, you can set compile options, like preprocessor directives.
- Set properties for individual files.
- Designate which file should be the first to open when your executable file is run.
- See instant previews of your files in the Project Explorer.
- Open several files at once.
- And if you create a file while a project is open, the file is automatically added to the project.

Creating a project file

1. Choose File | New Project.
2. In the New Project dialog box, type a name for the project.

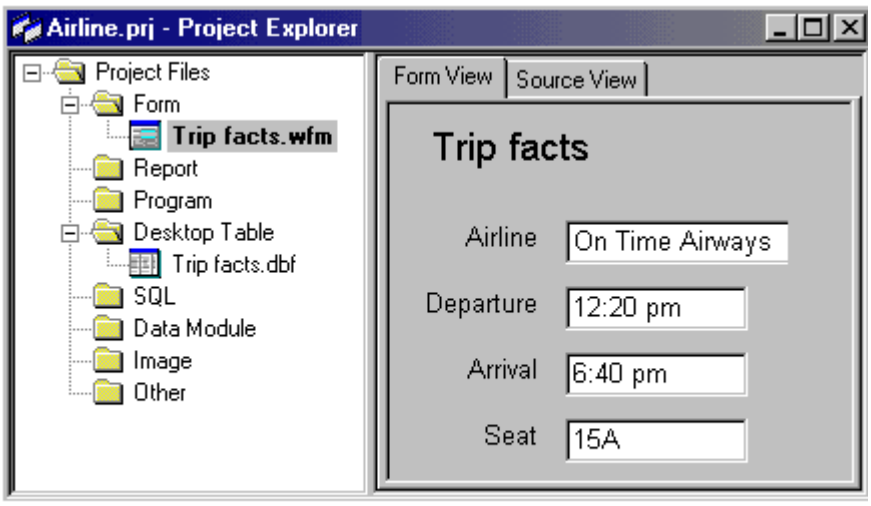
   This name will become the name of the project folder and the project file.

   The Project Location text box updates to show your project name at the end of the path, in a subdirectory called My Projects (which is created during Setup). All of your projects will go in this directory, unless you specify otherwise. If you want to specify a different location, leave the Project Name text box blank, and add your new project name (directory) to the end of your specified Project Location path. A new directory is created at that location.
3. Choose OK.

The Project Explorer opens, displaying a tree view of several empty folders on the left and a viewing panel on the right. The left panel looks and acts much like the Windows Explorer. Files have context (right-click) menus that let you open, run, compile, create something new, and so on.

When you have files in your project, the right-hand pane displays read-only views of a file you select. A file may have more than one view, in which case more than one tabbed page may appear. For example, one tab may show source code for a custom control, another, the visual object.

Figure 5.1 Project Explorer displaying a view of a file



A typical project tree displaying a view of the selected file. Clicking the Source View tab displays the code for this file.

Adding files to a project

You can add files to a project in three ways:

1. Create a file while a project is open. The file is added to the project automatically.
2. Drag files to the Project Explorer from the Navigator or Windows Explorer. You don't have to aim them at a particular folder; they will automatically drop into the correct folder. File types are assigned to folders in the Windows registry. If a file extension is not registered, *dBASE SE* places it in the Other folder (at the bottom of the tree).
3. Choose Project | Add Files To Project. You can select as many files as you want and choose Open to add them to the project. *dBASE SE* places them in the correct folders.

Notes on the Project Explorer

- To save a project, choose File | Save Project (Ctrl+S).
- You can create your own folders (Project | Folder, or use the context menu). In the dialog box that appears, give the folder a name and assign extensions to it. The folder and its associated extensions are added to the registry.
- Clicking on a folder in the left pane gives you a List view in the right pane. Right-click for a menu. You can multi-select files in the right pane and perform the same operation on them as a group: open, compile, delete, and so on.
- You can have only one project open at a time.
- The BIN directory contains a utility (CAT2PRJ) that converts your *Visual* dBASE 5.x Catalog files to Project files.

Building the user interface

dBASE SE forms (and the menus and toolbars you create for them) make up the user interface of an application. The forms you design become the windows and dialog boxes of your application. Some of the components you place on a form are the controls that let a user operate the application. Other components are data objects that are invisible when the application runs but that link the application with data in tables.

Components contain three kinds of information:

- **State information**. Information about the present condition of a component is called a *property* of the component. Properties are named attributes of a component that a user or an application can read or set. Examples of properties are *Height* and *Color*.
- **Action information**. Components generally have certain actions they can perform on demand. These actions are defined by *methods*, which are procedures and functions you call in code to tell the component what to do. Examples of component methods are *Move* and *Refresh*.
- **Feedback information**. Components provide opportunities for application developers to attach code to certain occurrences, or *events*, making components fully interactive. By responding to events, you bring your application to life. Examples of component events are *OnClick* and *OnChange*.

Form design guidelines

The process of designing forms involves clarifying the specific needs of your application, identifying the information you want to work with, and then devising a design that best meets your needs. This section briefly describes the process.

Goal of form design

The goal of form design is to display and obtain the information you need in an accessible, efficient manner. The form should encapsulate data so that it may be run without affecting other forms that use the same data. *dBASE SE* makes this simple.

It's important for your design to provide users with the information they need and clearly tell them what they need to do to successfully complete a task. A well-designed form has visual appeal that motivates users to use your application. In addition, it should use limited screen space efficiently.

Purpose of a form

Each form in your application should serve a clear, specific purpose. Forms are commonly used for the following purposes:

- Data entry forms provide access to data in one or more tables. Users can retrieve, display, and change information stored in tables.
- Dialog boxes display status information or ask users to supply information or make a decision before continuing with a task. A typical feature of a dialog box is the OK button, which the user clicks to process the selected choices.
- Application windows contain an entire application that users can launch from an icon off the Windows Start menu.

You should be able to explain the purpose of a form in a single sentence. If a form serves multiple purposes, consider creating a separate form for each.

Some guidelines for data entry forms

When designing data entry forms, consider the following guidelines:

- If data resides in multiple tables, use a query or data module that defines the relationships among tables.
- If users need access to only some of the information in a table, use a query or data module that selects only the records and fields you want.
- Determine the order in which users will want to display records, for example, alphabetically, chronologically, or by customer number. Use a query with indexes that arrange records in the order the users will want.
- Identify tasks users will perform when working with data on the form, and provide menus, pushbuttons, and toolbar buttons that users can choose to initiate tasks.

When designing a form, you can provide validation criteria on a field-by-field basis. Use the following questions to help decide which criteria you need.

- Do you require an entry for the field, or can users leave it blank?
- Are duplicate entries allowed?
- Must the data fall within a valid range?
- Must the data appear in a specific, fixed format, such as a phone number?
- Are valid entries limited to a list of values? If valid entries are *not* limited to a list of values, you can speed up data entry by compiling and displaying a list of frequently entered values, which also allows users to enter companies not on the list?

You can also provide form-level validation in a *canClose* event. This event returns True or False based on a condition you specify. If the condition is not met, the form will not close. For example, you could use *canClose* if a user has not saved the last row entered. In the method you write for this event, you would ask if the user wants to save the data and, if yes, allow the user to do so.

You can associate some field types with particular controls. By default, each dBASE SE field type is associated with a specific control type. For example, a Numeric field type uses a spin box control by default. You can change these associations to make data entry easier and more efficient in your particular application. Right-click the Field palette, and choose Associate Component Types.

If your form needs to contain many fields or controls, consider using the Notebook component. Divide controls into related groups and list each group on a separate page of the notebook. Or use a multi-page form with buttons for page navigation. Or, instead of buttons, add a TabBox component and set various TabBox properties to create page tabs and name each page.

Designing the form layout

You can put controls anywhere on a form. However, the layout you choose determines how successfully users can enter data using the form. Here are a few guidelines to consider:

- Put similar or related information together in a group, and use visual effects to emphasize the grouping. For example, put a company's billing and shipping address information in separate groups. To emphasize a group, enclose its controls in a distinct visual area using a rectangle, lines, alignment, and colors.
- On a form, the Tab key moves the focus from one control to another. Think about the order in which the user will be moving (tabbing) through these controls on the form. The basic pattern is from left to right, top to bottom. However, users may want to jump from one group of controls to the beginning of another group, skipping over individual controls.
- Users are typically more productive when a screen is clean and uncluttered. If it appears you're trying to cram too much information onto a single form screen, consider using a form with multiple pages, or a main form with optional smaller forms that users can display on demand.

Guidelines for using the z-order

All objects on a form exist in layers. When a form contains two or more controls, the plane in which a control exists always lies in front of or behind the plane in which another control exists. This affects two aspects of the form:

- **Visual layers**, or the z-order, which indicates the z-axis (depth) of the layout. This determines what appears in front of (or on top of) what. Even when controls are laid out side-by-side, each control is in a unique plane of the form. That is, each control occupies a unique position in the z-order.
- **Tabbing order**, which determines the order in which controls receive focus when a user presses Tab.

Each control is numbered to indicate its z-order position. The item in the back is number 1. The next item in front of the first item is number 2, and so on. By default, the z-order is the same as the order in which you added controls to the form. However, this is probably not the tab order you want. By choosing View | Order View, you can see the order of controls on a form and change the order by clicking on the numbers. Another way to change the order is to choose Layout | Set Component Order.

Another need for z-ordering is when you use a rectangle control, for example, to group a series of RadioButtons. The RadioButtons must appear on top of the rectangle, so you need to place the rectangle behind them in the z-order.

Creating a form

You can create a form in two ways:

1. **Use the Form wizard.** The Form wizard creates a data-entry form. It presents you with a series of options, and based on your selections, creates the form. It saves time, and you can modify and further develop the form in the Form designer (see ”Using the Form wizard’ on page 5-6).
2. **Use the Form designer.** Here you can create a form from scratch, by dragging components onto the form and specifying their properties. Since components have built-in functionality, you can actually create very simple applications with little or no coding. However, to create more complex and highly customized applications, you need to write your own event handlers and methods for various components.

During form creation, press F12 to open the Source editor, where you can see and edit the dBL code generated by *dBASE SE* as you design your form. Pressing F12 toggles you between the visual design view and the integrated Source editor.

Using the Form wizard

To use the Form wizard,

1. Choose File | New | Form. Or double-click the leftmost Untitled icon on the Forms page of the *dBASE SE* Navigator. Then choose Wizard.
2. Go through the steps of the wizard, clicking the Next button when you’re finished with each step. You can specify these things:
   - The table or query that contains the data you want to use in the form
   - The table fields you want to include in the form
   - The layout for fields on the form
   - Whether you want excess fields to spill over onto tabbed pages (using the TabBox component) or remain on the same page with a vertical scroll bar.
   - The colors and font for the elements on the form, including the form itself, push buttons, editing and nonediting components. You can select a preset scheme of colors and patterns or define your own and save it, making it available for future use.

The Form wizard generates the form you specify. At the end of the wizard, you have the choice of running the form or opening the form in design mode to further customize it (adding components, changing properties, writing event handlers, and so on).

Using the Form designer

To modify a wizard-created form or to design a form from scratch, use the Form designer (File | New | Form). These are the basic steps to follow in designing a form:

1. Place components on the form. To do so, drag files (including data modules, if you’re using them) from the Navigator or Windows Explorer to automatically link a table or database to a form, and drag the objects you need from the Component and Field palettes onto the design surface.

Note If you drag tables onto the form, the fields that are available on the Field palette are already linked to data. To link any other component to a field, set its *dataLink* property. see “Linking a form or report to tables” on page 6-5 for more information.

2. Set component properties, using the Inspector (or the Source editor, if you prefer).
3. Attach code to component events and write the methods you need.
4. Create menus, as necessary, using the Menu designer (see Chapter 8, “Creating menus and toolbars”).
5. Create toolbars and tool buttons, as necessary (see Chapter 8, “Creating menus and toolbars”).

The Form designer creates a .WFM file.

The components available in *dBASE SE* and the mechanics of using the Form designer, including the Inspector, are discussed in Chapter 7, "Using the Form and Report designers" Also see the samples that come with *dBASE SE*, installed by default in the dBASE\SE\Samples directory.

The following sections give you an orientation to the code generated by the Form designer.

.WFM file structure

The following code was generated by

1. Dragging a table from the Navigator table's page onto the Form design surface
2. Adding a pushbutton from the Component palette
3. Selecting the *onClick* event in the Inspector and clicking its tool button
4. Writing simple code for an event handler that tells how many rows are in the table when the form is run and the button is clicked.

Here is the code:

```
** END HEADER -- do not remove this line
*
* Generated on 08/24/00
*
parameter bModal
local f
f = new UntitledForm( )
if (bModal)
  f.mdi = .F. && ensure not MDI
  f.ReadModal( )
else
  f.Open( )
endif

CLASS AnatomyForm OF FORM
  this.Height = 12
  this.Left = 30
  this.Top = 0
  this.Width = 40
  this.Text = ""
  this.animals1 = new Query( )
  this.animals1.parent = this
  with (this.animals1)
    Left = 10
    Top = 4
    SQL = 'SELECT * FROM "C:\PROGRAM FILES\dBASE\SE\Samples\Animals.dbf"'
    Active = True
  endwith

this.pushbutton1 = new pushbutton(this)
with (this.pushbutton1)
    onClick = class::PUSHBUTTON1_ONCLICK
    Height = 1.1176
    Left = 20
    Top = 3
    Width = 15.875
    Text = "PUSHBUTTON1"
    Metric = 0
    FontBold = False
    Group = True
  endwith

this.Rowset = this.animals1.Rowset

// {Linked Method} Form.pushbutton1.onClick
function PUSHBUTTON1_onClick
this.text = form.rowset.count( )

ENDCLASS
```

There are four major sections in a .WFM file:

1. The first part is the optional Header section. This is any code above the ** END HEADER line. Comments that describe the file are usually put here.
2. Between the header and the beginning of the CLASS definition is the standard bootstrap code. This code instantiates and opens a form when you run the form, similar to the way the boot sector of a disk starts the

system when you turn on your computer. The standard bootstrap code allows you to open the form in two ways:

- If you DO the .WFM with no parameters, the form is opened with the open( ) method. The form is modeless.
- If you DO the .WFM with the parameter True, the form is opened with the *readModal*( ) method. The form is modal. A modal form cannot be MDI, so the form's MDI property is set to False first.

3. The main CLASS definition constitutes the bulk of most .WFM files. This is the code representation of forms designed visually in the Form designer.
4. Everything after the main class definition, if anything, makes up the General section. This is a place for other functions and classes.

Form class definition

Like any other CLASS definition, the main one in the .WFM file can be further broken down into two parts:

1. The constructor is the code that is run every time a NEW object of that class is instantiated. It creates, or constructs, an object of that class. Class constructors created by the Form designer are divided into four parts:
   - Assignments to the stock properties of the Form object.
   - data objects in the form, each with its own WITH block.
   - All the controls in the form, each with its own WITH block.
   - Housekeeping code; specifically to assign the rowset of one of the queries in the form to the form's *rowset* property as the form's primary rowset.
2. Class methods, if any, follow. This is usually event handler code, but can also contain other methods that pertain to the form and which often are called by the event handlers.

How the contents are generated

The contents of the class constructor reflect the work you've done in the visual development environment. You can create and edit class methods in the Source editor. Both the Header and General sections are also editable in the Source editor. You have no control over the bootstrap code generated by the Form designer.

Editing a .WFM file

As you become more proficient in *dBASE SE*, you will find that it is sometimes more convenient to edit a form directly in source code without opening the form in the Form designer. To edit the form file directly, right-click the .WFM file in the Project Explorer or Navigator and choose Open In Source Editor. This will open the .WFM file in the built-in Source editor or another programmer's editor you specified in the Desktop Properties dialog box.

One advantage to using the built-in Source editor is that you can run the .WFM file directly by pressing the F2 key. No matter which editor you choose, you must save and close the .WFM file if you want to edit the form in the Form designer.

When editing the .WFM file directly, you want to preserve the two-way nature of the Form designer so that any changes you make manually will not be lost the next time you save the form from the Form designer.

Editing the header and bootstrap

The first "safe .WFM" rule involves the line that says:

```
** END HEADER -- do not remove this line
```

Don't remove or modify it! If you do, you might lose the contents of the header or prevent the Form designer from being able to read the form from the .WFM file.

The next rule is about the standard bootstrap code: don't bother changing it. Every time the .WFM file is written the same standard bootstrap is rewritten anew, so any changes you make will be lost.

If you want to change the way the form is instantiated and opened when you run the form, instead of changing the bootstrap code, you need to add to it or replace it by placing your own bootstrap code in the header.

The key is to realize that a .WFM file is just a program file with a different extension. When you run the form, the code at the top is run just like when you run a program. To put it another way, there is nothing magical about the standard bootstrap code—it just happens to be the first code that is found at the top of a plain .WFM file. If there are some comments in the header they have no effect.

You can place any code you want in the header. The Form designer will ignore it.

Editing properties in the .WFM file

Inside a WITH block, you may assign values to existing properties only. Therefore, you are free to edit the values assigned to any of the properties in the class constructor, or add assignments to the objects' stock properties.

Most properties must be of a particular data type. For example, *pageNo* is a number and *sql* is a string. If you change the property, you must maintain the correct type.

One notable exception is the *value* property. If a component is *dataLink*ed to a field, the type of that field determines the type of the *value*. But if the component is not *dataLink*ed, its type can be any of the simple data types. In the Inspector, you can use the Type button to select the type of the value you're assigning to the property if the property can accept multiple types.

The Form designer leans toward literals as opposed to expressions. For example, suppose you want a Text component to default to the current date. You could edit the .WFM file so that the assignment reads:

```
value = date( )
```

That would work fine until the next time you edit the form in the Form designer. The expression gets evaluated when the form is loaded so that the *value* property has an actual date. Then that date gets saved to the .WFM file which causes the date that you last edited the form to be hard-coded into the form.

The simplest way to solve the problem is to set the *value* property programmatically, which puts it outside the reach of the Form designer. The most convenient place is the component's *onOpen* event. A simple codeblock like this will do it:

```
{;this.value = date( )}
```

When the form is run, the form's *onOpen* event and each component's *onOpen* event, if any, is called in turn. This codeblock updates the *value* to the current date. The Form designer knows that a codeblock is attached to the *onOpen* event, and reads and writes it, but it doesn't bother with what's inside it, and doesn't change it.

Types of form windows

dBASE SE lets you create windows that are standard features of the Windows environment:

- MDI windows
- SDI windows
- Modal windows
- Modeless windows

The following sections briefly describe these form types.

MDI and SDI applications

You can create windows that conform to the Windows Multiple Document Interface (MDI). MDI is a Windows standard that allows an application to manage multiple windows or multiple views of the same document within the main application window. In *dBASE SE*, for example, you can have multiple windows (Command window, Navigator, Table designer, and so on) open at the same time. You can also open the same document (form, table, report) multiple times.

You can also create Single Document Interface (SDI) windows with *dBASE SE*. Unlike an MDI window, an SDI window does not contain any child windows.

MDI windows are the most appropriate for data entry forms. Forms that you create with the Form designer are MDI windows by default.

Modal and modeless windows

dBASE SE lets you create both modal and modeless windows.

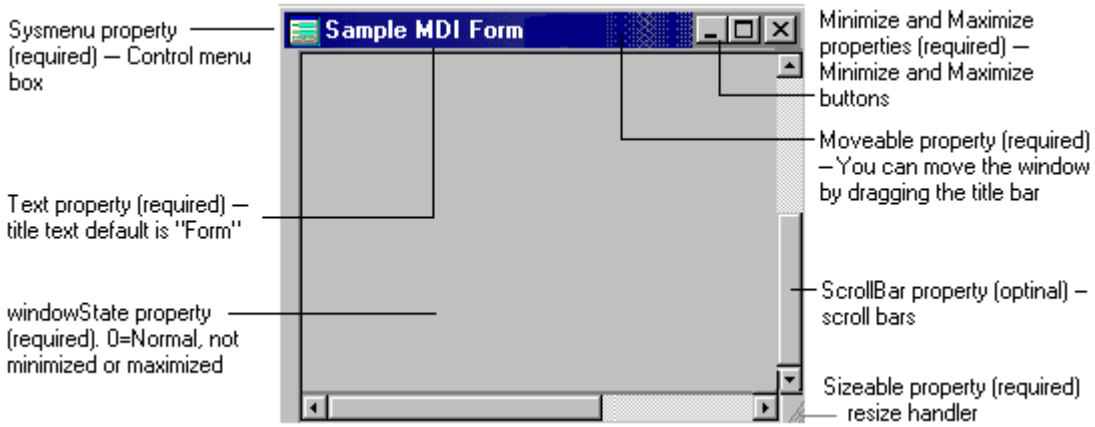
A modeless window does not take control of the user interface. A user can switch between modeless windows while an application is running. For example, the various windows that appear in the *dBASE SE* Form designer, such as the Control palette, the Field palette, and the Inspector, are modeless windows.

A modal window takes control of the user interface. A user cannot switch to another window without exiting a modal window. A dialog box is an example of a modal window; when it's open, users cannot take any other actions outside the dialog box until it is closed. Modal forms are most appropriate as dialog boxes in applications.

Customizing the MDI form window

The following sample MDI window shows required and optional window properties you can set for your form.

Figure 5.2 Sample MDI window



Standard features of MDI windows:

- They are moveable and sizeable.
- They have a window title, a Control-menu box, Maximize and Minimize buttons.
- When active, their menus replace the menus in the application menu bar.
- They are bounded by the MDI parent window's frame.

If the MDI property is set to *true*, those features are automatically applied to the form. Accordingly, the following form properties are automatically set to *true*: *Minimize*, *Maximize*, *Sizeable*, *Moveable*, and *SysMenu*. Changing the MDI-required properties will have no effect until you change the MDI property itself to *false*. For more information about any of these properties, press F1 when the property in the Inspector is highlighted.

Using multi-page forms

If your form needs to contain many fields or controls, you'll want to use a Notebook component or a multi-page form. Using either one, you can divide controls into related groups, with each group presented on a separate page.

It is easy to create forms with several pages and navigation buttons.

When you create a new form, the Form designer opens it on the first page. To create a multi-page form, choose the Next Form Page button on the toolbar. The Form designer appends a page each time you click the button.

To navigate between pages in the Form designer, use any of these techniques:

- Use the Next Form Page and Previous Form Page toolbar buttons.
- Choose View | Previous Form Page or View | Next Form Page.
- Use the PgUp and PgDn keys.
- In the Inspector, select the form object in the top selection box, and on the Properties page, change the numeric value of the *pageNo* property. Notice that as you change this value, the pages of the form change on the design surface.

Global page (forms)

In a multi-page form, page 0 is a "global" page. Controls you place on page 0 are visible on every page of the form.

To open page 0,

1. Select the form object in the Inspector's top selection box.
2. On the Properties page, change the numeric value of the *pageNo* property to 0.

Page 0 displays a composite view of all controls from all pages to help you position global controls so they will not interfere on the other pages. If you have several pages, naturally the various components of those pages may overlap in this composite view. To reposition the controls on other pages, you must navigate to the appropriate page.

Important When you save a multi-page form, the page that is active becomes the default page at run time. Therefore, make sure you return to page 1 before clicking Run.

Navigation buttons (form pages)

If you create a multi-page form, you will probably want to provide buttons to enable users to navigate between form pages. A simple solution is to create buttons at the top of the global page (*pageNo*=0) of the multi-page form.

To create one set of navigation buttons for a multi-page form,

1. Go to the global page, page 0 of the form (View | Go To Form Page Number).
2. Select the form itself in the Inspector's top selection box, and make sure the *pageNo* property is 0.
3. From the Component palette drag as many button components as you need to the visual design surface. Ensure that the buttons will not overlap controls appearing on other pages.
4. Select each button and set its *pageNo* property to 0 (the global page) so that it will appear on all pages. (Or multi-select the buttons and set the property once.)
5. Select each button's *text* property and change its value to whatever you want on the button, for example Next Page or Previous Page.
6. For each button, on its Events page, select *onClick* and click the tool button to display the Source editor. You'll see a comment for an *onClick* method. Write the code that will send the user to the appropriate page. Return to page 1 before running the form.

Creating a custom form, report, or data module class

When you use the designers in *dBASE SE*, by default the Form designer uses the Form class, the Report designer uses the Report class, and the dQuery/Web dataModule Designer uses the DataModule class.

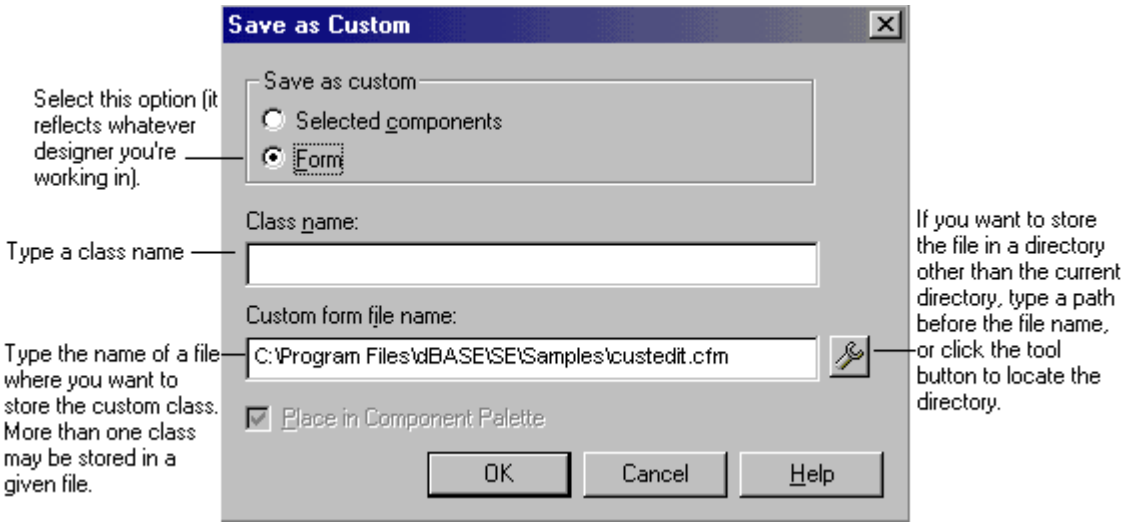
However, you can create *custom classes* and use them as templates (both for new forms, reports, and data modules and those already created). For example, if you want many forms in your application to have a similar

look, you can specify all the common attributes for those forms, such as colors, size, controls, event handlers, and so forth, once. When you have established all the common attributes, save that form as a custom form class. Then you can specify that class to be a template for forms. Changes you subsequently make to the custom form class will be reflected in all its derived forms.

To create a custom form, report, or data module class,

1. Use the appropriate designer to create the form, report, or data module template you want.
2. Choose File | Save As Custom to display the Save As Custom dialog box.
3. Choose Save Form (or Report or Data Module) As Custom, then complete the rest of the dialog box as described in Figure 5.3.

Figure 5.3 Saving a custom class



The new custom class file will be saved with the .CFM (custom form) extension if it's a form or .CRP (custom report) extension if it's a report or .CDM extension if it's a data module. Custom classes for forms, reports, and data modules are available from their respective pages in the Navigator. Their icons are yellow.

An alternate way to create a custom class is to double-click the yellow, "Untitled" icon on the Forms, Reports, or Data Modules page of the Navigator. This opens the appropriate Custom Class designer, which is almost identical to the Form or Report designers. Then add the common features you want to appear on all derived forms, reports, or data modules.

Note You can't run a file you've developed in this way; it is simply a template from which other forms, reports, or data modules can be derived.

Using a custom class

To use a custom form, report, or data module class,

1. Open a new or existing form or data module to which you want to apply a custom class.
   - Setting a custom report always causes a new report to be created. To apply a custom class to an existing report, open the report in the source code editor and change the CLASS statement:

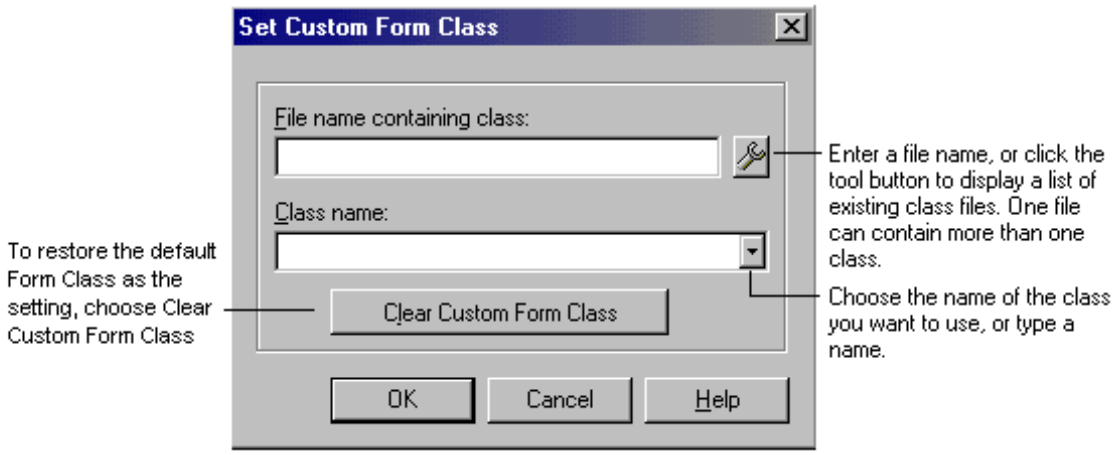
     ```
     CLASS MyReport OF Report
     ```

     to read:

     ```
     CLASS MyReport OF "MyCustomReport" FROM "MyCustomRep.CRP"
     ```
2. From the designer, choose File | Set Custom Form (or Report or Data Module) Class.

3 Complete the Set Custom Class dialog box and choose OK.

Figure 5.4 Set Custom form Class Dialog Box



Your custom class now applies to the current file in the designer. In addition, subsequent new files of that type will use the current setting in the Set Custom Class dialog box. To change this, choose File | Set Custom Class, and either enter a new form or report custom class, or choose the Clear Custom Class button to restore the default class as the setting.

Creating custom components

You can create your own customized components and add them to the Component palette for easy reuse. A custom component is based on one or more of the components already on the Component palette. You arrange these components, as you want them in the Form or Report designer, and set their properties, event handlers, and methods, as you desire.

Then you save your work into a custom component file (with the .CC extension) and optionally add it to the Component palette for convenient access.

Creating custom components

To create custom components that you can use again,

1 Drag a component or components to the Form or Report design surface, and arrange them the way you want them.

2 Set each component's properties, events, and methods.

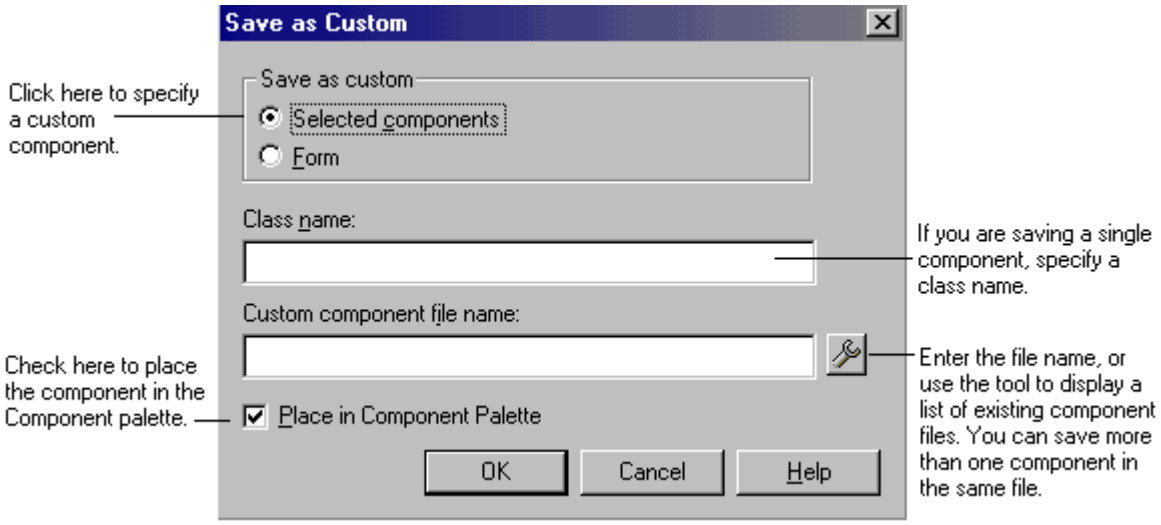
3 Select the component or group of components.

4 Choose File | Save As Custom to display the Save As Custom dialog box, then complete the dialog box:

- Type a class name for the customized component.
- Specify an entire path name and the file (with the .CC extension) in which you want to store this component. Note that the components in a .CC file are treated as a group, and although you can add them to the palette individually in this dialog box (by checking the appropriate check box), you can remove them from the palette only as a group. So, if you want to be able to add and remove custom components from the palette separately, put each in its own file.
- Check the Place In Component Palette check box, if you want this component to appear on the Component palette. If you are putting the component in an existing .CC file whose components are already on the

palette, and you don't check this box, then later, if you want to add the component to the palette, you'll have to remove the .CC file from the Set Up Custom Components dialog box, and then add it anew.

Figure 5.5 Save as Custom dialog box; saving Custom Components



5 Click OK.

The custom component is now stored in the .CC file you specified. You can open the file in the Source editor.

Adding custom components to the Component palette

If you have designed a custom component yourself, the simplest way to add it to the Component palette is to check the Place In Component Palette check box in the Save As Custom dialog box (File | Save As Custom) at the time you are saving your custom component. If you didn't do this, or if you have a custom component from someone else, here's what to do:

1 Choose File | Set Up Custom Components (or right-click in the Component palette for access to the same command).

2 The Set Up Custom Components dialog box appears. It lists paths to custom component files whose components already appear on the Component palette.

3 Choose Add to open the Choose Custom Component dialog box.

4 In the Choose Custom Component dialog box, locate the custom component file (with the .CC extension) that contains the component you want to put on the Component palette. Choose Open.

5 The path name to the selected custom component file now appears in the Set Up Custom Components dialog box.

6 With the desired .CC file selected, choose OK. The custom components you have saved in the .CC file appear on the Custom page of your Component palette (in both the Form and Report designers), ready to use just like any other component.

Removing custom components from the Component palette

To remove a custom component from the Component palette,

1 Choose File | Set Up Custom Components.

2 In the dialog box that appears, select the file that contains the custom component, and choose Delete.

All the custom components in that file are removed from the Component palette. The .CC file is not deleted from disk.

Chapter

6

Accessing and linking tables

To link your forms and reports to the data in tables, *dBASE SE* provides a set of data objects. In the designers, these objects are available on the Data Access page of the Component palette. These components make specialized database access functionality available to your *dBASE SE* applications.

This section discusses the following topics:

- The *dBASE SE* data model
- Linking a form or report to tables
- Creating master-detail relationships
- Creating and using a dataModule

Before you use the data objects, you should understand the *dBASE SE* data model, described in the next section.

Note Although the old dBASE Data Manipulation Language (DML) still exists for backward compatibility, those methods are no longer recommended. The new data object model is recommended because it utilizes the full power of object-oriented programming.

The dBASE data model

dBASE SE's advanced, event-driven data model is implemented entirely in a handful of classes:

- Session
- Database
- Query
- StoredProc
- Rowset
- Field

This section gives you a sense of how these classes fit together. It introduces each object and explains how its primary properties relate to the other objects.

Query objects

Query objects are the center of the data model. In most cases, if you want to access a table, you must use a Query object.

Note Alternatively, you could use a StoredProc object that returns a rowset from an SQL database, or a DataModRef object that points to a data module containing the appropriate data access code, including at least a Query or StoredProc object.

The Query object's main job is to house two important properties: SQL and *rowset*.

SQL property

The *SQL* property's value is an SQL statement that describes the data to be obtained from the table. For example,

```
select * from BIOLIFE
```

The * means all the fields and BIOLIFE is the name of the table, so that statement would get all the fields from the BIOLIFE table.

The SQL statement specifies which tables to access, any tables to join, which fields to return, the sort order, and so on. This information is what many people think of when they hear the word query, but in *dBASE SE*, SQL statements are only one of many properties of the Query object.

SQL is a standard, portable language designed to be used in other language products to access databases. When you use the Form and Report wizards or drag a table from the *dBASE SE* Navigator, *dBASE SE* builds the SQL statement for you. Once a table has been accessed by the SQL statement, you can do almost anything you want with *dBASE SE*'s data objects, including navigating, searching, editing, adding, and deleting.

Although knowing SQL is useful for initially configuring data objects for your databases, once these are complete and saved as custom components or in data modules, they can be reused without modification. Then others can create complete Windows database applications without knowing a word of SQL.

rowset property

A Query object is activated when its *active* property is set to true. When this happens, the SQL statement in the *sql* property is executed. The SQL statement generates a result: a set of rows, or rowset.

A rowset represents some or all the rows of a table or group of related tables.

Each Query object generates only one rowset, but you can add multiple Query objects to a form to use multiple rowsets from the same table, or from different tables. Using multiple Query objects also allows you to take advantage of *dBASE SE*'s built-in master-detail linking. See "Creating master-detail relationships (overview)" on page 6-7

The Query object's *rowset* property refers to the Rowset object that represents the query's results.

Rowset objects

While you must use a Query object to get access to data, you must use the Query object's resulting rowset to do anything with the data. All navigation methods for getting around in tables depend on the query's rowset.

The row cursor and navigation

The rowset maintains a *row cursor* that points to the current row in the rowset. When the Query object is first activated, the row cursor points to the first row in the rowset.

Synchronizing cursor movement in master-detail rowsets

Enabling a linked-detail rowset's *navigateMaster* and *navigateByMaster* properties allows master-detail rowsets to be navigated as though they were part of a single, combined rowset (similar to the xDML SET SKIP command).

Note Using these properties will modify the behavior of the *first*( ), *next*( ), *last*( ), *atfirst*( ) and *atlast*( ) methods. For more information, see Help and choose, *navigateByMaster*.

You can get and store the cursor's current position by calling the rowset's *bookmark*( ) method.

To move the row cursor, call the rowset's navigation methods:

- *next*( ) moves the cursor a specified number of rows relative to its current position.
- *first*( ) goes to the first row in the rowset.
- *last*( ) moves to the last row.

- *goto*( ) uses the value returned by *bookmark*( ) to move back to that specific row.

Because each rowset maintains its own row cursor, you can open multiple queries—each of which has its own rowset—to access the same table and point to different rows simultaneously.

Master-detail rowset synchronization can be overridden by using the _app object's *detailNavigationOverride* property. For more information on these properties, see Help.

Rowset modes

Once a Query object has been activated, its rowset is always in one of the following five modes (indicated by the rowset's *state* property):

- Browse mode, which allows navigation only.
- Edit mode, the default, which allows changes to the row.
- Append mode, in which the user can type values for a new row, and if the row is saved, a new row is created on disk.
- Filter mode, used to implement Filter-By-Form, in which the user types values into the form and *dBASE SE* filters out all the rows that do not match.
- Locate mode, similar to Filter mode, except that it searches only for the first match, instead of setting a filter.

Rowset events

A rowset has many events used to control and augment its methods. These events fall into two categories:

- can- events, so named because they all start with the word can—which are fired before the desired action to see whether an action is allowed to occur; and
- on- events, which fire after the action has successfully occurred.

Row buffer

The rowset maintains a buffer for the current row. It contains all the values for all the fields in that row.

You access the buffer by using the rowset's *fields* property, which refers to an array of Field objects.

Field objects

The rowset's *fields* array contains a Field object for each field in the row. In addition to static information, such as the field's name and size, the most important property of a Field object is its *value*.

value property

A Field object's *value* property reflects the value of that field for the current row. It is automatically updated as the rowset's row cursor is moved from row to row.

To change the value in the row buffer, assign a value to the *value* property and set the rowset's *modified* property to "true". This signals the rowset that values have been changed. If the row is saved, those changes are written to disk.

Important When referring to the contents of a field, don't forget to use the *value* property. For example,

```
this.form.rowset.fields[ "Species" ].value
```

If you leave out *value*,

```
this.form.rowset.fields[ "Species" ]
```

you are referring to the Field object itself, which is rarely intentional—except for *dataLink*s, explained next. Get in the habit of including *value* when referring to a field; if you don't, the code doesn't work.

Using *dataLink*s

Just as a Field object's *value* property is linked to the actual value in a table, a visual object on the form (such as an EntryField or RadioButton) can be linked to a field object through the visual object's *dataLink* property. This property is assigned a reference to the linked Field object. When connected in this way, the two objects are referred to as *dataLink*ed.

As the rowset navigates from row to row, the Field object's *value* is updated, which in turn updates the component on the form. If a value is changed in the form component, it is reflected in the *dataLink*ed Field object. From there, the change is saved to the table.

Database objects

Database objects are one level up from Query objects in the object hierarchy. Database objects have three main functions:

- To access a database
- Database-level security
- Database-level methods

Accessing a database

A Database object is needed to access SQL databases, ODBC databases, and any other tables you are accessing through a BDE alias.

Before you can use a Database object, you must set up BDE to access the database by using the BDE Administrator (available from the *dBASE SE* program group). See "How to connect to an SQL database server" on page 2-3.

To connect a Database object to a database, set the Database object's *databaseName* property to the BDE alias for the database.

Database-level security

Many SQL and ODBC databases require the user to log in to the database. You can preset the Database object's *loginString* property with a valid user name and password to log in to the database automatically.

Because each Database object represents access to a database, you can have multiple Database objects that are logged in as different users to the same database.

Database-level methods

The Database object contains methods to perform database-level operations such as transaction logging and rollback, table copying, and re-indexing. Different database formats support each method to varying degrees. Before accessing the methods of a Database object, the Database object itself must be active. The methods of a Database object will not function properly when it's *active* property is set to "false".

Default Database object

To provide direct, built-in access to the BDE-standard table types (dBASE and Paradox), each session includes a default Database object that does not have a BDE alias. When you create a Query object, it is initially assigned to the default Database object. Thus, if you're accessing dBASE or Paradox tables without an alias, you don't need to use a Database object.

If you're accessing other table types, you need to use the Database object. See "Linking to a table manually" on page 6-6.

Session objects

At the top of the object hierarchy is the Session object. Each session represents a separate user.

Each session contains one or more Database objects. A session always contains at least the default Database object, which supports direct access of dBASE and Paradox tables.

Session objects are important for dBASE and Paradox table security. Multiple users each have their own session, so that different users can be logged in with different levels of access, or they may share a single session, so that all users have the same level of access. For the Session object's security features to work, the *session* property of an active database object must be set to the session object.

A default Session object always exists whenever you run *dBASE SE* (either the environment or an application, sometimes referred to as a *dBASE SE* executable). In most cases, the default Session is all you need. There is usually no need to add a Session component to your forms or reports. *dBASE SE*'s App object has a property that points to the default session object and the default database object. Thus, when you create a Query object, it is automatically assigned to both the default Session object and the default Database object.

The Session object has an event called *onProgress* that you can use to display progress information on database operations.

StoredProc objects

The StoredProc object is used for calling a stored procedure in SQL databases. When you're calling a stored procedure, the StoredProc object takes the place of the Query object in the class hierarchy; it is attached to a Database object that gives access to the SQL database, and it can result in a Rowset object that contains Field objects.

The stored procedure can:

- Return values, which are read from the *params* array
- Return a rowset, which is accessed through the *rowset* property, if the server supports this capability

DataModRef objects

The DataModRef object points to preprogrammed data access components stored in a dataModule. If you maintain data access code in a dataModule, then you can use a DataModRef object to return rowsets in place of a Query or StoredProc component.

Data modules offer convenient reusability and easy maintenance of data access code. By storing custom or preset data access components in a dataModule, it is easy to maintain them (change links to changing databases, for example). Then, you can use just the DataModRef component (or custom class) to instantly implement the full set of current data access components.

To set a DataModRef object to point to a dataModule, set its *filename* property to the path name of the data module.

Note The DataModRef object is maintained for backward compatibility. Enhancements to the DataModule class make it a more desirable method of storing data objects.

Linking a form or report to tables

The Query object links a form or report to a table, making the table's fields available to the controls on the form or report. One Query object can refer to multiple tables in its SQL statement, or you can use multiple Query objects with an appropriate query statement in each.

If you need to link to table data in an SQL or ODBC database, you must first assign the database a BDE alias in the BDE Administrator. If you haven't done this yet, see "How to connect to an SQL database server" on page 2-3. Then, to see your tables listed in the Navigator:

1. Click the Navigator's Tables tab.
2. From the Look In drop-down list, select the alias of the database you want to access. Tables from the selected database appear listed on the Navigator Tables page.

If you are linking to BDE-standard tables, use the Navigator Look In drop-down list to select the directory that contains your tables. (Click the Tables tab to see the tables listed.)

From there, you can link to a table in two ways:

1. Automatically, by dragging from the Navigator or using a wizard
2. By dragging data access components from the Component palette to the design surface and setting linking properties

Linking to a table automatically

The easiest way to use table data in a form or report is to drag the table from the Navigator onto the form or report design surface.

- For BDE-standard tables that you're accessing without a BDE alias, this creates a Query object.
- For SQL, ODBC, and other tables you're accessing through a BDE alias, this automatically creates both the Database object, which is required to connect to the database, and the Query object for the table.

The SQL and *rowset* properties of the Query object, and the *dataBaseName* property of the Database object are both set automatically, and the *active* property of both objects is set to *true*. The link is complete, and fields of the table are available from the Field palette.

The SQL statement in the SQL property selects all the fields of the table. You can modify this statement in the Inspector. Click the tool beside the SQL property.

Linking to a table manually

Instead of dragging a table from the Navigator, you can use data objects from the Component palette.

For SQL, ODBC, and other tables you're accessing through a BDE alias,

1. Drag a Database object from the Component palette to the form or report design surface. (One Query object is added along with it.)
   - Assign the BDE alias to the *databaseName* property.
   - Set its *active* property to *true*.
2. For databases that require a login, you must either log in or set the Database object's *loginString* property, so that the table will open without requiring a password or ID to be entered. (Your login name and password must be set up by your database administrator.)
3. Select the Query object.
   - Type the SQL query statement you want in the Query object's SQL property. Your SQL query can access any number of tables in the database. Some servers are case-sensitive for the table name; some may require quotation marks (Oracle, for example).
   - Assign the Database object to the Query object's *database* property. This must be done before activating the query.
   - Set the Query object's *active* property to *true*.
4. Add additional Query objects, if needed for other tables, and set their properties as in step 2. (If you want to drag a table from another database, be sure to first select the desired alias from the Navigator's Look In drop-down list, or in the case of BDE-standard tables without an alias, use the Navigator to locate the desired directory.)

For BDE-standard tables without a BDE alias, you do not need the Database object. Use only Query objects, and follow the instructions in steps 2 and 3.

To use tables accessed through a BDE alias, you must create new Database objects. Provided that you have created the BDE alias for your database, you need only activate the database object (and login if required) to have access to that database's tables. You may also log transactions or buffer updates to each database to allow you to rollback, abandon, or post changes.

Note A table's fields do not appear on the Field palette until the Query object's *active* property is set to *true*.

Procedure for using a Session object

All database applications are automatically provided with a Session object that encapsulates the default BDE session. You can create, and manipulate additional session components as needed.

If you intend to add another Session object, follow the sequence in this procedure for adding data objects to the design surface:

1. Add the Session object.
2. Add a Database object to your form (if accessing tables through a BDE alias). It is automatically linked to the Session object already on the form.
3. Set the Database object's *databaseName* property to the name of the BDE alias, and set its *active* property to *true*.
4. Add a Query object. It is automatically linked to the Session and Database objects already on the form or report.
5. Set the Query object's SQL property, then set its *active* property to *true*.

Calling a stored procedure

When you want to call a stored procedure, use the StoredProc object. When a stored procedure returns a rowset, it can take the place of a Query object.

To call a stored procedure,

1. Drag a StoredProc object from the Component palette onto the design surface.
2. Set its *procedureName* property to the name of the stored procedure.
3. Set any parameters that are passed to the stored procedure in the *params* array.
4. Set its *active* property to *true*.

Using local and remote tables together

If you use both local dBASE or Paradox tables as well as client/server databases, it's a good idea to create a BDE alias for the local dBASE or Paradox table directories and any other directories containing tables as well. There are two reasons for this:

1. All your table connections will be listed in the *dBASE SE* Navigator Look In box when the Tables tab is selected.
2. Using a BDE alias for BDE-standard tables makes it easier to move them to another directory; only the alias in the BDE configuration need be updated, and not the source code for all the forms and reports.

Creating master-detail relationships (overview)

A master-detail form or report displays information selected from one or more related tables in a relational database. It groups the detail rows from the detail tables in relation to an associated row from the master table.

In a relational database, a master table can be linked to one or more related (detail) tables by key fields. A detail table may in turn act as a master table, with other key fields linked to other detail tables. Each detail table contains a *masterRowset* property pointing to its master table. You can implement a master-detail relationship between tables by setting this property in the detail tables.

A typical example is a CUSTOMER table with a key field called Orders. You could link it to an ORDERS table by setting the ORDERS table's *masterRowset* property to the master CUSTOMER table. You could then generate a report on a selection of customers (from the master table CUSTOMER) that lists the rows of each customer's orders (from the detail table ORDERS). The result groups each customer's orders with each customer's name.

By creating a master-detail relationship and adding SQL statements to the Rowset or Query object properties, you can create forms and reports that group detail rows from detail tables with a selection of rows from the master table. For example, a report could include a filter on the ORDERS rowset to display a customer's orders only for the month of March. You can create complex filtered joins and perform virtually any programmatic operation on a database.

This section includes three different procedures to link master and detail tables:

1. Use an SQL JOIN statement to generate a rowset from two or more tables. This procedure is often the fastest and easiest. It is illustrated in the AIRCRAFT.REP report in the SAMPLES directory.
2. For local BDE-standard tables, use the Rowset object's *masterRowset* and *masterFields* properties.
3. For client/server databases, use the Query object's *masterSource* property. (You can also use this in local tables.)

In general, for any procedure, you begin with these steps:

1. Make sure each pair of tables is indexed on a common field.
2. Drag the tables from the Navigator to the visual design surface of the designer you're working in. This creates a Query object for each table.

Using an SQL JOIN statement

The sample report FLIGHT.REP (located in your *dBASE SE* SAMPLES\FLEET directory) uses a single Query object whose SQL property contains an SQL JOIN statement linking the master table AIRCRAFT.DBF and the detail table FLIGHT.DBF. Both tables are indexed on a common field.

In the resulting report, each aircraft (stored in the master AIRCRAFT.DBF table) is displayed in the headerBand, followed by a list of flights for that aircraft (stored in the detail FLIGHT.DBF table) in the DetailBand.

This technique is usually faster and easier than adding and linking two Query objects. (However, in some cases, with BLOB fields, for example, it might be faster to use two Query objects.) You should also be aware that rowsets resulting from an SQL JOIN statement are read-only, and therefore cannot be edited.

By using two joined tables, you gain several advantages:

- You can use the data as if it were all in one table.
- Separately the tables can be more easily maintained.
- If you have bitmaps, you need store them in only the master table, rather than duplicating the image in every row of the detail table.
- This method does not require an index (although with indexes it is much faster).

To create a master-detail relationship by using an SQL JOIN statement,

1. Add a Query object to the design surface.
2. Select the Query object. In the Inspector, click the wrench tool beside the SQL property to display the SQL Property Builder.
3. Do one of the following:
   - Write an SQL JOIN query in the SQL Property Builder
   - Locate a query you've already written

Note Including image fields slows performance.

1. If you're designing a report, after the SQL property is set, choose Layout | Add Groups And Summaries. In the Groups And Summaries dialog box, all the fields from both tables appear in the Available Fields pane.
2. Select the field on which you want to group the detail rows.

Linking master-detail in local tables

Creating a master-detail relationship by using the properties *masterRowset* and *masterFields* is the most efficient technique when working with local .DBF tables. It is similar to the older technique of using the SET RELATION and SET INDEX commands, but that technique is no longer recommended.

To link local master-detail tables,

1. Drag the two tables onto the design surface of the designer you're working in.
2. Select the Query object of the detail table and set its *masterRowset* property to the name of the master table's Query object. To do this, select the name of the Query object from the property's drop-down list (the down-arrow button, not the tool button).
3. With the Query object of the detail table still selected, click the rowset property's tool button to display the rowset properties.
4. Click the rowset's *masterFields* property and from the drop-down list select the fields you want to link from the master table.
5. Set the *indexName* property to the same field as the *masterFields* property. If the field names between the two tables are not identical, then in the *indexName* property select the index that corresponds to the *masterFields* setting.
6. If you're designing a report, choose Layout | Add Groups And Summaries, and in the dialog box group the detail fields under the appropriate master-table field.

Using the *masterSource* property

By using the *masterSource* property to create master-detail relationships you do not need an index, although it would improve performance. You might choose to use the *masterSource* property when

- You can improve performance, for example, in cases where large BLOB fields would be copied to temporary files
- You are working with client/server databases
- You are working with a one-to-many relation in a form, and you want the form to be updateable
- You want the order of the "many" table to be different from that of the linked fields.

To create a master-detail relationship by using the *masterSource* property,

1. Drag the two tables onto the design surface of the designer you're working in.
2. Select the Query object of the detail table and set its *active* property to false.
3. Change the SQL statement in the Query object's SQL property to use host variables. For example, in a master-detail report on CUSTOMERS and ORDERS, you might use this:

   ```
   SELECT * FROM ORDERS WHERE ORDERNO = :ORDERNO
   ```

   assuming that ORDERNO is the exact field name in the table.
4. Set the detail table's Query object's *masterSource* property to the name of the master table's Query object. To do this, select the name of the Query object from the property's drop-down list (the down-arrow button, not the tool button).
5. Set the detail table's Query object's *active* property back to true.
6. This creates a parameter using the key fields from the master table.

What is a dataModule?

A dataModule is a *dBASE SE* class for centralized handling of data access objects (Query, Database, StoredProc, and Session). A dataModule enables you to:

- Place all your data access objects in a single container instead of duplicating them on each application form.
- Design queries once for use with many forms and reports instead of recreating them separately for each one.
- Create business rules—using object events, and additional methods you add to the source code for a dataModule—that can be shared across an entire application.
- Separate business logic and data access from user interface code for easier maintenance.

After you've set up data access objects in a dataModule, it's easy to maintain them (change links to changing databases, for example).

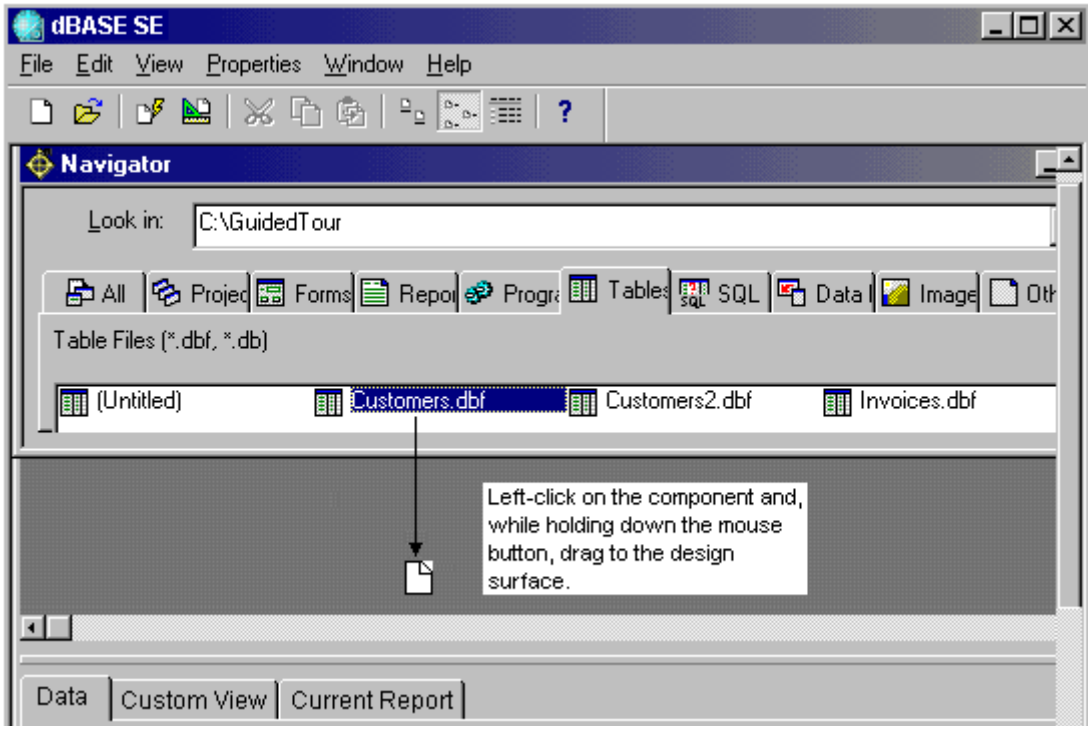
Creating a dataModule

To create a dataModule,

1. Open the dQuery/Web dataModule designer by choosing File | New | Data Module, and select Designer from the New Data Module window.
2. From the Navigator, drag the components you need onto the design surface, set their properties (SQL and so on), and write their event handlers.

   Press F12 to toggle between the Source editor and the visual designer.
3. When everything is set up as you want it, save the dataModule (File | Save). It is saved with a .DMD extension.

Figure 6.1 Drag & Drop



Creating business rules in a dataModule

Besides writing event handlers for the components in a dataModule, you can code methods directly in the source file for a dataModule. These methods can be applied to the forms that use the dataModule as business rules. For example, you might write a procedure to perform month-, quarter-, or year-end bookkeeping. You might call the procedure from an event handler for a component in the dataModule.

Using a dataModule

To use a dataModule in a form or report, do one of the following:

- Add a DataModRef object from the Component palette and link it to the desired dataModule file by setting the DataModRef's *filename* property to the path and filename of the dataModule.
- Drag the dataModule file from the Navigator or Project Explorer to a form or report design surface. This adds a dataModule object to the form.

The properties, event handlers, and methods you set for components in a dataModule apply consistently to all forms and reports that use the module. For a demonstration of how to create and use dataModules, see "Opening, Saving and Creating a New dataModule" on page 3-4.

Chapter

7

Using the Form and Report designers

This section shows you the common elements you have to work with in the Form and Report designers. Other designers—for data modules, labels, and custom classes—are variations on the Form and Report designers. Their menus and tools vary, and they might look a little different, but otherwise they all work basically the same. This section refers to the Form and Report designers, but the information applies to the other designers, as well. The section includes the following:

- A description of the Form and Report designer windows
- What's available on the Component palette for use in your forms and reports (this is an overview in table form; see Help for more detailed information on how to use specific components)
- A discussion of the Field palette and how to populate it with components linked to fields in a table
- How to change component properties and create event handlers and other methods by using the Inspector
- How to manipulate components (change alignment, spacing, formatting, and so on)

You can open any of the designers from the File menu (File | New) or from the Navigator or Project Manager.

Note The yellow untitled icon on the Form and Report pages of the Navigator is for creating a custom class that you can use as a template.

The designer windows

The form and report windows are visual design surfaces on which you position the components you need for your application. These can be invisible components, like data objects (queries, stored procedures, databases, sessions, and data module references) and visible components, like text, graphics, list boxes, check boxes, and so on.

In both designers, the work you do with the visual design elements is reflected in the underlying code and vice versa. Press F12 to switch between the design surface and code.

You can change the size and position of a designer window either by dragging the edges of the window or, if you need to be precise, by changing the values of height, left, top, and width in the Inspector.

By default, a grid appears when you start the Form and Report designers, and objects are constrained to line up along the grids (Snap To Grid). In addition, vertical and horizontal rulers appear.

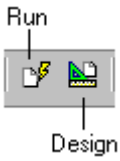
Both designers have the following tools:

- Component palette for dragging user-interface elements and data-access objects to the design surface
- Field palette for dragging linked fields to the design surface
- Inspector for setting properties and writing event handlers and other methods

- Format toolbar for formatting Text objects
- Alignment toolbar for aligning objects
- Layout, Format, and Method menus
- Status bar to show you your location on the design surface, show you what object is selected, and to give you instructions and other information

To display a tool window that's not open, choose View | Tool Windows.

Design and Run modes



You can view forms and reports (and other files) in Design mode and Run mode.

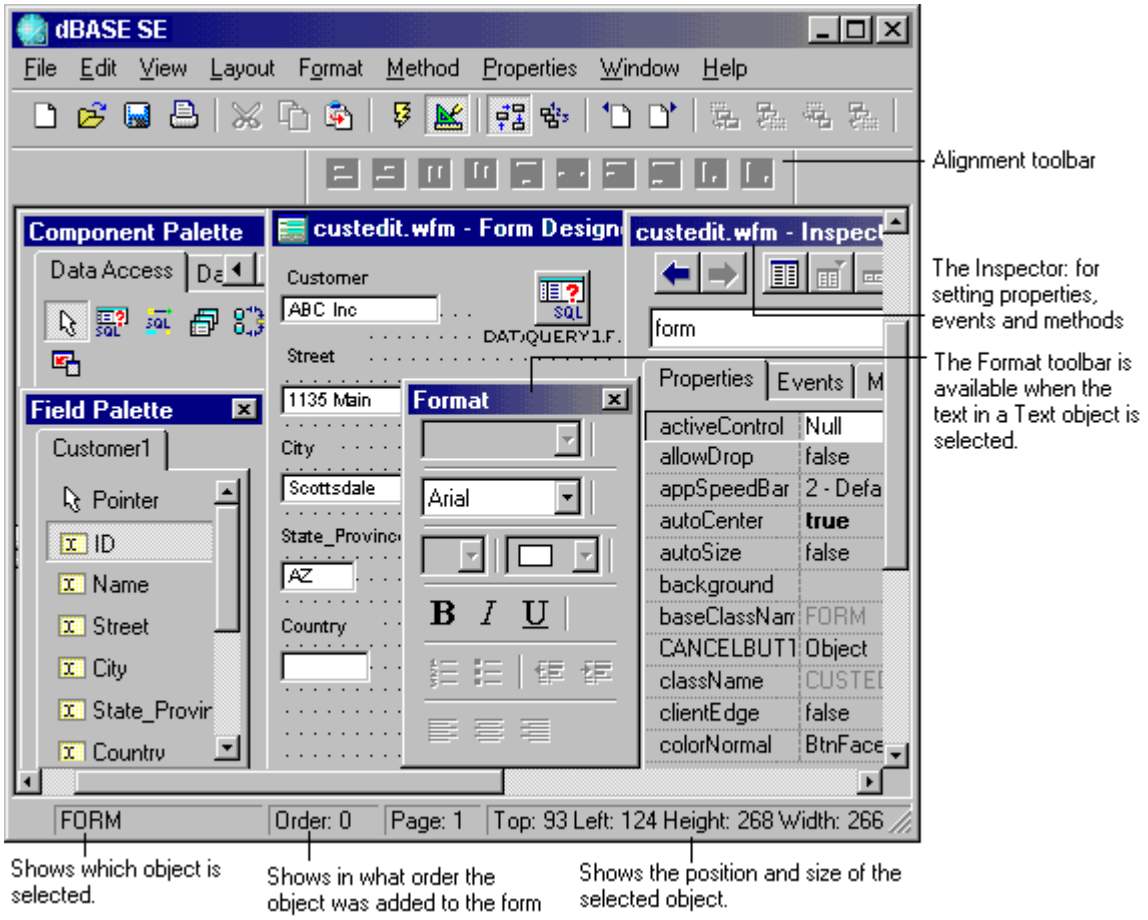
- Use Design mode to design the appearance and behavior of the form or report and the components you put on it.
- Use Run mode to see how a form or report looks when running. In Run mode, the components become active. For example, you can enter data into an entryField control and edit data that's already there.

Use the Design and Run toolbar buttons, or choose the appropriate command from the View menu, to switch between modes.

The Form Design Window

A form in the Form Designer appears on the desktop a shown in Figure 7.1

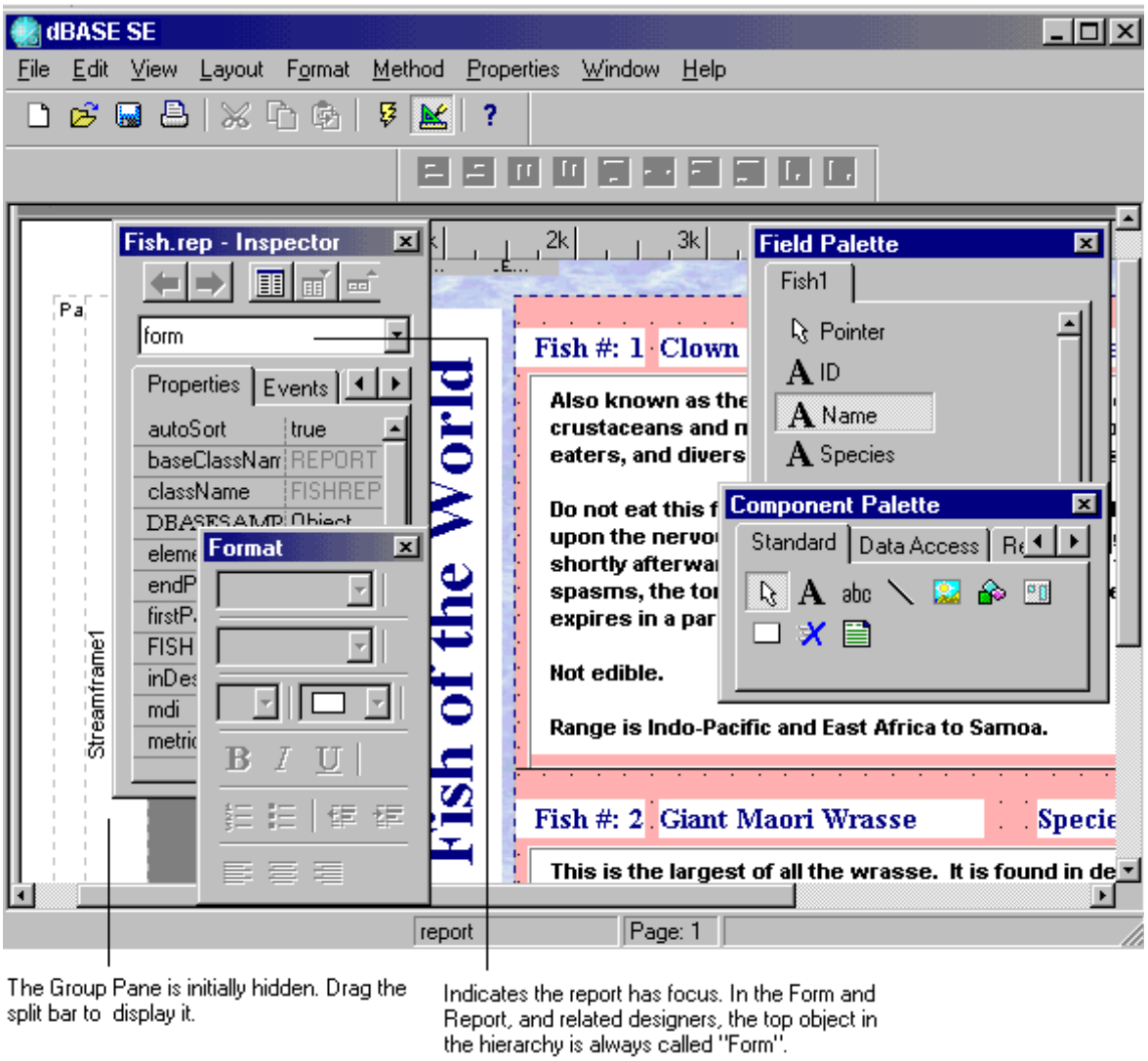
Figure 7.1 Form Designer with a wizard-created form



The Report Design window

A report in the Report designer appears on the desktop as shown in Figure 7.2

Figure 7.2 Report Designer with a wizard-created report



The report design surface has several objects the form design window does not, for example, pagetemplate and streamframe. These objects are necessary for formatting report pages. See Chapter 12, "Designing reports", for information on working in the report design window.

The visual design is reflected in your code

In both designers, the work you do with the visual design elements is reflected in the underlying code and vice versa. Press F12 to switch between the design surface and code.

Component palette

The Component palette displays the components and data objects you can add to the form or report you're designing.

To open the Component palette, do one of the following:

- Choose View | Component Palette.
- Right-click anywhere on the form or report window and choose Tool Windows | Component Palette from the context menu.

Depending on which designer has focus, or whether you have installed the *dBASE SE* samples (which include a number of custom components that appear automatically on the Component palette), you'll see a selection of the following pages on the Component palette:

Tab name	What's on the page
Standard	Common user interface controls, such as list components, buttons, text and image components, and so on.
Data Access	Database access objects required to connect to a table, group of tables, or to ensure record-locking
Data Buttons	Buttons and toolbars (both image-style and text-labeled) for navigating through data. Installed with the *dBASE SE* samples
Report	The streamframe and group objects used to lay out reports
Custom	Custom components that you create yourself (or obtain from a third party) or that appear in applications in the *dBASE SE* samples
ActiveX	ActiveX applications from third-party developers.

Standard page

This table briefly describes the standard user-interface controls appearing on the Standard page of the Component palette. For more details, select the component and click the Question Mark button on the toolbar.

Table 7.1 Standard controls

Component	Use to . . .	Example/Explanation
Text 	Display text that cannot be edited by users. The text can be any alphanumeric characters allowed in a character expression	Use for a field label, heading, instruction, prompt, or any other non-editable display text. Format with the Format menu or Format toolbar.
TextLabel 	Display information on a form or report, wherever features such as word-wrap and HTML formatting are not required.	TextLabel is a simple, light-duty object which consumes fewer system resources than the Text component. The TextLabel component does not support in-place editing on design surfaces. The *text* property of the TextLabel component may contain character string data only.
EntryField 	Let a user enter a single value, text or numbers, into a data-entry field	Example: Data entry area for entering a value for a particular field of a table. Must be DataLinked to the table field.
PushButton 	Let a user perform a task with a single mouse-click.	A control that a user can click to execute code that you attach. (Sometimes called a command button.)
CheckBox 	Let a user toggle between two choices of a logical value. Or choose a number of options that are not mutually exclusive.	Check boxes often are arranged in groups to present choices or options a user can turn on or off. Any number of check boxes in a group can be checked at a time
RadioButton 	Let a user select one choice among a group of mutually-exclusive possible values.	Example: A group of buttons labeled Credit, Cash, Check, Visa, and MC to choose among for entering only one of those values in a PAY_METHOD field of a table.
Line 	Organize elements visually	The line is a divider that may be extended vertically, horizontally, or diagonally to visually divide a form into sections. Users cannot edit or manipulate it.
Editor 	Display the contents of a text file or memo field.	Text exceeding the size of the box causes a scrollbar to appear. You can choose to allow users to edit this text.
ListBox 	Display values in a fixed-size, scrollable list box, from which a user can select one or more items.	The values in a list can be file names, records, array elements, table names, or field names.

Table 7.1 Standard controls

Component	Use to . . .	Example/Explanation
ComboBox 	Combine an entry field and a drop-down list box. A user clicks the down-arrow button to display the list.	A ComboBox accepts a value typed into the entry field or selected from the drop-down list box.
Image 	Display an image.	Display area for a bitmap image stored in a binary field, resource file, or graphic file.
Shape 	Visually divide a form into sections, for example, to place related RadioButtons within a box.	A visual appearance element. By setting the component's *shapeStyle* property you can create rounded rectangles, rectangles, ellipses, circles, rounded squares, and squares. You can also set the line style, weight, and interior color.
Container 	Create moveable panels that can contain other components on a form.	Example: Moveable toolbars and palettes.
Browse 	Display multiple records in row-and-column format.	The Browse component is maintained for compatibility and is suitable for viewing and editing tables open in work areas. For forms that use Data Access, use a Grid object instead.
Grid 	Display live table data in row/column format in a programmable component.	The Grid object is a multi-column grid control for displaying the contents of a rowset. The *dataLink* property is set to the rowset. Columns are automatically created for each field in the rowset
Rectangle 	Organize elements visually into boxes or create custom buttons.	A graphic element for boxing objects. You can set the size, line weight, and fill of the box. It can respond to mouse clicks and other events.
Progress 	Provide visual feedback to the user about the progress of long operations or background processes.	A rectangular bar that "fills" from left to right, like that shown when you copy files in the Windows Explorer. Use *position* to set a default position for the progress bar. At run time, *position* tracks the exact location as values increment. Use *max* and *min* to set the range of *position*. By default, the progress meter advances by a value of one.
PaintBox 	Create custom form controls	The PaintBox provides a window space in which you can call API functions in Windows. Users never interact with it directly. *dBASE SE* does not paint the area.
NoteBook 	Make a multipage dialog box, with labeled tabs to display sections of information or groups of controls within the same window. See TabBox for full-size tabbed forms.	You might use the Notebook control to create a tabbed dialog box with different groups of controls on each tabbed page. The Desktop Properties dialog box is a good example of this. Use the DataSource Property Builder to name or add tabbed pages to the window. Then drag the components you want to each tabbed page.
TreeView 	Display and control a set of objects as an indented outline based on their logical hierarchical relationships. The control includes buttons that allow the outline to be expanded and collapsed.	Use a tree view component to display the relationship between a set of containers or other hierarchical elements. You might use the TreeView as a way to select items from nested lists, much like the hierarchical view in the left pane of the Windows Explorer.
Slider 	Define the extent or range of values. By moving the slider along the trackbar, the user can change the current value for the control	You can set the trackbar orientation as vertical or horizontal, define the length and height of the slide indicator and the slide bar component, define the increments of the trackbar, and whether to display tick marks for the control. Examples: A volume control to play back sound files, or a color saturation adjuster for an image viewer
Vertical scroll bar 	Allow users to vertically scroll a grouping of controls, or a large control that has no integrated scroll bars	Example: A custom dialog box containing an area filled with many file icons.
Horizontal scroll Bar 	Allow users to horizontally scroll a grouping of controls, or a large control that has no integrated scroll bars	Example: A custom dialog box containing an area filled with many file icons.

Table 7.1 Standard controls

Component	Use to . . .	Example/Explanation
TabBox 	Group related data items on overlapping pages with labeled tabs	Use a TabBox to display multiple pages the full size of the form. A user selects a tab to display the items on the TabBox. Similar to Notebook, except for the full form size.
SpinBox 	Provide up and down arrows to assist changing a numeric value.	You can type a number into the numeric entry field or can increment or decrement the number by clicking the up and down arrows.
OLE 	Create an object linking and embedding (OLE) client area in a form, in which you can embed, or link to, a document from another application	Using an OLE control, a document from another application, for example, a sound file from a sound recorder application, can be opened from your *dBASE SE* application
ReportViewer 	Displays a report in a sizeable frame	The report is executed when the form is opened.

Data Access page

Data objects provide live connections and session control to tables and databases. A form or report that accesses a table must have at least one Query object on it, returning a rowset from the table. A StoredProc object that returns a rowset (as a query would) can be used in place of the Query object.

Note Once you have set up a group of data objects to return rowsets, you can save that group in a data module for easy reuse in other forms and reports or other applications.

This table describes the data objects available from the Data Access page of the Component palette. For details on the *dBASE SE* Data Model and use of the Data objects, see Chapter 6, "Accessing and linking tables".

Table 7.2 Data Access

Object	Lets you...	Explanation
Query 	Run an SQL query on any table, including local .DBF and .DB tables. Query objects enable components to display data from tables on forms and reports.	You set a Query object's SQL property to the SQL statement that selects a rowset. In addition to linking a table to a form or report, this populates the Field palette. You must use a Query object containing an appropriate SQL statement to connect to a table or database (unless you are using a StoredProc object to return a rowset from an SQL database).
StoredProc 	Run a stored procedure on an SQL server. This capability is available only when accessing tables on a server that supports stored procedures.	Place the StoredProc control on a form or report and link the control to a stored procedure by setting its *procedureName* property. If the stored procedure returns a rowset, it may be used in place of a Query object.
Database 	Set up a persistent connection to a database, especially a remote client/ server database requiring a user login and password.	Gives *dBASE SE* forms and reports access to SQL databases (or another group of tables identified by an alias). To add connections to SQL databases or other multiple tables via a BDE alias, add a Database object to your form. You must have first created a BDE alias for the database by using the BDE Administrator.
Session 	Session objects enable basic record-locking, so that multiple users do not modify the same record at the same time. Session objects also help to maintain security logins for local .DBF or .DB tables.	Use only if you are creating a multithreaded database application. When you open a form, a default session is created, linking the form to the BDE and connected tables. If you need separate threads for each user (to ensure record-locking), add a Session object to your form. A unique session number is assigned to track each user's connection to the table.
DataModRef 	Use a preset data access setup stored in a data module.	Use to give a form or report access to a set of data access components you've programmed and stored in a data module.

Data Buttons page (forms)

If you installed the *dBASE SE* samples, the Component palette in the Form designer displays a page of buttons that let users navigate through records, locate and filter data, edit data, and so on.

Both standard and image-style buttons with identical functionality are available. The names of standard button components begin with button, and the names of image-style components begin with bitmap. In addition, you can choose a VCR-like control panel including a full set of navigational buttons, a report page-number object, and a rowstate object. This table describes the components available for working with data.

Table 7.3 Shading Properties in the Form Designer

Component	What it is	What is does
ButtonAppend BitmapAppend	An append-record control.	Lets users put the table that is linked to the form into Append mode to enter a new record. Clicking the Append button again adds the new record to the table and keeps the table in Append mode.
ButtonDelete BitmapDelete	A delete-record control.	Lets users delete the current row from the table that is linked to the form.
ButtonSave BitmapSave	A save-record control.	Lets users save the current row.
Buttonabandon Bitmapabandon	An abandon-changes control.	Lets users abandon any changes made to the current row and return to the last saved contents of the row.
ButtonLocate BitmapLocate	A search-records control.	Lets users go to the first row that matches the criteria. When the user clicks the Locate control, the form goes blank. The user then types in the criteria for the search and clicks the Locate control again.
ButtonFilter BitmapFilter	A filter-records control.	Lets users display records that meet a specific criteria. When the user clicks the Filter control, the form goes blank. The user then types in the criteria for the filter and clicks the Filter control again.
ButtonEdit BitmapEdit	An edit record control.	Lets users edit the current row. (Required only when *AutoEdit* is false.)
ButtonFirst BitmapFirst	A first-record control.	Displays the first record in the table that is linked to the form.
ButtonPrevious BitmapPrevious	A previous-record control.	Displays the previous record in the table that is linked to the form.
ButtonNext BitmapNext	A next-record control.	Displays the next record in the table that is linked to the form.
ButtonLast BitmapLast	A last-record control.	Displays the last record in the table that is linked to the form.
BarDataVCR	A set of navigational controls.	Contains the bitmap versions of the First, Previous, Next and Last buttons listed earlier in the table.
BarDataEdit	A set of edit controls.	Contains the bitmap versions of the Append, Delete, Save, Abandon, Locate, and Filter buttons listed earlier in the table.
Rowstate	Displays the state property of a given rowset, for example, whether it is Read-Only.	The other controls update this control.

Report page

This page of the Component palette contains the data formatting components required for reports.

Table 7.4 Components specific to reports

Component	What it is	What it does

Table 7.4 Components specific to reports

StreamFrame 	The StreamFrame object receives and displays rowset data streamed from linked tables (specified in its *streamSource* property). One or more streamFrame objects may be contained within the pageTemplate object.	Dropping a component, such as a check box, into the streamFrame area of a report will cause that object to be printed as part of the report's row data.
Group 	The Group object is descended from the streamFrame object that contains data from the query's rowset. By dropping a Group object on a report's streamframe, a Headerband and Footerband are created, with editable placeholder text for the group's heading. A streamFrame may contain several Group objects.	Groups the display of rowsets by the value of a selected field. For example, in a "Sales by District" report, you might have a Group object for each District to display sales rowsets for that district.

Custom page

The Custom page of the Component palette contains custom-built components. If you didn't install the *dBASE SE* samples, you won't see the Custom page until you create your first custom component and assign it to the palette. If you did install the samples, you'll see that the Custom page already contains custom components that are used in the sample applications.

You can build new components from scratch, and you can alter existing components and save them as custom components. See "Creating custom components" on page 5-13 for instructions.

Using ActiveX (*.OCX) controls

To use an ActiveX control in your forms and reports,

1. In the Form or Report designer, choose File | Set Up ActiveX Components.

   The dialog box that appears shows all available controls registered on your system.

2. Select the desired controls.

   Selected controls appear on the ActiveX page of the Component palette, ready for use. After placing a component on a form, the Inspector shows the properties of the ActiveX control. To use the control's own property dialog box, right-click the control and choose ActiveX Properties from the context menu.

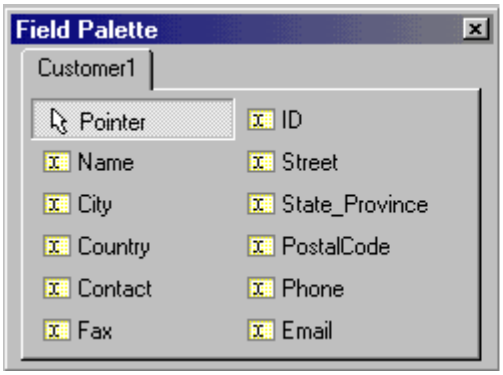
The Field palette

The Field palette displays fields for each Query object that's linked to an existing table, as long as the Query object's *active* property is set to *true*. Fields available on the Field palette are linked to a table through the *dataLink* property.

To open the Field palette, either

- Choose View | Tool Windows | Field Palette (it's a toggle).
- Right-click anywhere in the designer and choose Tool Windows | Field Palette from the context menu.

Figure 7.3 Field Palette



If you haven't checked Revert Cursor To Pointer in the Customize Tool Windows dialog box, click the Pointer button to return the cursor to a standard pointer after you have used it to place a field.

Fields shown are from a table named "Customer". Each field is "live" and will show data in the designer. All data will be available when you run the form or report.

Dragging a field from the Field palette onto a form or report saves you the work of having to set its *dataLink* or *text* property manually for each component you want to link to a field in a table, although you can do it manually, if you want to.

If no active Query object exists on the form or report, the Field palette is empty, showing only the Pointer button. When you begin to design a data-aware form or a report, first add a Query object and set its *sql* property to the appropriate SQL statement and its *active* property to *true*. If you drag a table from the Navigator to the design surface, this automatically creates a Query object that selects all the records in that table and links the table to the form or report. See Chapter 6, "Accessing and linking tables", for more details.

Once an active Query object exists on the form or report with its *active* property set to *true*, its fields appear on the Field palette as linked components. The type of the component depends on the data type of the field. For example, a Boolean field appears on the Field palette as a CheckBox control. To change the control type of a field, right-click the Field palette, and choose Associate Component Types from its context menu, or choose File | Associate Component Types.

If more than one Query object exists on the form or report, each table's fields are displayed on a separate page of the Field palette.

The Inspector

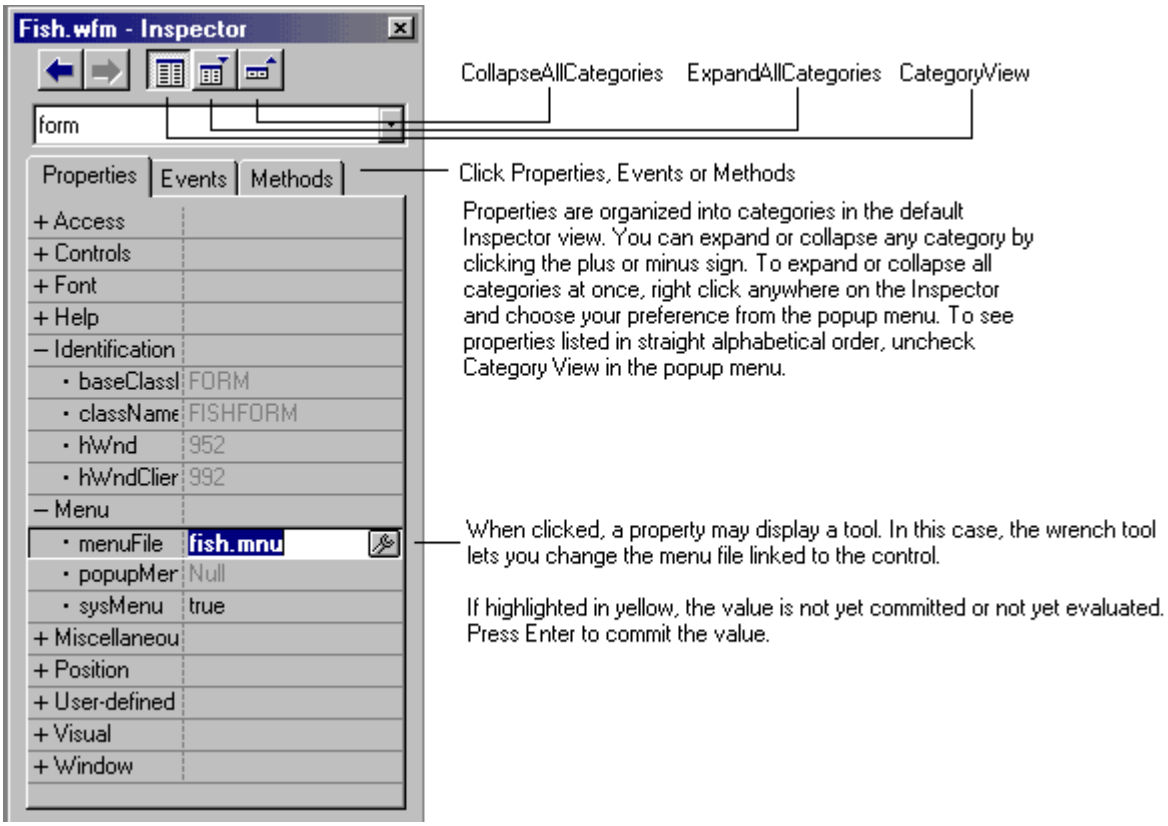
You can change a component's properties in the Inspector. When you select a component in a form or report, the Inspector displays the component's properties. If the Inspector is not open, do one of the following, all of which toggle the view of the Inspector:

- Press F11.
- Choose View | Inspector (this command is a toggle).
- Right-click the selected component and choose Tool Windows | Inspector from the context menu.

When you have selected multiple components, you can change their common properties simultaneously. When you change a property value or link code to an event for a multiple selection, the change affects all components in the selection.

Note You cannot change methods for multiple selections.

Figure 7.4 Using the Inspector



The Inspector has three tabbed pages that show the properties, events, and methods of the selected object. The name of the currently selected object appears in the drop-down list box at the top of the Inspector. Click the Down arrow of this box to select a different object, or select the object on the form or report, itself.

Properties, methods and events set by you, or that have no default value, are shown in bold. (Bold properties are ones that will be streamed out.)

Properties page of the Inspector

The Inspector's Properties page displays the properties of the current object. The right column shows the current value for each property.

You can set a property value in any of the following ways:

- Type the value into the column to the right of the property name.

Note Yellow highlighting of an entry means that it's not yet committed or not yet evaluated. Press Enter to commit a change.

- Press Ctrl+Enter in the value column to rotate through a list of properties or to toggle logical values, or double-click the value column to do the same.
- Select a value from a history list or other drop-down list, when available.

- Click the wrench tool button that appears to the right of the property value. Tools are not available for every property.

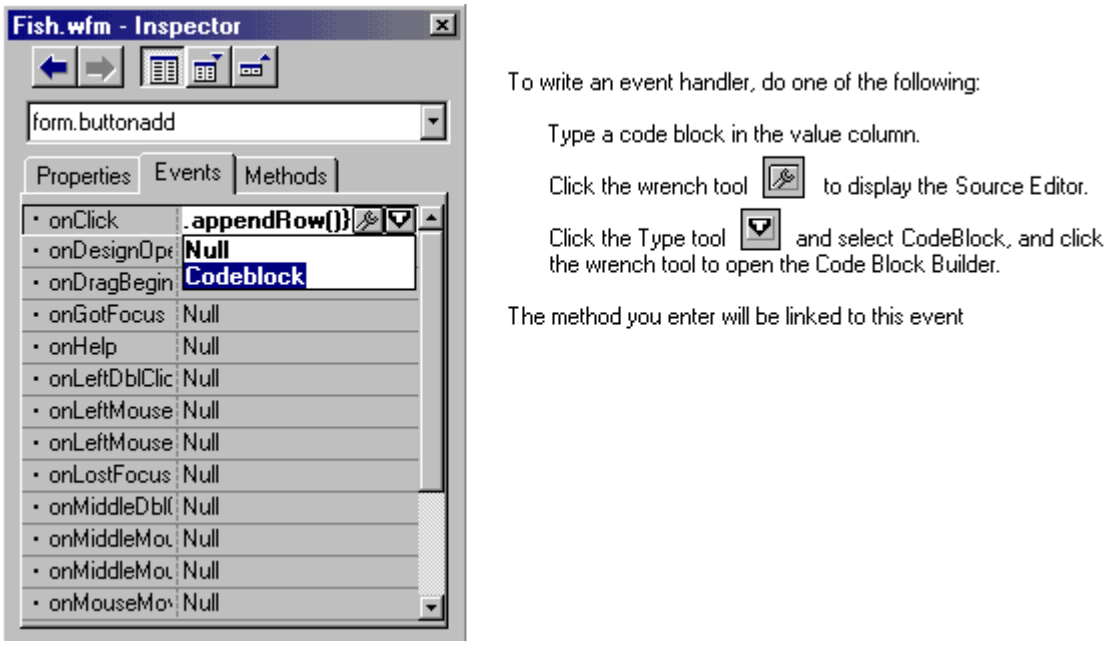
  The tool button may produce

  - A property builder in which you can build or select a value. For example, you can display the *Color* property builder to set the color for an object.
  - The String Builder dialog box, which makes it more convenient to type a long string.

Events page of the Inspector

The Inspector's Events page displays the events to which the current object can respond. When you select an event, its value area becomes a text box with a tool button.

Figure 7.5 Events page of The Inspector



To specify what you want to happen when an event occurs, you can do one of the following:

1 Type a code block into the text box for the event. Or, if you want to use the Code Block Builder,

- Click the Type drop-down list beside the text box, and select CodeBlock.
- Click the wrench tool beside the text box.

  This opens the Code Block Builder. Type parameters, if any, in the Parameters text box, and type the code block in the Commands Or Expression box. It's okay to put only one statement on each line, and end it with a semicolon, where appropriate. When you click OK, the code block becomes a one-line code block in the event's value text box and in your code. See "The Code Block Builder" on page 9-3 for more information.

2 Write a method to link to the event. Click the tool button to display the Source editor with the cursor inside the skeleton of a new method, ready for you to type.

For information about code block syntax and writing a method, see Help.

You can also link and unlink events by using the Method menu from within the Source editor. See "The Method menu", on page 7-12.

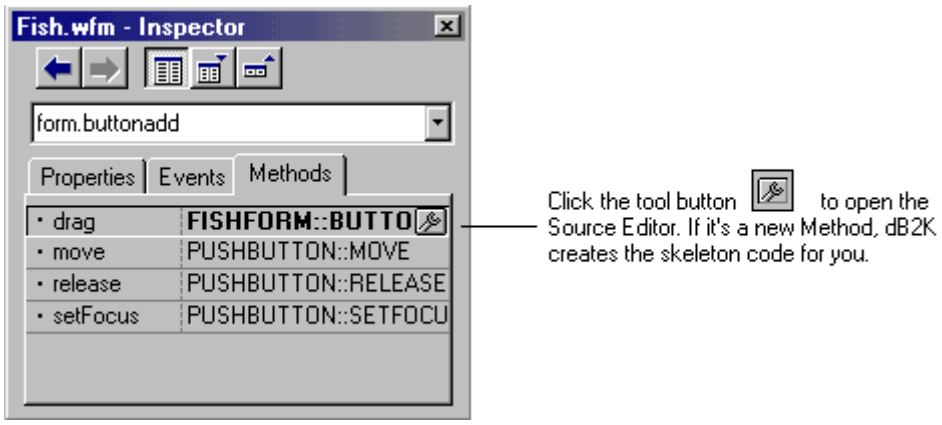
Methods page of the Inspector

The Inspector's Methods page displays the current object's built-in methods, that is, the methods pre-defined for the component. You can call these methods with methods you create in the Source editor. Methods you create in the Source editor can be inspected on the Methods page.

To delete a method in code, you must be in the Source editor. Then, with the cursor in the method you want to delete, choose Method | Delete Method.

Note A function inside a class is a "method." The keyword for method is "function."

Figure 7.6 Methods page of The Inspector



The Method menu

You can use commands on the Method menu when working with code. The last three commands open dialog boxes that can simplify writing methods.

Table 7.5 Method menu commands

Command	What it does
New Method	Creates dBL skeleton code for a new method in the Source editor: // {Linked Method} Form.OnOpen function Form_OnOpen You can do the same thing by clicking the tool beside an event in the Inspector.
Delete Method	Deletes the method that has the cursor in it and all references to the method from the code.
Verify Method	Attempts to compile the method, to make sure there are no syntax errors. This also happens when you switch focus from the Source editor to the designer.
Edit Event	Displays a dialog box that allows you to select objects in the left pane and, in the right pane, select one of the available events for editing. The selected event is then displayed in the Source editor for editing.
Link Event	Displays a dialog box similar to the Edit Events dialog box. You choose a control from the left pane and one of its events in the right pane. When you click OK, the new event is linked to that event.
Unlink	Displays a dialog box that allows you to view multiple events linked to a method and to remove any or all of them. When you click OK, the selected link is unlinked from that event.

Manipulating components

This section describes how to work with components: placing them, resizing, aligning, and so on.

Placing components on a form or report

You can place a component on a form or report by selecting its icon from the Component palette or from the Field palette.

Note To see fields on the Field palette, you must have first placed an active Query object on the form. See page 7-8. Fields represented on the Field palette are already linked to the fields of the table(s) specified in the Query object.

To place a component,

1 Click the component on the palette to select it.

2 Drag on the design surface until the component is the size you want, or click on the design surface without dragging to add a component in its default size.

Note If you're placing a field, simply click the form window; dragging will not size the field while you're adding it, although you can size it by dragging it after you've dropped it on the form or report.

Alternatively, you can add a component in these ways:

- Double-click the component in the palette; it appears at a default position on the design surface.
- Drag the component from the palette to the design surface.

By default, the mouse reverts to a pointer after you place a component on the design surface. If you want to place multiple instances of a component without having to return to the Component palette to select the component anew each time, uncheck the Revert Cursor To Pointer option in the Customize Tool Windows dialog box (View | Tool Windows | Customize Tool Windows). If you've unchecked this option, then before you select another component you have to first click the Pointer icon on the Component palette.

Special case: container components

Besides the form itself, *dBASE SE* provides other components that themselves contain components. Examples are the Container and Notebook components. You can use these components to group other components so that they behave as a unit at design time. For instance, you might group pushbuttons and check boxes that provide related options to the user.

When you place components within container components, you create a new parent-child relationship between the container and the components it contains. Design-time operations you perform on these "container" (or parent) components, such as moving, copying, or deleting, also affect any components grouped within them.

Note The form remains the owner for all components, regardless of whether they are parented within another component.

You generally want to add container components to the form before you add the components you intend to group, because it's easiest to add components that you want grouped directly from the Component palette into the container component. However, if a component is already in the form, you can add it to a container component by cutting and then pasting it. If you drag it in, it does not become a child to the container, and will not act as part of the container unit.

Selecting components

To work with a component once you've placed it on the form, first select it. Once you select a component, you can resize it, move it, or delete it. You can also change its properties.

To select a component, do one of the following:

- Click the component.
- Press Tab or Shift+Tab until it's selected.
- Select it from the drop-down list at the top of the Inspector.

When a component has focus, its *handles*—small, black squares around the periphery—are visible.

Note If it is a component that is part of a custom form or report class, the handles are white to remind you that you have selected such a component, because you may not want to change it.

Moving components

To move a component, select it, and then do one of the following:

- Drag the component to the position you want. As soon as you move the mouse, the pointer becomes a hand. This indicates you're moving the component.
- Press any of the arrow keys to move the component in the direction of the arrow. If Snap To Grid is turned on, the object moves one gridline at a time.

- Change the object's position properties in the Inspector.

To move a multiple selection of components, put the mouse cursor within the borders of one of the components, and then either drag or press the appropriate arrow key to move your selection in the direction you want.

If Snap To Grid is checked in the Properties dialog box of the designer you're working in, then components align to the grid.

Cutting, copying, pasting, deleting components

You can access the cut, copy, and paste commands from the Edit menu, the context (right-click) menu, or the toolbar buttons. Select the component or components, and then choose the appropriate command. To delete a selected component or multiple selection of components, choose Edit | Delete (or press Del).

Undoing and redoing in the designers

You can undo operations on a form or report. Once you undo an operation, the previous action is available to Undo.

You can undo and redo values that you set in the Inspector. Once you undo a value, the Undo command on the Edit menu becomes Redo.

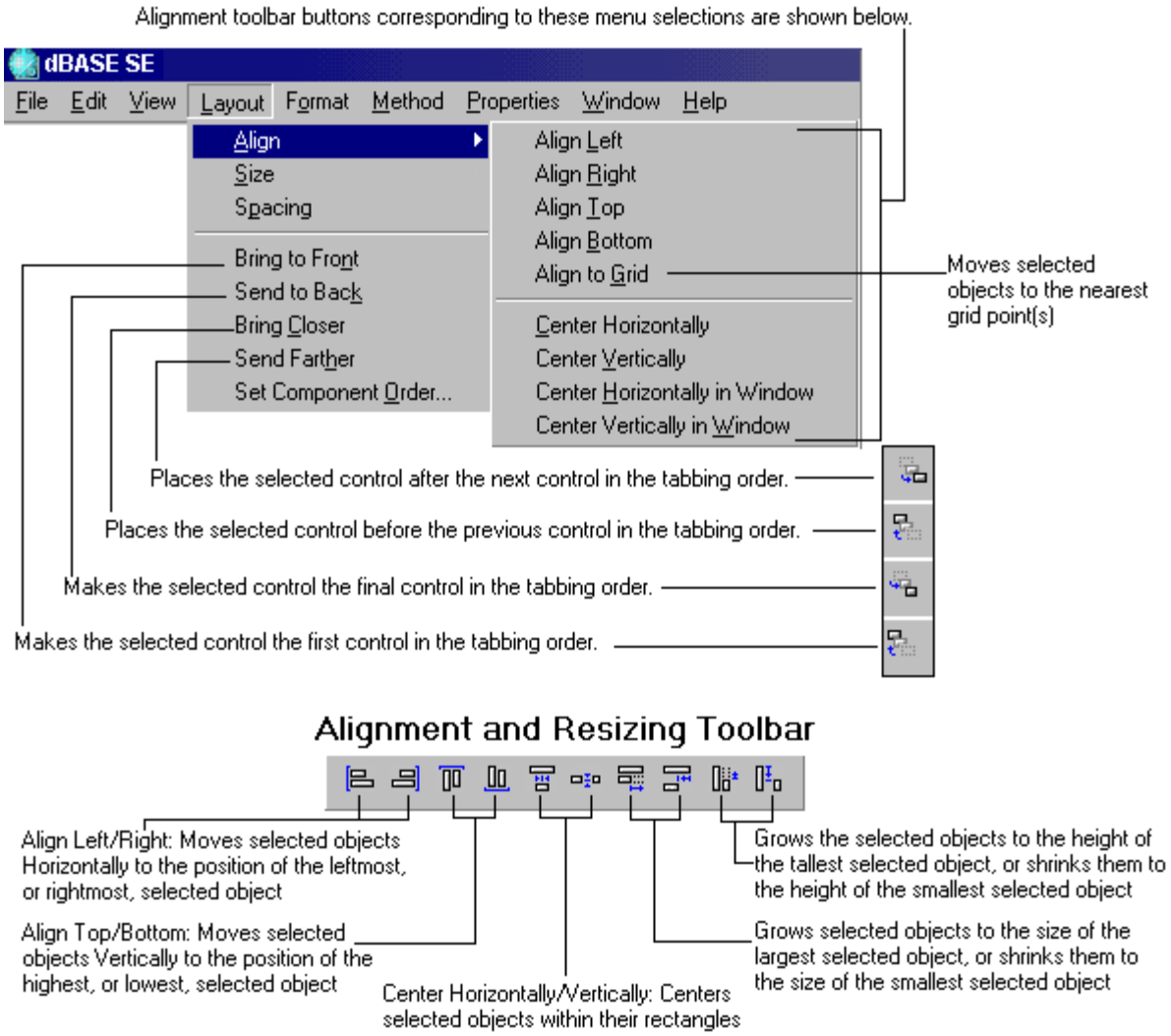
To undo an operation, choose Edit | Undo (or press Ctrl+Z). To redo an operation, Choose Edit | Redo (or press Ctrl+Z).

Aligning components

You can align one or more selected components by using the Layout | Align menu commands or the corresponding toolbar buttons (to display the Alignment toolbar, choose View | Tool Windows, and check Alignment Toolbar.) These commands adjust the position of objects in relation to each other or in relation to the form or report. To find out what each option does, highlight it and read the explanation in the status bar, or press F1. Here's a summary

:

Figure 7.7 Layout | Align commands



Resizing components

To resize a component, select it and do one of the following:

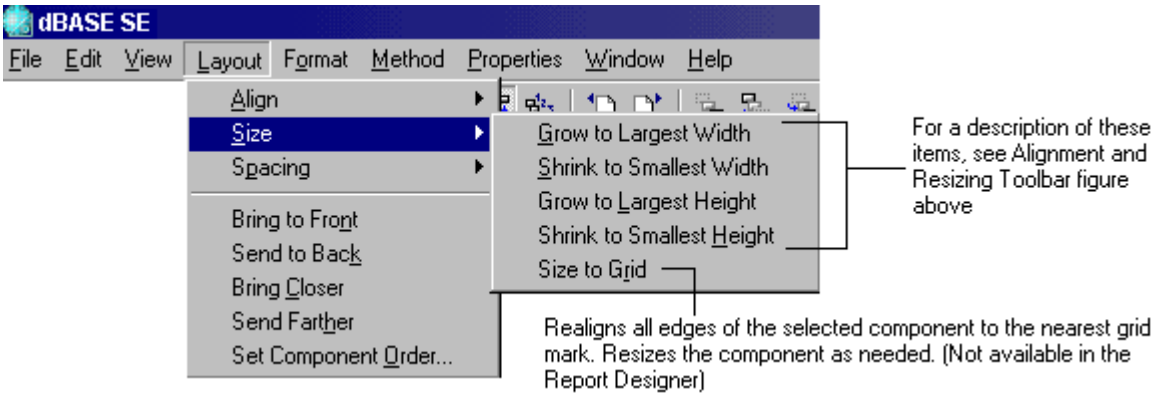
- Place the mouse pointer on one of its handles. When the pointer turns into a double-headed arrow, drag the handle to size the component the way you want.
- Press Shift+any arrow key to resize it in the direction of the arrow.

You cannot resize a multiple selection of components with the mouse; however, you can press Shift+an arrow key to resize a multiple selection in the direction of the arrow.

To change the sizes of multiple objects to conform to one size, choose an option from the Layout | Size menu or the corresponding button in the Alignment toolbar. (To display the Alignment toolbar, choose View | Tool

Windows, and check Alignment Toolbar.) To find out what each option does, highlight it and read the explanation in the status bar.

Figure 7.8 Layout | Size commands



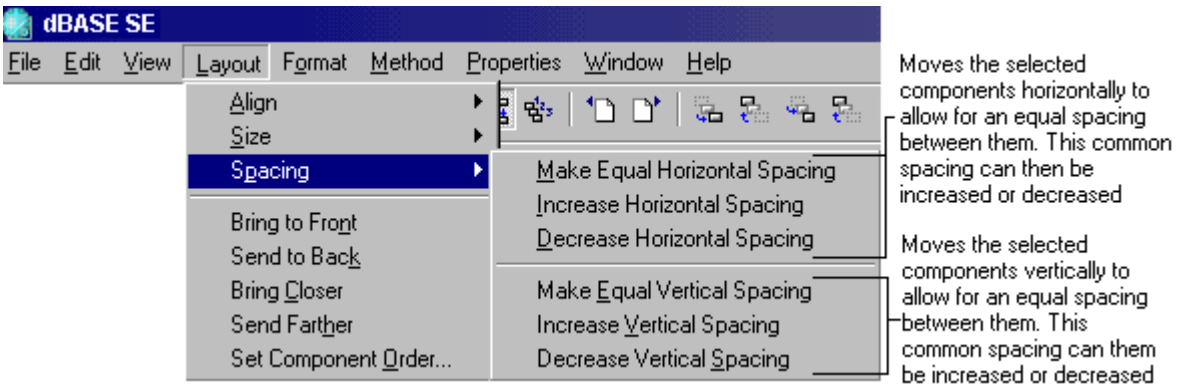
Spacing components

To change the spacing of multiple components in the Form designer, select the components, and choose an option from the Layout | Spacing menu.

Note The Spacing menu is not available in the Report designer.

To find out what each command does, highlight it and read the explanation in the status bar.

Figure 7.9 Layout | Spacing commands

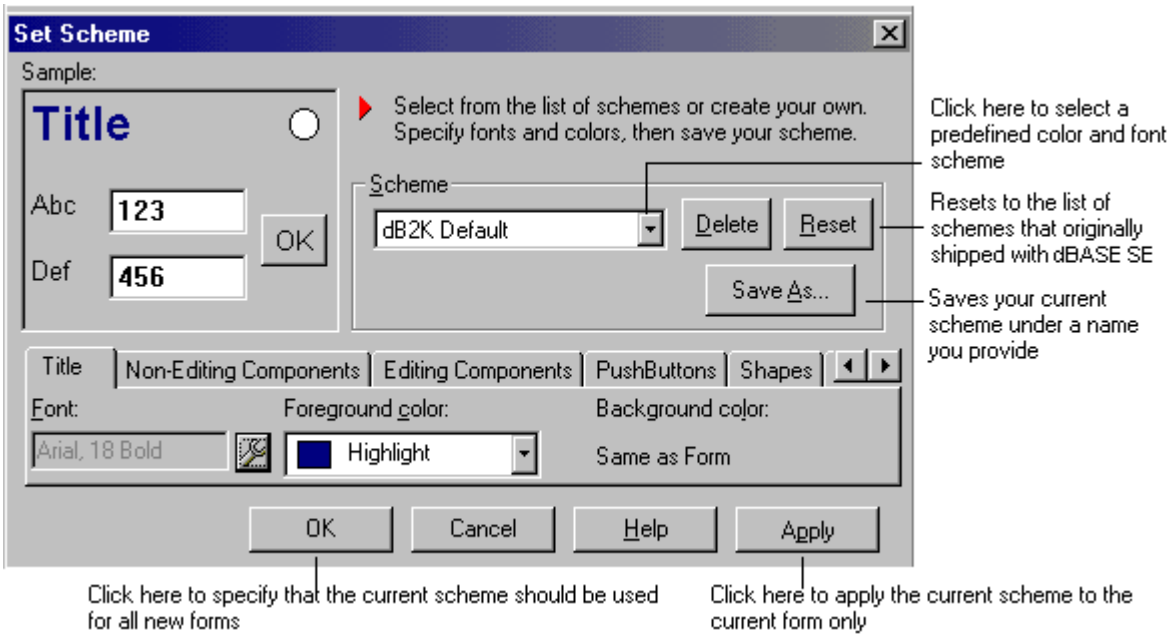


Setting a scheme (Form designer)

The Format | Set Scheme command displays the Set Scheme dialog box, which lets you set colors and background images and save them as a reusable scheme (you can do this in the Form wizard, as well—it uses the same dialog box). This is useful for maintaining a consistent look over several pages of a form or across related applications. You can either

- Choose a predefined scheme
- Set your own scheme

Figure 7.10 Set Scheme dialog box



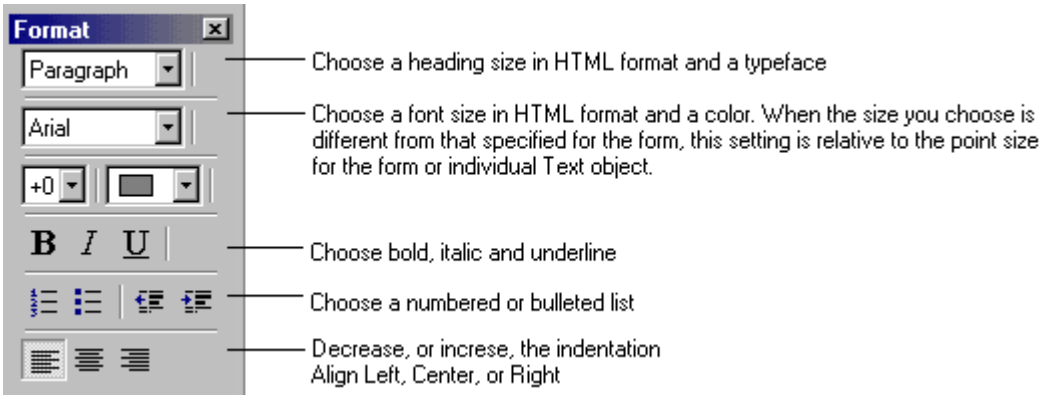
Editing a Text object

You can change the words, font properties, and color of an entire Text object by selecting the object and setting the desired properties in the Inspector. Use the *text* property to specify the words. (Click the wrench tool beside the text property to open a string-builder dialog box.)

To edit directly on the design surface, or to format *parts* of a Text object individually, or to format in ways not available in the Inspector (for example, to specify a list format), select the Text object, and then select the text to get an insertion point. After you have an insertion point, you can drag over words to select them or double-click to select one word. Then you can edit the words in-place, or format, as desired, using either of the following:

- Format toolbar (View | Tool Windows | Format toolbar)
- Format menu

Figure 7.11 Format toolbar



Saving, running, and printing forms and reports

To save a form or report design, either

- Click the Save toolbar button.

- Choose File | Save or File | Save As.

Enter a file name and specify a directory location. A form is given the extension .WFM; a report is given the extension .REP. A new file is placed into the current project, if a project is open.

Opening a form or report in Run mode

To open a form or report in Run mode, do one of the following:

- Choose File | Open. If you're opening a form, in the Open File dialog box, choose the form you want to run, select the Run Form button at the bottom of the dialog box, and choose OK. If you're opening a report, running it is your only choice from this dialog box.
- In the Navigator, double-click the form or report you want to run. Or select it and press F2.
- Type DO Formname.wfm in the Command window, where Formname is the name of your form, or DO ReportName.rep, where ReportName is the name of your report.

Printing a form or report

Print a form or report in Design or Run mode by doing one of the following:

- Click the Print toolbar button.
- Choose File | Print.

Chapter

8

Creating menus and toolbars

Most Windows applications offer menus of some kind—standard pulldown menus, popup menus, or both. Most also feature static or detachable toolbars.

This section describes how to create these objects and integrate them into your *dBASE SE* applications.

Like all other objects in an object-oriented environment, menus and toolbars can be designed to be completely reusable by any number of forms. For that reason, we'll start with the task you'll face most often—attaching objects to forms.

Attaching pulldown menus to forms

To attach a pulldown menu to a form, choose your form's *menuFile* property, click the tool button, then locate the .MNU file you want on the form.

If you haven't already created a .MNU file or don't have a sample installed, you can create one using the Menu designer (described later in this section).

Note that at design time, menus don't appear on your forms. To see an attached menu in action, you have to run the form. If the menu you're attaching is also open in the Menu designer, you must close that as well before running the form.

Also note that if the MDI property of your form is set to *true* (the default), your pulldown menu appears on the parent window or application frame, not on the form itself.

Attaching popup menus to forms

Popup menus normally appear when a user right-clicks a form or control. Like dropdown menus, popup menu files (extension .POP) can be created using a special designer, also described later in this chapter.

However, popups are attached to forms in a different manner than pulldown menus.

To attach a popup menu, you must assign the popup object to your form's PopupMenu property. Unlike pulldown menus, however, you can't make the connection with the popup menu file name alone. To attach a popup, you need to add some code, either through a codeblock or within a form or control event handler.

The simplest and most common means of attaching a popup to a form is through a form's *onOpen* event. If, for example, you create a popup menu file called MYPOPUP.POP, you can make the menu available to any form by typing a codeblock like this into the form Inspector's *onOpen* event:

```
{;do mypopup.pop with this, "popup"; this.popupmenu = this.popup}
```

Alternatively, you can click the *onOpen* event's tool button and apply the same code as a linked method:

```
// {Linked Method} Form.onOpen
  function Form_onOpen
        do mypopup.pop with this, "popup"
        this.popupmenu = this.popup
```

Creating toolbars and attaching them to forms

Note that this topic covers both object creation and attachment. That's because, like popup menus, you need to add some code to attach toolbars to your forms. However, unlike pulldown or popup menus—which you can create using special visual designers—you also have to define your toolbars programmatically, either in a reusable program or within your form's code.

Like any other object, toolbar and toolbutton classes have a number of properties that allow you to modify the behavior and appearance of a toolbar. These properties, some of which are illustrated in the following code examples, will be described later in this chapter and are covered in detail in the printed and online *Language Reference*.

Creating a reusable toolbar

Here's an example of an object definition program, MYTOOLBR.PRG, which defines a basic two-button toolbar for use in any form or application.

```
parameter FormObj
  if pcount( ) < 1
    msgbox("DO mytoolbr.prg WITH <form reference>")
    return
  endif
t = findinstance( "myTBar" )
if empty( t )
  ? "Creating toolbar"
  t = new myTBar( )
endif
try
  t.attach( FormObj )
catch ( Exception e )
  // Ignore already attached error
  ? "Already attached"
endtry

class myTBar of toolbar
  this.imagewidth = 16
  this.flat = true
  this.floating = false
  this.b1 = new toolbutton(this)
  this.b1.bitmap = 'filename ..\artwork\button\dooropen.bmp'
  this.b1.onClick = {;msgbox("door is open")}
  this.b1.speedtip = 'button1'
  this.b2 = new toolbutton(this)
  this.b2.bitmap = 'filename ..\artwork\button\doorshut.bmp'
  this.b2.onClick = {;msgbox("door is shut")}
  this.b2.speedtip = 'button2'
endclass
```

Note The toolBar and toolButton properties used above - as well as other properties for the ToolBar and ToolButton classes - are covered in detail in the *Language Reference* and Help (search for "class ToolBar" or "class ToolButton".

Attaching a reusable toolbar

As with popup menus, you can attach a reusable toolbar definition file to your forms with a simple DO command. However, since forms don't have a toolbar property, the connection is defined in the toolbar's own *attach(_)* property. Thus, if you choose to connect the program described above through a form's *onOpen* event, the integration codeblock is simply this:

```
{;do mytoolbr.prg with this}
```

Or, if you prefer the linked method approach, click the *onOpen* event's tool button and add the integration code:

```
// {Linked Method} Form.onOpen
function Form_onOpen
    do mytoolbr.prg with this
```

Of course, you also need to provide a way to restore the toolbar if the user has closed it. You can do that by also adding the integration code (or codeblock) to the *onClick* event of another control, such as a menu item or

button. Should the toolbar already be running when it is summoned, *findInstance(_)* will let you know and let you block the creation of a new instance.

As is the case with pulldown menus, keep in mind that if your form's MDI property is set to True, your toolbar is owned by (and may only be docked to) the form's parent window or application frame.

Creating a custom toolbar

Defining a custom toolbar within a form uses much of the same basic code described above for defining and creating a reusable toolbar. The primary difference is that the toolbar is available only to the form in which it is defined.

Here's how the same toolbar described above could be adapted for use within a single form:

```
** END HEADER -- do not remove this line
*
* Generated on 08/20/00
*
parameter bModal
local f
f = new tooltestForm( )
if (bModal)
   f.mdi = .F. // ensure not MDI
   f.ReadModal( )
else
   f.Open( )
endif

CLASS tooltestForm OF FORM
with (this)
    onOpen = class::show_toolbar
    height = 8.6471
    left = 3.625
    top = 1.7059
    width = 23.75
    text = ""
endwith

this PUSHBUTTON1 = new PUSHBUTTON(this)
with (this.PUSHBUTTON1)
    onClick = class::show_toolbar
    height = 1.1176
    left = 4
    top = 2
    width = 15.875
    text = "PUSHBUTTON1"
    metric = 0
    fontBold = false
    group = true
endwith

// {Linked Method} Form.onOpen
function Form_onOpen

// {Linked Method} Form.pushbutton1.onClick
function PUSHBUTTON1_onClick

function show_toolbar
    t = findinstance( "myTBar" )
    if empty( t )
        ? "Creating toolbar"
        t = new myTBar( )
    endif
    try
        t.attach( form )
    catch ( Exception e )
        // Ignore already attached error
        ? "Already attached"
    endtry

ENDCLASS

class myTBar of toolbar
    this.imagewidth = 16
    this.flat = true
    this.floating = false
    this.b1 = new toolbutton(this)
    this.b1.bitmap = 'filename ..\artwork\button\dooropen.bmp'
    this.b1.onClick = {;msgbox("door is open")}
```

```
        this.b1.speedtip = 'button1'
        this.b2 = new toolbutton(this)
        this.b2.bitmap = 'filename ..\artwork\button\doorshut.bmp'
        this.b2.onClick = {;msgbox("door is shut")}
        this.b2.speedtip = 'button2'
    endclass
```

Note that the only change to the contents of the earlier program is the removal of the FormObj parameter definition (and related change to the referenced form object, the form, in the new method called *show_bar*) and the removal of the unneeded *pcount*(_) parameter check at the top of the file.

Otherwise, the code was simply partitioned and placed in the appropriate areas of the form source, and the new method, *show_bar*, was created to hold the instance-checking and toolbar creation/attachment code.

Creating menus with the designers

Two designers are available for creating menus—one for pulldown menus and one for popup menus. To open them, do one of the following:

- From the main menu, choose File | New | Menu (Alt+FNM) for the pulldown Menu designer, File | New | Popup (Alt+FNP) for the Popup Menu designer.
- From the Navigator, select the Forms tab, then double-click the Untitled menu icon for the pulldown Menu designer or the Untitled popup icon for the Popup Menu designer.
- From the Command window: enter CREATE MENU or CREATE POPUP.

Note that the only difference in appearance between the two designers is that the pulldown Menu designer contains a horizontal rule. This rule is the top-level menu border.

The designer menu

When you use either designer, a number of shortcuts are available through the main *dBASE SE* menu. These options are available by choosing Menu when either designer has focus.

You can use these shortcuts to insert an item before the current item (Insert Menu Item, Alt+NM or Ctrl+N), start a new submenu (Insert Menu, Alt+NM or Ctrl+N) or insert a separator (Alt+NT or Ctrl+T) before the current item in a pulldown or submenu. You can also delete the current item with Alt+ND or Ctrl+U (also see, "Adding, editing and navigating", on page 8-5.

If you’re designing a pulldown menu, two preset menus are available for insertion anywhere on your menu bar with the Insert "Edit" Menu (Alt+NE) and Insert "Window" Menu (Alt+NW) choices.

The last item on the Menu list—Toggle Type (Alt+NO)—is available for use on those occasions when you change your mind about the type of menu you want. It automatically switches the currently selected designer and converts its contents from pulldown style to popup style—or vice versa—any time.

Building blocks

Building basic menus through the designers is a simple two-step process of adding items, then adding code to make the items do what you want them to do.

Like any other object, each menu item has its own set of properties available through the Inspector (F11 to view). The "action" code is applied through an item’s *onClick* event.

Not all items need to perform an action, however. Some, like top-level items, normally only serve as entry points to additional menu choices. Lower-level items, and any item in a popup menu, can also serve as entry points to additional menus. These types of menus are called submenus (also known as "cascading" or "flyout" menus). File | New on the main *dBASE SE* menu is an example of this type of menu. And any submenu item can be specified as an entry point to another submenu.

Another type of "non-action" item is the separator bar, a horizontal line that lets you group items within menus. You specify a separator anywhere except in a top-level item. To make a separator, set an item’s Separator property to True.

To provide further visual cues and functionality, you can add graphics, mnemonics, check marks, shortcut keys, and conditionally enable or disable any item in any menu.

Adding, editing and navigating

To create a new menu, open a new Menu designer or Popup Menu designer window, type the name of your first item, and press Enter.

The cursor automatically drops a level and opens an editing block for the next item. Use the same sequence for entering additional items.

To edit items above or below the current item, use your Up and Down arrow keys. Tab and Shift+Tab lets you navigate left and right through your structure.

To add a submenu, select the item that will be the entry point to your submenu and press Tab. A new editing block appears to the right of the current item.

To add a new top-level item in a pulldown menu, select the rightmost existing top-level item and press Tab.

Note that other pulldown and submenus are hidden while you create new ones. You can return to view or edit the others any time by selecting the root item for each.

To insert a top-level or submenu root item in front of an existing one, choose an item, then choose Menu | Insert Menu (Ctrl+N or Alt+NM) from the main *dBASE SE* menu.

To delete an item, select the item and choose Menu | Delete Current (Ctrl+U or Alt+ND) from the main *dBASE SE* menu. Be aware, however, that deleting a top-level or submenu root item also removes all items and submenus below the item you are deleting.

You can also perform structural changes by dragging items and entire root/submenu systems from one location to another within your menus. To move items, just click, hold, drag, and release onto another item. Note that if you drag a held item onto another top-level or submenu entry point, the pulldown or submenu open up to allow you to relocate your dragged item.

To see your menus in action, you have to attach your menus to a form (as instructed earlier in this chapter), and save and close the designer that contains the menu you want to test. You can reopen saved menus for editing or redesign from the Forms page in the Navigator. As noted earlier, pulldown menus carry the extension .MNU, and popups are saved as .POP files.

Features demonstration

The following exercise demonstrates a number of menu creation principles and features, including preset menus.

1. Open a new pulldown Menu designer window.
2. Type &File (including the ampersand) at the cursor, then press Enter. Type &Form into the new item entry box, then press Tab. A new item entry box appears to the right of the current entry. Type &Close into this box.
3. If it isn't already open, press F11 to open the Inspector. Choose the Events tab, then type form.close( ) into your Close item's onClick event. Press Enter to save the change.
4. Back at the Menu designer, select the top-level "&File" item.
5. Press Tab. A new top-level item entry box appears. Press Alt+NE to insert a complete menu of basic editing commands. Now press the Tab key again for one more top-level item entry box.
6. Press Alt+NW. This time, a new top-level "&Window" item is created. This item has no subentries yet, but it will later.
7. Save the menu as MTEST.MNU, then close the Menu designer.
8. Open a new form in the Form designer. Add an entryfield control to the form.
9. Click on the form background. If it's not already in view, press F11 to view the Inspector. Click the tool button on the form's MenuFile property (Menu category), choose your MTEST.MNU file, and click OK. Keep other form properties at their default settings.

10 Press F2 to save (MTEST.WFM, for example) and run the form.

Because this is an MDI form (the default setting), the menu appears on the application frame, replacing the *dBASE SE* menu while the form has focus.

Click the Windows menu item; you should see a selectable list of other active *dBASE SE* windows. Now try the Edit menu commands. You should be able to use all of these standard Windows text editing commands on the text in your form's entryfield control.

The reason these two menus provide full functionality without any coding on your part is that the items use built-in menubar objects. You'll see how these objects work in the next topic when we examine the code behind the menu.

Note Since the properties used to create these preset menus belong only to the menubar class and are not available to the popup class, you can't use the properties in a popup menu.

Finally, try your File | Form | Close item to test your first piece of menu action code by closing the form.

Now let's go to the Source editor to examine the code structure of this menu.

Examining menu file code

The model for building menus is based on the hierarchy and containership of menu objects, not the kind of menu. You don't explicitly define menu bars, pulldown menus, or submenus. Instead, you build a hierarchy of menu objects, where each menu object contains another menu object or executes an action.

Just as a form contains controls, menus objects contain other menu objects. *dBASE SE* automatically determines where menus appear based on their level in the hierarchy.

The code below is the source for the menu file described in the previous topic, and illustrates how *dBASE SE* interprets and implements a menu structure.

(To view the source for any other menu file, choose a .MNU or .POP file on the Navigator's forms page, then choose Open In Source Editor from the file's context menu. Or you can type modi comm <filename.ext> in the Command window, where *filename.ext* is the .MNU or .POP file you want to examine.)

```
** END HEADER -- do not remove this line
//
// Generated on 10/24/00
//
parameter formObj
new mtestMENU(formObj, "root")
class mtestMENU(formObj, name) of MENUBAR(formObj, name)
  this.MENU2 = new MENU(this)
  with (this.MENU2)
    text = "&File"
  endwith
  this.MENU2.MENU3 = new MENU(this.MENU2)
  with (this.MENU2.MENU3)
    text = "&Form"
  endwith
  this.MENU2.MENU3.MENU7 = new MENU(this.MENU2.MENU3)
  with (this.MENU2.MENU3.MENU7)
    onClick = {;form.close( )}
    text = "&Close"
  endwith
  this.MENU12 = new MENU(this)
  with (this.MENU12)
    text = "&Edit"
  endwith
  this.MENU12.UNDO = new MENU(this.MENU12)
  with (this.MENU12.UNDO)
    text = "&Undo"
    shortCut = "Ctrl+Z"
  endwith
  this.MENU12.CUT = new MENU(this.MENU12)
  with (this.MENU12.CUT)
    text = "Cu&t"
    shortCut = "Ctrl+X"
```

```
    endwith
    this.MENU12.COPY = new MENU(this.MENU12)
    with (this.MENU12.COPY)
      text = "&Copy"
      shortCut = "Ctrl+C"
    endwith
    this.MENU12.PASTE = new MENU(this.MENU12)
    with (this.MENU12.PASTE)
      text = "&Paste"
      shortCut = "Ctrl+V"
    endwith
    this.MENU17 = new MENU(this)
    with (this.MENU17)
      text = "&Window"
    endwith
    this.MENU11 = new MENU(this)
    with (this.MENU11)
      text = ""
    endwith
    this.windowMenu = this.menu17
    this.editCutMenu = this.menu12.cut
    this.editCopyMenu = this.menu12.copy
    this.editPasteMenu = this.menu12.paste
    this.editUndoMenu = this.menu12.undo
  endclass
```

In the code above, after the menus are defined, certain key menus are assigned to menubar properties which automatically give the menus their required functionality. For example, when this.menu12.copy is assigned to the menubar's *editCopyMenu* property, the copy menu takes on the following characteristics:

- The Copy item remains dimmed unless there is highlighted text in an appropriate object on the form, such as an Entryfield or Editor object.
- When text is highlighted, the Copy item is enabled.
- When the Copy item is selected, the highlighted text is copied to the Windows clipboard.

The remaining *Editmenu* properties function in a similar fashion.

You can modify the preset Edit menu by adding, inserting, or changing item characteristics from the pulldown Menu designer properties sheet.

The *windowMenu* property is useful only with top-level menus on MDI forms. The menu assigned to *windowMenu* will automatically have a menu added to it for each open child window (such as all other active *dBASE SE* windows). This feature provides a means for the user to easily switch windows.

Another important menubar feature is the *onInitMenu* event, which is fired when the menu system is opened. You can use this event to check for certain conditions and then modify your menus accordingly.

If, for example, you offer a Clear All item on your Edit menu, you can set an *onInitMenu(_)* event to disable the item if no tables are open when your form opens. To do that, you could add a pointer to the top of your menu file:

```
NEW MTESTMENU(FormObj,"Root")
CLASS MTESTMENU(FormObj,Name) OF MENUBAR(FormObj,Name)
this.onInitMenu = class::chkClearAll
```

And then create a method to handle the event:

```
function chkClearAll
    if alias( ) == ""
        this.edit.clear_all.enabled = false
    endif
return
```

Changing menu properties on the fly

You'll often need to modify menu properties while a form is open and your application is running.

For example, you might want to change what menu items are offered based on the currently selected control. The following are two event handlers for the *OnGotFocus* and *OnLostFocus* properties of a grid object, respectively. When the grid gets focus, the previously defined Edit menu is enabled; when the grid loses focus, the menu is disabled.

```
function GridMenus              // Assign to OnGotFocus of grid object
    form.Root.Edit.Enabled = true
return
PROCEDURE NoGridMenus           // Assign to OnLostFocus of grid object
    form.Root.Edit.Enabled = false
return
```

Menu and menu item properties, events and methods

Each menu (choose form.root in the Inspector's drop-down object selector list) and menu item (form.root.*itemname*) has its own set of properties, events and methods, only a few of which were applied in the samples above. The following tables describe the primary elements you'll use to define your menus.

Note Where an element is available to only one of the menu classes, the class is noted in the tables below. Otherwise, the element is available to both menubar and popup classes.

Table 8.1 Menubar and popup root properties, events and methods

Property	Description
Alignment *(popup only)*	Lets you align items on your popup menus and submenus. Options are left-aligned, centered, and right-aligned. Default is left-aligned.
EditCopyMenu, EditCutMenu, EditPasteMenu, EditUndoMenu *(menubar only)*	These four built-in objects are available for assignment to items on a preset Edit menu on a pulldown menu (menubar class). To access properties for these objects, click the object's Tool button.
Left *(popup only)*	Sets the position of the left border of the popup. Default is 0.00.
Name	String used to reference the root menu object. Except for Edit and Window menu names (which use the defaults EDIT and WINDOW), default for custom menus is ROOT. A reference to a default item would thus be this.root.menuNN, where NN is a system-assigned item number.
Top *(popup only)*	Sets the position of the top border of the popup. Default is 0.00.
TrackRight *(popup only)*	Logical value (default true). Determines whether popup menu items can be selected with a right mouse click. If set to false, the popup menu is still opened with a right-click, but items must be selected with a left-click.
WindowMenu *(menubar only)*	This built-in object is available for assignment to items on a preset Window menu on a pulldown menu (menubar class). To access properties for the WindowMenu object, click the object's Tool button.
Event	**Description**
onInitMenu	Codeblock or reference to code that executes when the menu is initialized (when its parent form is opened).
Method	**Description**
Open *(popup only)*	Opens the popup menu.
Release	Removes the menu object definition from memory.

Table 8.2 Item properties, events and methods

Property	Description
Checked	Logical value (default false). Adds or removes a checkmark next to the item text.
CheckedBitmap	Graphic file (any supported format) or resource reference. When the menu is run, the graphic you specify appears next to an item to indicate that it is currently selected. Alternative to the Checked property. Works with UncheckedBitmap to offer visual cues to the current "on/off" state of an item.
COPY, CUT, PASTE, or UNDO *(dBASE SE. variable properties; available only to menubar class if preset Edit menu is in place)*	If using a preset Edit menu, these references offer a Tool button to let you view or modify properties for the selected item.
Enabled	Logical value (default true) that dims or activates this item.

Table 8.2 Item properties, events and methods

Property	Description
HelpFile	Specifies the Windows Help file that provides additional information about this item. If you choose to use a Help file, you must also specify a Help topic reference in the HelpId property.
HelpId	Specifies a Help topic that you want to appear when the user presses F1 while selecting this item. If you specify a Windows Help file in the HelpFile property, HelpId is a topic reference within that Help file. You can either specify a context ID number (prefaced by #) or a Help keyword.
Name	String used to reference the item object. Except for Edit and Window menu names, default is MENU*nn*, where *nn* is a system-assigned number.
Separator	Designates a menu item as a separator bar. A separator bar appears as a horizontal line with no text; a user can't choose or give focus to a separator bar. Use separator bars to begin a group of related menu items. You can also define a separator in the Menu or Popup Menu designers by choosing Menu | Insert Separator from the main *dBASE SE* menu.
ShortCut	Specifies a keystroke or keystroke combination the user can press to choose the menu item. Shortcuts, also known as accelerators, provide quick keyboard access to a menu item. For example, you can set the ShortCut for an "Exit without saving" menu item to Ctrl+Q. To define a shortcut key for a menu item, enter it in the *Shortcut* property. For example, to specify the key combination Ctrl+X to exit a menu, enter CTRL-X. Thereafter, when the user presses Ctrl+X, the *OnClick* event occurs automatically. This key combination also appears in the menu title.
StatusMessage	Type text here to display a message in the status bar (if a status bar object is included) of your non-MDI form, or, if you are attaching the menu to an MDI form, in the status bar of your application frame.
Text	Item name, as it appears on the menu. You can also define item names directly in the Menu designer. To specify a letter as the mnemonic key that will be used to access the item, precede the letter in the text string with an ampersand (&). For example, Help menus are usually defined as *&Help*.
UncheckedBitmap	Graphic file (any supported format) or resource reference. When the menu is run, the graphic you specify appears next to the item to indicate that it is not currently selected. Works with CheckedBitmap to offer visual cues to the current "on/off" state of an item.
Event	**Description**
onClick	"Action code" that executes when the item is clicked. If the item is an entry point to a pulldown or submenu, then no code is required for this event. Nor is code required for the items in the preset Edit or Window menus (described earlier in this chapter).
onHelp	Optional code that executes when the user presses F1. Use this to provide user information as an alternative to using the *HelpFile* and *HelpId* properties to define an online Help topic.
Method	**Description**
Release	Removes the object definition from memory.

Toolbar and toolbutton properties, events and methods

Each toolbar and toolbutton has its own set of properties, events and methods, only a few of which were applied in the samples above. The tables on the next few pages describe the primary elements you'll use to define your toolbars.

You can find additional toolBar examples in the samples that come with *dBASE SE* and more detailed coverage of ToolBar class elements, with examples, in Help (search for "class ToolBar" or "class ToolButton").

Tip To inspect all toolbar and toolbutton properties, methods and events, as well as the defaults for each, type four lines like this into the Command window:

```
t1 = new toolbar( )
t2 = new toolbutton( t1 )
inspect( t1 ) // opens the Inspector with toolbar properties visible
```

```
inspect( t2 ) // opens the Inspector with toolbutton properties visible
```

Table 8.3 Toolbar properties, events and methods

Property	Description
Flat	Logical value (default true) which toggles the appearance of buttons on the toolbar from always raised (false) to only raised when the pointer is over a button (true).
Floating	Logical value (default false) that lets you specify your toolbar as docked (false) or floating (true).
imageHeight	Adjusts the default height for all buttons on the toolbar. Since all buttons must have the same height, if ImageHeight is set to 0, all buttons will match the height of the tallest button. If *ImageHeight* is set to a non-zero positive number, images assigned to buttons are either padded (by adding to the button frame) or truncated (by removing pixels from the center of the image or by clipping the edge of the image).
imageWidth	Specifies the width, in pixels, for all buttons on the toolbar.
Left	Specifies the distance from the left side of the screen to the edge of a floating toolbar.
Text	String that appears in the title bar of a floating toolBar.
Top	Specifies the distance from the top of the screen to the top of a floating toolbar.
Visible	Logical property that lets you hide or reveal the toolbar. Default is *true*.
Event	**Description**
onUpdate	Fires when the application containing the toolbar is idle, intended for simple routines that enable, disable or otherwise update the toolbar. Because this event fires continuously when the application is idle, you should avoid coding elaborate, time-consuming routines in this event.
Method	**Description**
Attach	Attach(<form object reference>) establishes communication between the toolbar and the specified form and sets the Form property of the toolbar. Note that a toolbar can be attached to multiple MDI forms or to a single SDI form. For examples, see Help (search for "class ToolBar").
Detach	Detach(<form object reference>) ends communication between the toolbar and the specified form, and closes the toolbar if it is not attached to any other open form.

Table 8.4 Toolbutton properties, events and methods

Property	Description
Bitmap	Graphic file (any supported format) or resource reference that contains one or more images that are to appear on the button.
BitmapOffset	Specifies the distance, in pixels, from the left of the specified Bitmap to the point at which your button graphic begins. This property is only needed when you specify a Bitmap that contain a series of images arranged from left to right. Use with *BitmapWidth* to specify how many pixels to display from the multiple-image Bitmap. Default is 0 (first item in a multiple-image Bitmap).
BitmapWidth	Specifies the number of pixels from the specified Bitmap that you want to display on your button. This property is only needed when you specify a Bitmap that contain a series of images arranged from left to right. Use with *BitmapOffset*, which specifies the starting point of the image you want to display.
Checked	Returns true if the button has its *TwoState* property set to true. Otherwise returns false.
Enabled	Logical value (default true) that specifies whether or not the button responds when clicked. When set to false, the operating system attempts to visually change the button with hatching or a low-contrast version of the bitmap to indicate that the button is not available.
Separator	Logical value that lets you set a vertical line on the toolbar to visually group buttons. If you specify a separator button, only its Visible property has any meaning.
SpeedTip	Specifies the text that appears when the mouse rests over a button for more than one second.
TwoState	Logical value that determines whether the button displays differently when it has been depressed and consequently sets the *Checked* property to true. Default is true.
Visible	Logical value that lets you hide (*false*) or show (*true*) the button. Default is *true*.
Event	**Description**
onClick	"Action code" that executes when the button is clicked.

Chapter

9

Using the Source editor and other code tools

This chapter introduces three tools for working with code in *dBASE SE*:

- The Source editor

  A full-featured, customable ASCII text editor, the main window for editing dBL code (both .PRG files and other project-related files, such as form, report, menu, query, and data module files). The Source editor displays all the code in a file. To view or edit code in the Source editor, press F12 when a design window has focus, or right-click a file in the Navigator, and choose Open In Source Editor. (Not all files have this command available).

  You can have several files open in the editor; each opens on a separate page of the editor, with its name on the page tab. Menus and the toolbar change, as appropriate, depending on the type of file you are editing.

- The Code Builder

  A dialog box available from the Inspector, that lets you conveniently edit code blocks (either commands or expressions). Since code blocks must be on one line, they can be cumbersomely long when you're editing in the Source editor. The Code Block Builder displays the line of code set up in a dialog box, command by command, for easy editing without horizontal scrolling.

- The Command window

  A two-paned command-line interface that lets you experiment with *dBASE SE* commands and expressions, instantly viewing results. You can use the Command window freely at any time. To open it, choose View | Command Window. Your work in the Command window is not saved.

Using the Source editor

The Source editor contains the entire source code for the form, report, menu, query, or data module you're designing. If you're designing several files, the source for each one appears on a different tabbed page. Likewise for a .PRG file, which appears in the same editor.

Note If you want, you can choose not to have tabbed pages in the editor but to open more than one instance of the Source editor, instead. Set this preference in the Editor Properties dialog box, Display page (Properties | Editor Properties).

To open the editor, do one of the following:

- Design a new or existing form, report, menu, query, or data module. Both the design view and the editor open, with the design view having focus. Press F12 to switch focus to the editor. (If in a prior session you closed the editor, it does not open automatically with the design view, but pressing F12 will open it.)
- Right-click a file in the Navigator, and choose Open In Source Editor. (Not all files have this choice.)
- Press F12 when you have a designer open (except the Table designer).

Thereafter, use F12 to toggle between design view and the code page for any given designer file. Changes made in either the Source editor or visual designer are reflected by the other when you move focus between them. Code is automatically compiled when you shift focus to the designer. If an error occurs during compilation, *dBASE SE* displays an error message and points to the offending line in the file.

If an error occurs during runtime, *dBASE SE* displays a dialog box, giving you the opportunity to fix the error. If you cancel the Fix dialog box, then the only copy of the work is in a temporary disk file which is placed on another page (or another instance) of the editor. You can do with this as you wish.

Two-pane window with tree view

The Source editor is a two-pane window:

- The left pane is a tree view showing the hierarchy of the current file (including the *this* object for classes with a constructor). You can enlarge the pane, or you can hide it. To do either one, move the split bar to the left or right, using the mouse. The tree view is dynamically updated during program editing, unless the tree view is closed.

  You can expand the nodes in the tree view pane by clicking the plus signs and collapse the nodes by clicking the minus signs, same as in the Windows Explorer. The expanded or collapsed state of nodes and the selected item are maintained in the tree-view pane when you take actions in the right-hand pane.

  The tree view displays object bitmaps for the standard controls. You can turn this off in the Editor Properties dialog box, Display page (Properties | Editor Properties).

- The right pane contains the code. Click an item in the tree view to highlight the first line of that object in code. Double-click an item in the tree view (or select it and press Tab) to jump to the start of that object in the code.

You cannot use the Source editor to select an object in the Inspector. You must do that in the design window or in the Inspector, itself.

Notes on the Source editor

Here are additional comments on the Source editor. For more information on editing, including keystroke commands, see Help.

- Compared to the former Method editor:

  During stream-out, procedures have a comment generated inline which identifies all methods linked to them. This plus the hierarchy visible in the tree view replaces the function of the "linktext" static text control that appeared in the former Method editor.

  The tree view points at the top and bottom of the source files, showing the equivalent of "header" and "general" in the former Method editor.

- When editing a .PRG file, the Method menu's New Method, Delete Method, and Verify Method commands are available. They work on whatever "method" is at the current cursor position. If no method can be identified, the menu commands are unavailable.

- When designing a form, report, menu, or data module file, three more commands are available on the Method menu: Edit Event, Link Event, and Unlink. Edit Event can generate wrappers for functions or procedures that are not yet part of the source, useful with the new tree view.

- If you attempt to edit a method in a base class, and you elect not to override that method in the derived class, *dBASE SE* opens the source file for that base class in the designer. If it is already opened, it is given focus, and the cursor is positioned at the method.

- When you switch focus from the designer to the editor, it purges the editor's Undo buffers.

- Opening a file named in code:

  Choosing Edit | Open File At Cursor opens a highlighted file, or the file at the cursor position. If no matching file is found, the Open File dialog box appears.

  Choosing Edit | Open File At Cursor when a block of selected text includes more than just the file name, or no file name at all, opens the Open File dialog box.

Files with .WFM, .CFM, .REP, .CRP, .PRG, .CC, and .H extensions are opened in another instance of the editor. Other files are opened in their specific visual designer (for example, .DBF files are opened in the Table designer).

File names with extensions unknown to *dBASE SE* and not registered with Windows produce an error.

Creating a new method

To create a new method, select an event in the Inspector, then click the tool button to the right of the text box. This creates the skeleton of a new method and links it to the event. The Source editor receives focus.

You can write a new method to link to the current event in the Inspector, or you can display the Edit Event dialog box to link the event to an existing method.

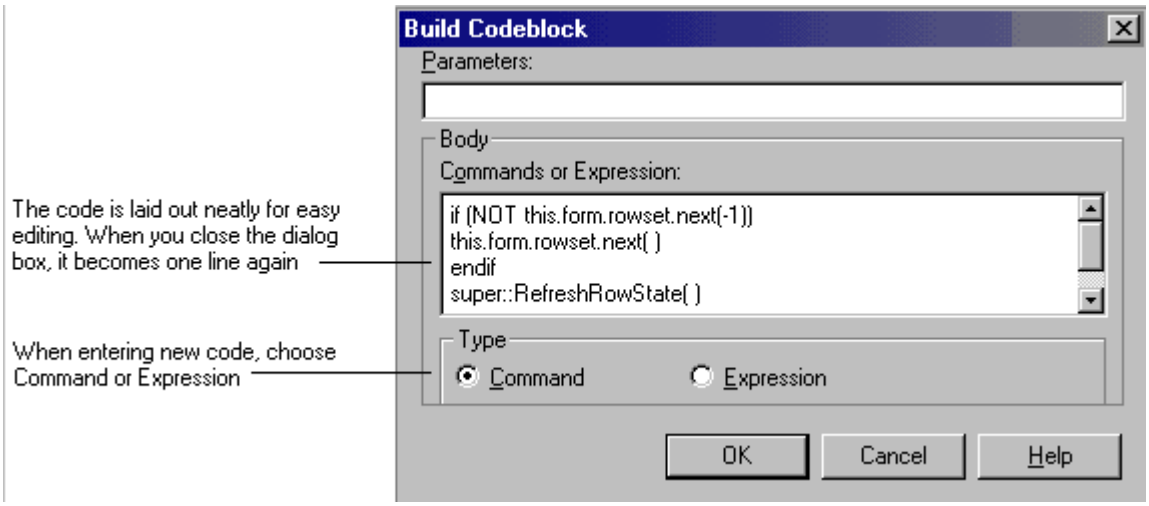
Note A method is a function defined in a class. The Form and Report designers are object-oriented; forms and reports are classes. Therefore all methods are defined and appear in the Source editor with the reserved word function, and are sometimes (loosely) referred to as functions.

The Code Block Builder for editing code blocks

A code block is a data type that can be stored in a variable or property. Code blocks are used in forms and reports to define events or text properties.

Because code blocks cannot span multiple lines, using the Source editor to edit a long code block can be cumbersome. So, when you choose to, you can open the Code Block Builder, which temporarily lays out the code one command per line, with the parameters in a separate text box.

Figure 9.1 Code Block Builder



Edit what you need to, and choose OK. The code block appears in your code as one line again.

You don't have to open the Code Block Builder if you don't want to. You can edit directly in the Inspector or the Source editor.

To create or edit a codeblock

To create a new codeblock for an event,

1. Select CodeBlock from the event's drop-down Type list.
2. Click the wrench tool beside the event to open the Code Block Builder.

To create a new codeblock for a Text control,

1. Select its *text* property in the Inspector.

2 Select CodeBlock from the *text* property's drop-down Type list.

3 Click the wrench tool beside the property to open the Code Block Builder.

Editing an existing code block

If you have an existing code block you want to edit (it will be in an event or the *text* property of a Text control), you can open the Code Block Builder dialog box in these ways:

- For an event,
  - Select the event in the Inspector, and then select the wrench tool beside it, or
  - Choose Method | Edit Event, and if a code block is already associated with the event, the Code Block Builder opens (otherwise, the Source editor opens).
- For a Text control, select its *text* property in the Inspector, and then select the wrench tool beside it.

Make your changes in the Parameters text box and in the Commands Or Expression text box.

When you click OK, *dBASE SE* checks the syntax of the code block. If an error exists, *dBASE SE* attempts to repair the error. If it can't, a warning message box notifies you of the error, and the Code Block Builder stays open so you can fix the error. Focus is placed on the text box where the error occurs: Parameters or Commands Or Expression. If you don't know how to fix the error, you can choose Cancel and *dBASE SE* keeps the previous code.

If the code block is error-free, then the Code Block Builder closes. The code block is condensed back into one line and displayed in the appropriate line in the Inspector. The indentations and carriage returns are removed.

The Command window

The Command window is used to directly execute one-line *dBASE SE* commands. It is handy for testing simple expressions and immediately seeing the results in the results pane. (It is the *dBASE SE* counterpart to the dot prompt found in dBASE IV and earlier DOS versions of dBASE.)

Note The Command window is for temporary work only; you cannot save your work. However, you can copy the contents of the window or drag and drop to a source file. You can also print the contents (select what you want to print, and choose File | Print).

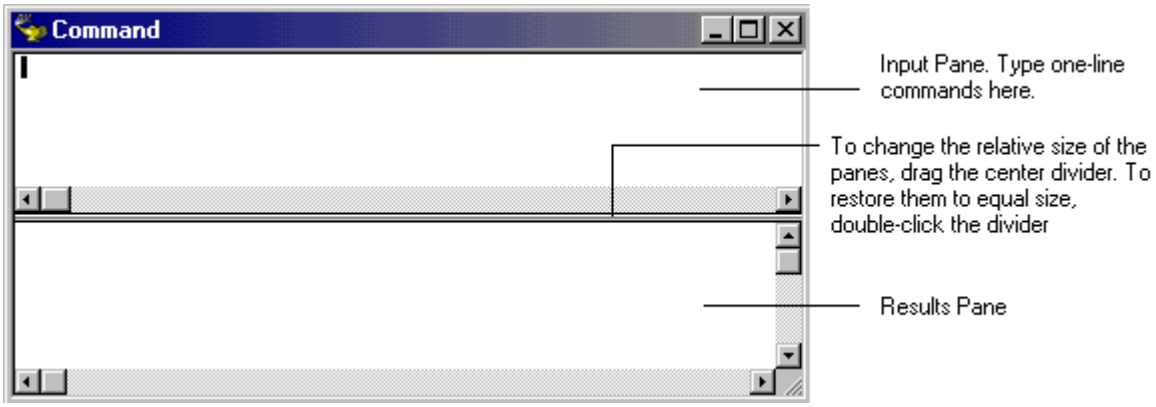
To use your own functions in the Command window, you must first load them:

```
set procedure to <filename> additive
```

To open the Command window, choose View | Command Window.

The Command window has two panes, as shown in Figure 9.2

Figure 9.2 The Command window



Command window panes have specific functions:

- The input pane is where you enter interactive commands You might use the input pane in this way if you find typing commands easier or faster than using the mouse and menus.

  The input pane echoes your actions in the *dBASE SE* interface, keeping a history of the commands you've executed. For example, when you create a new table by double-clicking the Untitled table icon, the Command window shows that you've executed a CREATE command.

- The results pane is where your command output appears, unless your commands create or call separate windows. It is also the default destination for the output of many programs. The results pane retains the last 100 lines.

To change the relative size of the two panes, drag the center divider. To restore them to equal sizes, double-click the divider.

To clear the contents of the input pane, close the window and select View | Command Window. To clear the results pane, choose Edit | Clear All Results.

Typing and executing commands

To execute a command, type it in the input pane and press Enter. You can also click the Execute Selection button on the toolbar or choose Edit | Execute Selection. You can delete commands like any other text. The commands you enter in the Command window remain there until you close the window or exit *dBASE SE*.

Because pressing Enter executes the command line, you must press the down-arrow key to enter more than one line into the Command window. The maximum number of characters per line is configurable in the Editor Properties dialog box. The maximum number of lines the input pane can hold is limited by virtual memory.

The command line defaults to insert mode, as indicated in the status bar. To switch between insert and overwrite modes, press the Ins key.

Executing a block of commands

In addition to typing multiple command lines, you can paste lines of command text from another source. You can also execute a block of command lines, provided the block does not contain nested structures or methods.

To execute more than one line of text in the input pane, select the lines with the mouse or use Shift and the arrow keys. Press Enter, or click the Run button on the toolbar, or choose Edit | Execute Selection.

Reusing commands

To reuse commands you've already entered in the input pane,

1. Scroll the window, if necessary, to display the commands you want.
2. Click the command line you want, or select a block of commands.
3. Execute the command (or commands) by pressing Enter, clicking the Run button, or choosing Edit | Execute Selection.

Editing in the Command window

Edit text in the input pane as you would in a text editor, using standard editing keys such as Backspace and Delete, and the Edit menu commands. Use the Edit | Search commands to search for and replace text in the input pane.

The command line is the line in the input pane containing the insertion point.

You can cut or paste code from Help or use commands from a program file by opening the file, copying the commands, and pasting them into the Command window. After the commands are in the Command window, you can test or modify them. The sample files provided with *dBASE SE* are a good source of working commands.

Saving commands into programs

If the input pane contains dBL code you want to use again, you can copy and paste it into a new program (.PRG) file or insert it into an existing program file.

You can also mark a block and choose Edit | Copy To File. *dBASE SE* displays the Copy To File dialog box so you can name the new file for the selected text. By default, the file has a .PRG extension, but you can change it to another extension. If a block is not marked, the Edit | Copy to File is grayed out.

Chapter

10

Debugging applications

Debugging is the process of locating and eliminating errors—bugs—from an application. Use the *dBASE SE* Debugger to repair broken code and resolve problems in your forms, reports and programs.

With the Debugger you can:

- Load and debug multiple files.
- Control program execution by stepping through an entire program line by line or skipping to defined breakpoints.
- Monitor the values of variables, fields, and objects. You can even make temporary changes for testing purposes, and then update your code using the *dBASE SE* Source editor.
- View subroutines (methods, procedures, and functions) that the main program calls, and track the points at which each is called.
- Stop program execution at any point, or run full-speed to the cursor position.
- Run the Debugger as a standalone application to debug compiled programs.

Types of bugs

The two most common types of bugs are syntactical and runtime errors.

Errors in syntax include such oversights as misplaced braces or *endif* statements, and are generally caught by the compiler before you even get to the debug stage. If you run uncompiled code through the Debugger, however, it will easily catch any syntactical errors.

Runtime errors, such as calls to non-existent tables, are also quickly exposed by the Debugger, which automatically halts at the offending reference.

When you are stopped by any error, you can either cancel or suspend further execution of the program, ignore the error and continue running the code through the Debugger, or note the problem, open your *dBASE SE* Source editor, fix the code, and then return to the Debugger to check for additional errors.

The third, and least obvious type of bug is an error in program logic, and these are not detected so easily. If, for example, your program includes a method that is supposed to execute after a certain event, but the event is bypassed, you may need to use all the debugging power described in this chapter to track down and correct the problem.

Using the Debugger to monitor execution

There are three ways you can use the Debugger to monitor how your program executes:

- Run the program locally from the *dBASE SE* integrated development environment (IDE). Running from the IDE is convenient for checking code syntax or various parts of your program while you develop it.

- Compile your application, then debug it by typing debug <programname.exe> into the Command window. This method provides a "real world" test, showing how your program accesses tables, for example. After running your tests, you can use the *dBASE SE* Source editor to make any needed adjustments.
- Run the Debugger as a standalone application, set breakpoints, then run your program from *dBASE SE*. When the program reaches a breakpoint, control is handed over to the Debugger.

General debugging procedure

This section gives you a quick overview of debugging procedures. The process is examined in greater depth in subsequent sections.

The Debugger is always available whenever you run into an error when in Run mode. To open the Debugger and deal with the error on the spot, you only need to click the Debug button in the error dialog. When the Debugger opens, you can then proceed from step 2 in the instructions below.

Alternatively, you can run it before running your program, then choose a program to debug. Here's how:

1. Start the *dBASE SE* Debugger by right clicking on a source file in the Navigator and choosing Debug. The program's code appears in the Debugger's Source window, under a tab with the file's name. If you open multiple programs, each appears under its own labeled tab.
2. You can then configure a number of options:
   - If you intend to pause the execution of the program at certain points or isolate a section of the code for test-fix-test cycles, set breakpoints by double-clicking the Stop Hand pointer at the left of the line before which you want a breakpoint.
   - Open any tool windows you intend to use, for example, to watch variables.
3. If you set breakpoints, or if any kind of error occurs
   - The Error Message box displays the *dBASE SE* error message. Click OK.
   - The Debugger's Source window appears. The offending line immediately precedes the blue-highlighted line.
   - Check for the more obvious and typical errors: a misspelling or missing punctuation or spaces.
   - Inspect your public or private variables and expressions by holding the cursor over a variable until a speedtip appears with the variable's current value. This can give you a clue about what went wrong. You can also view all variables in the Variable tool window or set watchpoints for particular expressions and monitor these in the Watch tool window. The debugger, however, won't find the value of a LOCAL variable.
   - If the form does not appear quite the way you intended when you created it in the Designer, try adjusting some of the display parameters in one of the tool windows. If this works, make the same changes permanently in the code by editing the program in the *dBASE SE* Source editor.
   - For other types of errors, return to the main *dBASE SE* Source editor, locate the offending line and make your corrections. Save the file, then restart your program to check the results.
4. If your application initially appears stable and displays properly, proceed by testing each of its features, entering or editing values, submitting changes to the server, or requesting updates. Click each button and interact with the program in every possible way. Errors generated from these interactions might require you to step into subroutines and again check variable values at each step.

Note Selecting File | Exit from the Debugger menu, the Debugger closes and program execution continues. In prior dBASE versions, selecting File | Exit while in the Debugger, halted program execution and returned to the Command Window. In *dBASE SE* program execution is halted by using the Stop button on the Tool Bar.

Whenever you need to run the program again from the top, simply switch focus to the Debugger, then switch back to your main program window.

Debugging runtime applications

To debug compiled programs, simply run the Debugger as a standalone application, set breakpoints, then run your program from *dBASE SE* (not from the Debugger).

When the program reaches a breakpoint, control is handed over to the Debugger.

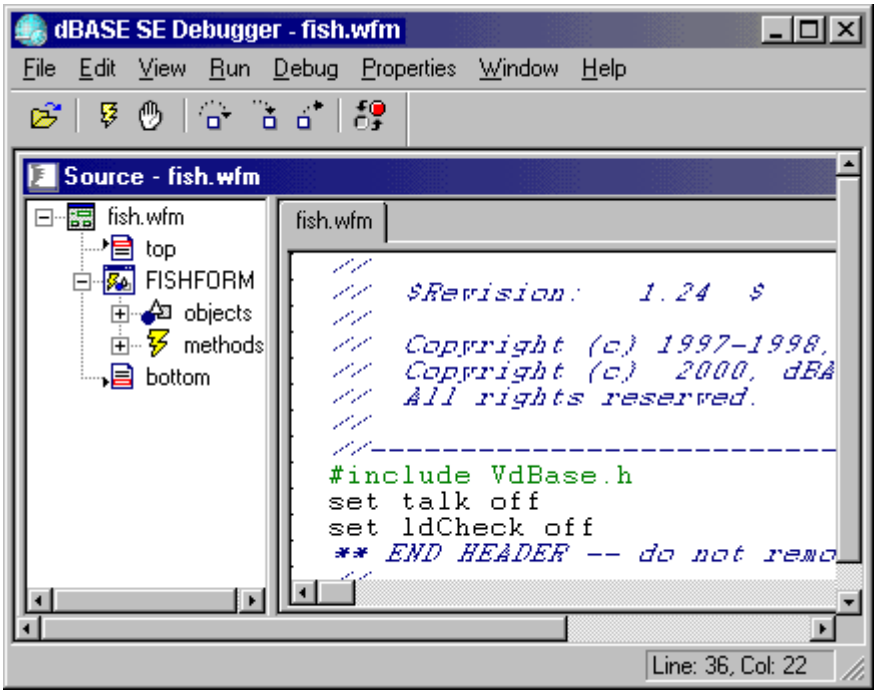
The Source window

The Source window is the main Debugger window. The code is read-only. The left pane of the Source window shows a hierarchical view of the objects in your code.

You can use the Source window to step through code execution line by line, to highlight breakpoints and identify the location of errors.

However, you cannot edit or repair your code in the Debugger's Source window; to do that, you have to use the *dBASE SE* Source editor.

Figure 10.1 Source Window



As your program runs, source code scrolls to the line about to receive program control. You can scroll the source code without affecting program execution; the next command to be executed remains highlighted regardless of where the cursor is in the Source window.

Your program runs in the background while the Debugger retains focus. You can interact with your program by moving focus to it (use Alt+Tab to switch to it, or minimize the Debugger and click your program window). While you interact with your program, its code continues to scroll in the Source window. If you close the application, your program's code remains open in the Source window.

You also use the Source window to locate and set breakpoints. If an error occurs, the Source window automatically scrolls to the offending line, highlighting (in blue) the line immediately following the offending line.

To correct any detected errors, return to the *dBASE SE* Source editor, load your file, locate the offending line (using the *dBASE SE* Source editor's line-numbering feature) and fix the error.

To locate and move to a line number in the Source window

1. Choose Edit | Go To Line Number from the Debugger menu or right-click the Source window and choose Go To Line from the popup menu.
2. Specify the number in the Go To Line Number dialog box.

Using Go To Line Number to move to a line number in the Source window doesn't affect program execution, which remains paused at the last line you stepped or traced to, or at the last executed breakpoint.

To find a text string in the current program file

1. Choose Edit | Find Text from the Debugger menu or right-click the Source window and choose Find Text from the popup menu.
2. Specify the text to find in the Find Text dialog box.

The search begins from the current position of the cursor and is case-insensitive. If the text is found, the Source window scrolls to the line containing the text, and the cursor moves to the beginning of the text. To find additional occurrences of the same text, choose Edit | Find Text from the Debugger menu or Find Text from the Source window popup menu.

The Debugger tool windows

The Debugger's View menu offers access to four tool windows, described below, that help you track program elements during a debugging session.

Variables

Lists variables found in the currently selected program. You can limit the extent of the variable search by deselecting options in the Debugger Properties dialog box (Properties | Debugger Properties).

Watches

The Watch tool window lets you specify particular variables, fields, and expressions that you would like to watch as the program code executes. The window shows the changing values for the watched items as you test the program (by clicking buttons, sending queries, and so on).

Call Stack

This tool window displays a list of subroutines called by the current program. Each call to a separate program file (or module) is displayed as it occurs, showing its line number, function name, and path name.

Trace

Displays the output that appears in the Output panel of the Command window.

Docking the Debugger tool windows

To dock a tool window in the Debugger, click the frame of the window and drag to the side of the Source window where you want to fix the window's position. The small tool window enlarges to fit the side or bottom of the Source window.

To undock a tool window, click the frame of the tool window and drag to the center of the screen until the small outline appears, then release.

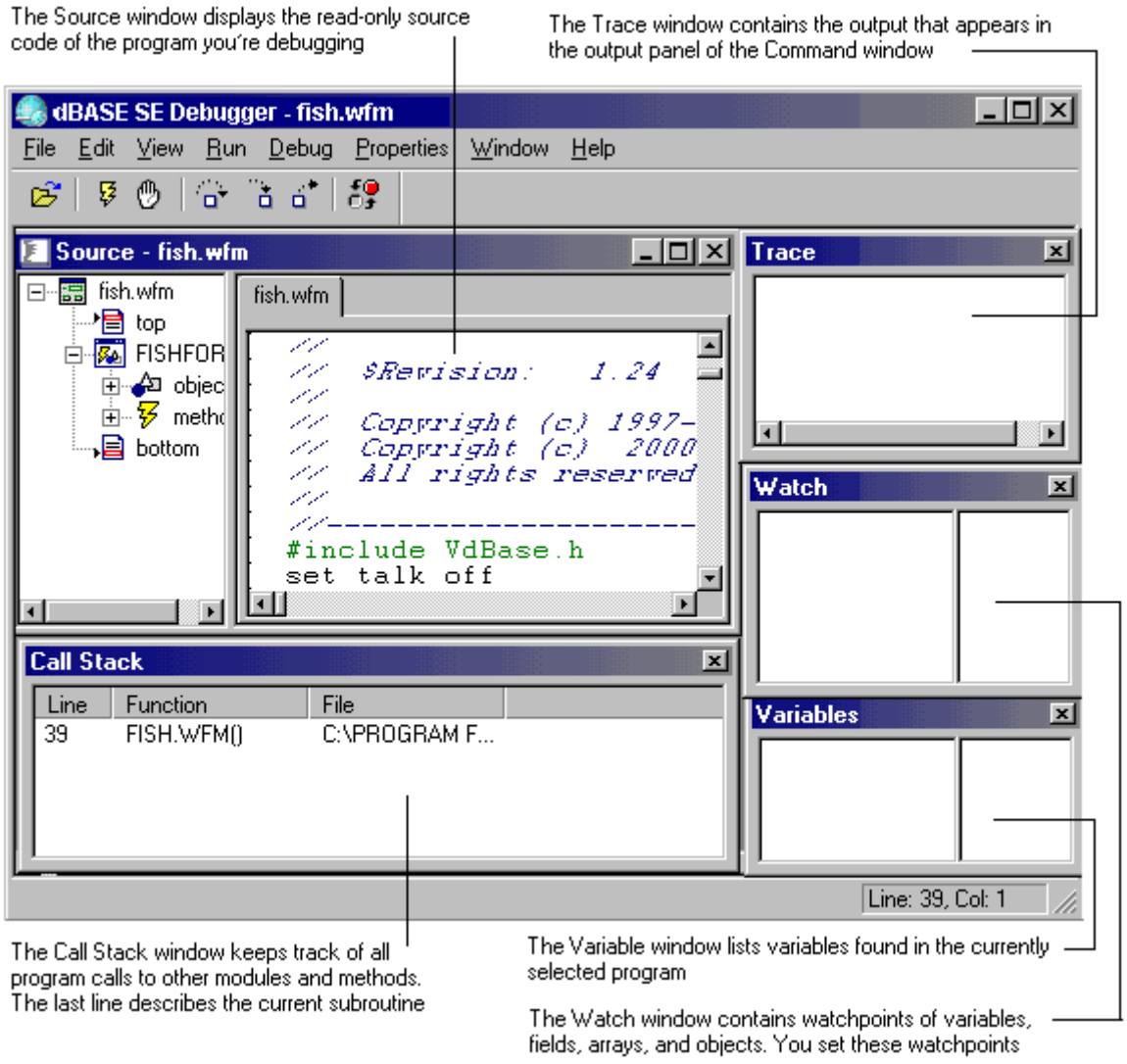
To disable docking, choose View | Tool Window Properties and uncheck the tool windows for which you want to disable docking. See Figure 10.2 on page 10-5

Excluding variable types

To exclude variable types from the tracking process, choose Properties | Debugger Properties to open the Debugger Properties dialog box, then deselect items from the variables type group.

The Debugger Properties dialog box also permits you to hide exceptions during a debugging session.

Figure 10.2 Debugger tool windows, docked



Controlling program execution

You can control program execution in the Debugger using the following commands and procedures, all of which are available from the Run and Debug menus.

Table 10.1 Methods of controlling execution in the Debugger

Method	Type of program execution control
Run	Runs the program, stopping at each error to display an error message. The line after the offending line is highlighted in blue.
Stop	Stops execution of the program and closes the debugger.
Run To Cursor	Executes lines from the current position to the cursor location and stops there.
Step Over	Skips subroutines (any called functions, methods, or procedures).
Step Into	Shows line-by-line execution of subroutines, stopping at the end of each subroutine. You can step into another nested subroutine.
Step Out	Returns display of line-by-line execution to the preceding level of the program.
Break on next executable line	Causes program execution to stop at the next executable line, regardless of where execution starts.

Table 10.1 Methods of controlling execution in the Debugger

Method	Type of program execution control
Using breakpoints	Breakpoints are specific points in the program that you set to stop execution, letting you assess the situation. You can isolate a section of code for closer study by placing a breakpoint at the beginning and end of the section, then running that section repeatedly. You can further subdivide the section by adding more breakpoints.
Watching variables	You can see the current real-time value of public and private variable by holding the cursor over it until a speedtip appears. You can select particular variables (from the Variable tool window which shows all the variables in the program) and watch how their values change in the Watch tool window.

Stepping in the Debugger

Stepping executes your program line by line, pausing after each line so you can evaluate the result.

If you are confident about a block of code, you don't have to step through it again and again to get to the uncertain areas. You can, for example, step over any lines that call subroutines (including methods and expressions).

For example, if you have already determined that no bugs exist in a particular subroutine, select the line that calls the subroutine, click the Step Over button in the toolbar, and run the program. The Debugger steps over the execution of the subroutine so you can focus on the rest of the program. The call to the subroutine still occurs, and the stepped-over subroutine still executes; you just don't see the line-by-line execution in the Source window. The Debugger then stops at the line following the stepped-over subroutine.

On the other hand, if you want to check out what a subroutine is doing, you can *step into* it, and further, step into any nested subroutines as well. Then you can *step out* from the subroutine and return to the main program level.

Commands for stepping over, stepping in, and stepping out of subroutines are available from the Run menu and toolbar.

Using breakpoints

A breakpoint stops program execution so you can evaluate variables, fields, arrays, objects, and expressions; change the value of variables, arrays, and objects; and check what subroutine the program is in. Using breakpoints lets you run the program at full speed until it comes to a problem area; breakpoints give you an alternative to stepping through the entire program.

When you step or trace through a program, you're essentially breaking at each line. However, once you are certain that no bugs exist in certain parts of your program, there is no need to repeatedly step through each line; instead, set breakpoints at crucial places where the code is less certain, then run the program at full speed and evaluate program values at the breakpoints.

If, for example, you suspect a bug in occurs at one particular place, such as when a subroutine is called, you could set a breakpoint at the line that calls the suspect subroutine. You could then *step into* the called method or function.

Setting and removing breakpoints

To set a breakpoint, select a line of code in the Source window and either

- Move the pointer to the left of the command line where you want to enter a breakpoint. When the pointer changes to a Stop sign, double-click. The line is then highlighted in red.

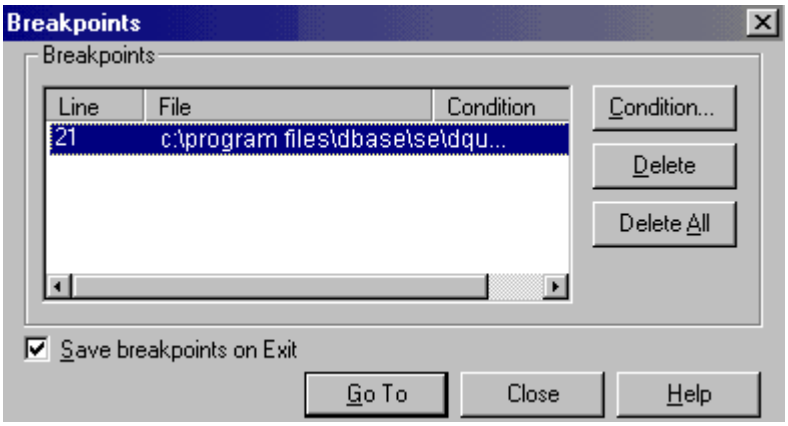
  To remove the breakpoint, double-click the highlighted line again.
- Press Ctrl+B to add a breakpoint to the selected line of code or to remove a breakpoint from the selected line.
- Choose Debug | Toggle Breakpoint from the Debugger menu (or from the Source window's popup menu).

To remove all current breakpoints, choose Debug | Delete All Breakpoints.

Working with breakpoints

To see a list of breakpoints in a program, choose Debug | Breakpoints.

Figure 10.3 Breakpoint window



You can use the Breakpoints list to keep track of existing breakpoints in all open modules. The line number of each breakpoint is listed next to the path name of the program in which it appears.

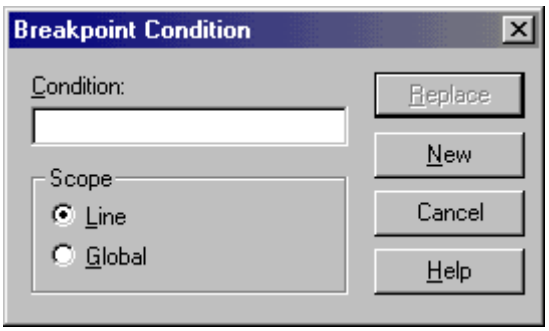
To delete a breakpoint, select it and click the Delete button.

To delete all breakpoints, click the Delete All button.

To go to any breakpoint, simply double-click the line number of the desired breakpoint (or select it and click Go To). The Breakpoint window closes and the Source window opens to the selected breakpoint.

To conditionally set breakpoints, click Condition to open the Breakpoint Condition dialog box, and type in an expression—such as the value of a variable, a global condition, or any conditional expression that defines a condition in which the current breakpoints should be active. If the condition is not met, breakpoints are ignored.

Figure 10.4 Breakpoint Condition dialog box



- **Line** Click the Line radio button to specify a condition for the line currently selected in the Breakpoint window. The conditional expression you enter applies only to the selected breakpoint line.
- **Global** Click the Global radio button to enter a conditional expression for the entire program. For example, you might have noticed that when a global variable reaches a certain value, say 10, the program becomes unstable, but until then everything works fine. To test your observation, you could set the condition *x=10* to force all breakpoints to activate when the global variable *x* reaches 10.

Running a program at full speed from the Debugger

The alternative to stepping and tracing, which examine your code line-by-line, is to debug at full speed. This lets you skip areas that you know are bug-free and concentrate on suspected problem areas.

If you set breakpoints in your program, you can debug at full speed and execute to the first breakpoint. You can then decide whether to continue running the program from the breakpoint or to proceed by stepping over or into subroutines.

To debug at full speed, either

- Click the Run toolbar button
- Choose Run | Run from the Debugger menu, or
- Press F9

Running to cursor position

You can also run at full speed until execution reaches the current cursor position in the Source window. To try this approach, choose Run | Run To Cursor.

Stopping program execution

In addition to using breakpoints and stepping techniques, you can halt execution of a running program any time by

- Clicking the Stop button on the toolbar
- Choosing Run | Stop from the Debugger menu

Debugging event handlers

You can use the same basic techniques to debug event handlers as you do for any other code sections.

These are the general steps for approaching error handlers:

1. Open the Debugger and load the program containing the event handler code.
2. Set a breakpoint.
3. Run the program, either at full speed, or by stepping or tracing through it.
4. When the program opens a form, the form gets focus. Interact with the form to trigger the event associated with the event handler. For example, if the event handler is an *onClick* method for a pushbutton, click the pushbutton.

When the event handler reaches the line or condition specified by the breakpoint, execution stops and focus returns to the Debugger. You can then inspect, step, or perform other debugging tasks.

Viewing and using the Call Stack

The Call Stack window tracks calls made to methods or functions, whether inside or outside of the running program. The list includes the line number, function name, and file name to which the calls were made.

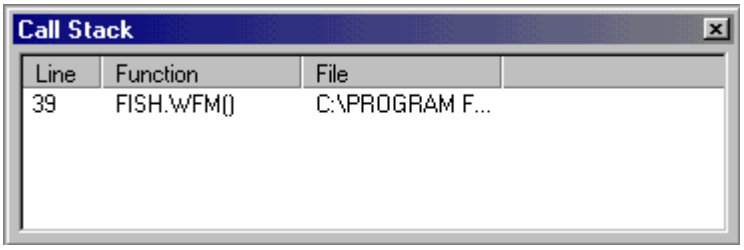
The last line in the Call Stack list is always a reference to the main program level.

Note that if you step through or over nested subroutines, calls within the nested subroutines are removed when execution steps out again.

Otherwise, the Call Stack list is updated whenever program execution pauses.

You can also use the Call Stack list to quickly shift Source window focus to any subroutine call. To do that, select a subroutine in the Call Stack, right-click, and choose Go To Source Line from the popup menu. Note that though the Source window is refocused to the calling line, the current execution point remains where it was, and will continue from that point if you resume program execution though the Source window is refocused to the calling line, the current execution point remains where it was, and will continue from that point if you resume program execution.

Figure 10.5 Call Stack window



Watching expressions

The Debugger's Watch window lets you monitor expression execution and results. You can even use it to temporarily change and retest variables. It can help you detect such problems as the assignment of incorrect values to variables, or assignment of values at the wrong points.

Watchpoints are evaluated and their current values displayed in the Watch window whenever program execution is paused.

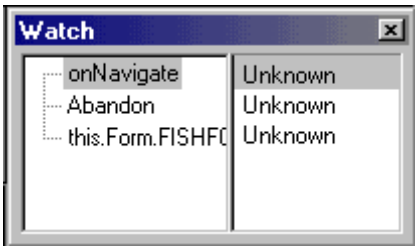
Adding watch points

To add a watch point, do one of the following:

- Choose Debug | Add Watch from the Debugger menu.
- Right-click the Watch window and choose Add Watch from the popup menu.
- Press Ctrl+W or Ctrl+A.

The selected expression appears in the Watch window.

Figure 10.6 Watch window



Note If you haven't already run the program past a point where variables, fields, arrays, or objects in the selected expression are initialized, "unknown" is displayed to the right of the expression.

Editing watch points

To edit an existing watch point,

1. Select the watch point in the Watch window.
2. Choose Debug | Edit Watch Value or Debug | Edit Watch Name from the Debugger menu. Alternatively, right-click the Watch window and select one of the editing commands in the popup menu.

Changing watchpoint values

In addition to watching expressions, you can temporarily change the value of a variable. If, for example, a variable is receiving a correct value, but at the wrong point, you could

- Add a watch point for the variable.

- Pause program execution by setting up a breakpoint at the line where the correct value is supposed to be assigned to the variable.
- Use the Watch window to directly assign the correct value to the variable.

It's important to remember that this sort of testing does not result in a permanent code change. It is only intended to let you examine how your program behaves when the correct variable value is assigned. If the test is successful, you can open the *dBASE SE* Source editor to permanently correct the part of the program that was responsible for returning the wrong value.

Chapter

11

SQL designer

The *dBASE SE* SQL designer lets you create, edit, and execute SQL queries on any supported data source.

You needn't be an SQL expert to use the SQL designer. In fact, you can use the tool to learn SQL by examining the results and structure of the queries you create. You can start with simple SELECT statements and progress to the most complex table joins and grouped queries—all of which can be easily constructed in the designer's table pane and notebook pages.

When your statements are structured to produce the results you want, you can save them to disk, edit them with the Source editor, or use them as templates in the SQL Property Builder.

You can also add your saved queries to a *dBASE SE* program directly by assigning an .SQL file to the SQL property of a Query object, or by copying statements from your query files directly into your code.

Opening the SQL designer

For new queries

You can open the SQL designer to create a new query by using any of the following methods.

- Menu: Choose File | New (Alt+FNQ).
- Navigator: SQL tab, double-click the Untitled icon.
- Command window: Type

  ```
  create query
  ```

To view or modify an existing query

You can open the SQL designer to load and review or modify an existing query by using any of the following methods.

- Menu: Choose File | Open (Alt+FO or Ctrl+O) for the Open File dialog, then choose SQL from the Files of Type list, and locate your saved .SQL file.
- Navigator: SQL tab, double-click the icon for an existing query file or select it and either press F2 or click the Design button on the toolbar.
- Command window: Type

  ```
  modify query <queryfile.sql>
  ```

  where <queryfile.sql> is a saved query file. Note that if the file is not in the current directory (as shown in the Look In box in the Navigator), you must also include the full path to the file.

SQL designer elements

The SQL designer is comprised of two main sections:

1. The **table pane**, the upper portion of the designer, where tables and fields are depicted. This area lets you select and deselect data for your queries. You can also use these table listings to quickly create joins.
2. The **query notebook** is the tabbed lower portion of the designer. Each page of the notebook offers a grid layout for specifying query parameters and options.

The notebook pages are:

- **Criteria**. This page enables you to create summary data. It also allows you to specify a customized output name for a field or summary data in the query results.
- **Selection**. Where you specify which rows of data are to be included in the query results.
- **Grouping**. Use this page to create a grouped query. A grouped query groups the data from the source tables and produces a single summary row for each row group.
- **Group Criteria**. Enables you to specify group selection criteria used in the HAVING clause that SQL designer adds to the query. A HAVING clause selects and rejects row groups.
- **Sorting**. Where you specify a sort order for the query.
- **Joins**. This page lets you create multi-table SQL queries (joins).

Interacting with the Source editor

Like other *dBASE SE* designers, the SQL designer is a two-way tool. If you open a saved .SQL file in the designer, or save a new query, any change you make thereafter is immediately reflected in the source file (your.SQL file), and vice versa.

To view or edit your source file any time, press F12. The source code appears in the Source editor.

Entering data in the SQL designer

As with data entry in the Inspector, it's important to remember to press Enter after completing a formula or entering data in an SQL designer grid. If you don't, the formula or data may not register.

Running a query from the SQL designer

When you're designing a query in the SQL designer, you can check the results of your query any time by pressing F2 or clicking the Run button on the toolbar. Results are displayed in a table (default is grid view).

When viewing your query results you have all of the standard table-editing and navigational tools at your disposal. For more information on these tools, see Chapter 15, "Editing table data".

To return to the SQL designer from the results table, press Shift+F2 or click the Design button on the toolbar.

Putting your queries to work

There are a number of ways to incorporate your .SQL files into your forms, reports and applications.

The quickest way is to add the name of your .SQL file to the SQL property of a Query object.

To do that,

1. Use the Form or Report designer to open a new form or report.
2. Choose the Data Access tab on the Component palette, and drag a Query object onto your form.
3. If it's not already in view, press F11 to display the Inspector.
4. In the Query object's SQL property, type:

   ```
   @mysql.sql
   ```

   where mysql.sql is the name of your query file.

Using your .SQL files with the SQL Property Builder

You can use your saved SQL statements as templates in the SQL Property Builder.

To do that, use steps 1-3 from the instructions above, but this time click the tool button next to the SQL property. This opens the SQL Property Builder.

Choose the second option in the builder, SQL Statement File, then click the tool button next to that field. A file search dialog opens to let you locate your saved .SQL file.

When you load the file, your SQL statement appears in the builder's edit window. You can then either edit the statement or click OK to attach it to the Query object.

Even if you edit the statement, your original .SQL file remains intact, available for modification or updating in the SQL designer or service as a template for another Query object.

Looking at the table pane

The upper portion of the SQL designer, the table pane, is where tables and fields are depicted. This area lets you select and deselect data for your queries. You can also use these table listings to quickly create joins.

About the table boxes

Each table appears as a scrolling, sizable, collapsible window containing a header with the table name and a list box containing all of the table's fields.

Next to each field and table name is a check box. If a table name has a dark blue check mark, all fields from that table will be in the query. Otherwise, only fields that are checked will be in the query. If at least one, but not all, fields are checked, the table name will have a light gray check mark.

Dragging the mouse over a table name shows you the full name of the table. For example, the full name for a customer table might be ":mydb:Customer.dbf".

To view tables in a collapsed mode where only the table name appears, click on the minimize button next to the table name.

Tables can be joined by dragging a field from one table window to a field in another table window. When two tables are joined, a join line appears that links the joined tables. For more information on joins, see "Creating joins in the SQL designer" on page 11-12.

Adding tables in the SQL designer

Tables are added to a query in the SQL designer by simply adding them to the table pane. In the table pane you can select some or all fields of one or more tables to be included in the result set. You can also graphically join one table to another. A table may be added more than once.

To add a table, either click the Add Table icon on the toolbar, either press Ctrl+A or choose SQL | Add Table. When the Add Table dialog appears, select a table from the list and press Add. You may also to use the Look In list and browse folders to locate additional tables.

When you're finished adding tables, press Close.

Renaming a table

1. Right-click on the table window and select Edit Table Alias.
2. Type an alias for the table name in the edit box.

Removing a table

1. Right-click on the table window and select Remove Table.

2 A table may also be removed by pressing the Delete button when the table name has focus.

Selecting fields in the SQL designer

In the SQL designer, each table window in the table pane has a table and field name with a check box that allows you to select some or all fields to be included in the result set.

Selecting all fields in a table

In a table window, click on the table name's check box to add a blue check mark. A blue check mark indicates that all fields are selected. Uncheck this box to remove all fields from the query. If the check box has a gray check mark, only a portion of the fields have been selected to appear in the result set.

The F6 key and spacebar also enable you to toggle field selections.

Selecting all fields automatically adds them to the Selection page.

Selecting individual fields in a table

Check each field you wish to appear in the result set. When a field is checked and unchecked, it is automatically added to and removed from the Selection page.

Fields can also be selected by dragging them from the table window and dropping them on the Selection page grid.

Reordering selected fields

In the left gray column of the grid, drag the row and drop it at the new location.

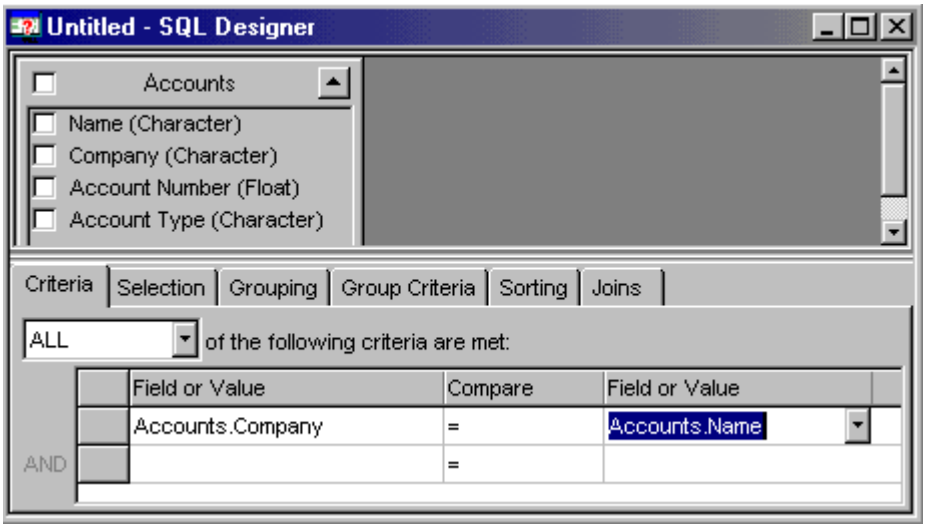
Criteria page (SQL designer)

In the SQL designer, the Criteria page allows you to specify selection criteria that the query will use to include only certain rows of data in the query results. Adding selection criteria to this page adds a WHERE clause to the query. The criteria can be either a simple expression, an SQL expression, or an exists clause.

- The grid contains the selection criteria by which the query will exclude rows of data.
- The drop-down list box specifies whether all, any, none, or not all of the criteria apply

.

Figure 11.1 SQL Designer Criteria Page



Deleting a row

After selecting the row to delete, right-click and choose Delete Row from the context menu.

Adding selection criteria in the SQL designer

Selection criteria in a query specifies which rows of data are included in the query results. In the SQL designer, you enter selection criteria into the Criteria page of the Query notebook.

Specifying selection criteria

1. In the Criteria page grid, choose the type of criteria you want by right-clicking in the grid and selecting the appropriate criteria from the grid's context menu. You may choose from Simple Equation, SQL Expression, and EXISTS. Each type is described below.
2. Enter criteria into the row according to the type of criteria you have chosen.

Simple Equation

A simple equation compares the values of two values for each row of data. For example,

```
CustNo >= 1000
```

The values can be either a field name, constant value or any valid SQL expression. String and date constant values must be surrounded by single quotes.

When defining a simple equation the grid has three columns: Field or Value, Compare, and Field or Value.

To enter a simple equation:

1. Enter the first field or value you wish to compare into the first Field or Value column. This can be done by either dragging a field from a table window in the table pane and dropping it onto the Field or Value column, selecting a field from the drop-down list box, or entering a constant value or valid SQL expression into the Field or Values column.
2. Select the appropriate comparison operator from the Compare column drop-down list box. You can choose from =, >, <, >=, <=, <>, LIKE, IN, BETWEEN, NOT BETWEEN, IS NULL, or IS NOT NULL.

3 Enter the field or value you wish to compare to the first into the second Field or Value column. This can be done by either dragging a field from a table window in the table pane and dropping it onto the Field or Value column, selecting a field from the drop-down list box, or entering a constant value or valid SQL expression into the Field or Values column.

Remember to press Enter after entering the last element of your equation.

SQL Expression

Enter an SQL expression directly into the SQL Expression column. For example,

```
((CustNo < 2000) OR (CustNo > 3000))
```

String and date constant values must be surrounded by single quotes.

EXISTS Clause

Adding an EXISTS clause returns True when the subquery produces at least one row of query results.

When a row has this type of selection criteria, the row in the grid has two columns: Operator and SQL Expression. Select EXISTS from the Operator column. You can now enter an SQL expression to see if any rows are produced.

The following example returns all the companies who have placed orders:

```
SELECT Company FROM Customer.dbf WHERE EXISTS (SELECT * FROM Orders WHERE Orders.CustNo =
Customer.CustNo)
```

In the preceding example, you would enter the statement following the ‘EXISTS’ into the SQL Expression column.

String and date constant values must be surrounded by single quotes.

Combining selection criteria

Row info

In the SQL designer, when Row Info is enabled, a NOT, OR, and AND are displayed to the left of the grid next to a row and indicates the rules for combining the criteria rows. See Figure 11.1.

To enable row info, right-click on the grid and select Row Info from the grid’s context menu.

Criteria combo box

To specify how selection criteria rows are combined to form more complex selection criteria, select from the drop-down list box above the criteria grid.

- **ALL**. Specifies that all selection criteria in the grid must be true for the combined criteria to be true. With row info enabled an AND will appear next to each additional row after the first row.
- **ANY**. Specifies that at least one of the selection criteria in the grid must be true for the combined criteria to be true. With row info enabled an OR will appear next to each additional row after the first row.
- **NONE**. Specifies that all of the selection criteria in the grid must be false for the combined criteria to be true. With row info enabled a NOT will appear next to the first row and an OR will appear next to each additional row after the first row.
- **NOT ALL**. Specifies that at least one of the selection criteria in the grid must be false for the combined criteria to be true. With row info enabled a NOT will appear next to the first row and an AND will appear next to each additional row after the first row.

Grouping selection criteria in the SQL designer

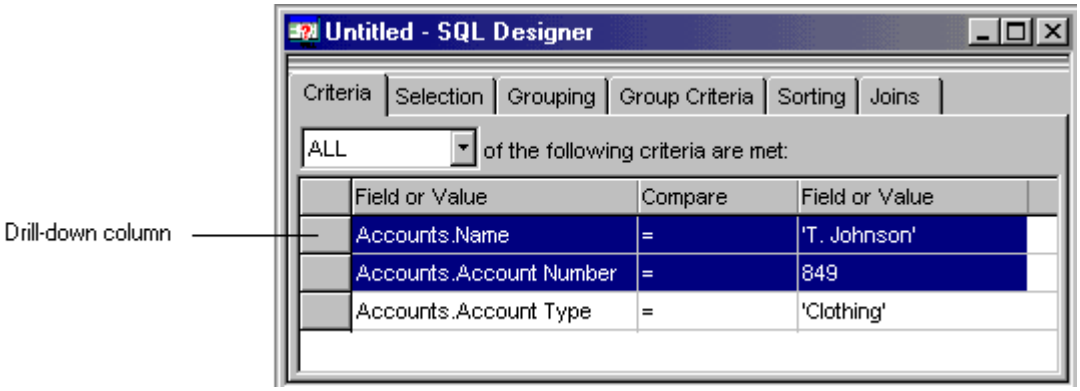
In the SQL designer, individual selection criteria can be grouped together to form nested selection criteria.

To group selection criteria,

1. Select the rows to group by holding down the Ctrl key and clicking each row you want to select in the drill-down column. The drill-down column is located in the leftmost column of the grid and doesn't have a header description.
2. Right-click in the drill-down column and choose Group Rows from the context menu.

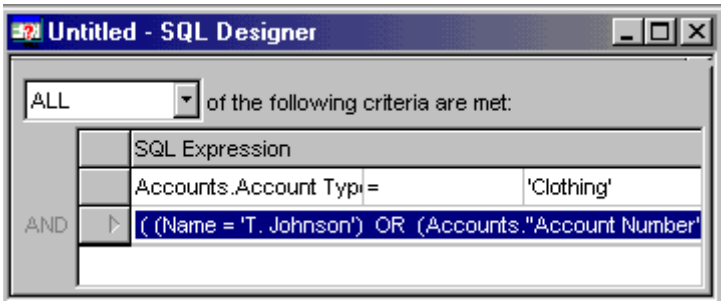
For example, to group the Name and Number criteria into a nested AND statement from the following rows, hold down the Ctrl key, and click the Name and Number rows in the drill-down column. Right-click in the grid, and choose Group Rows from the control menu.

Figure 11.2 SQL Designer: Group selection



After grouping Name and Number

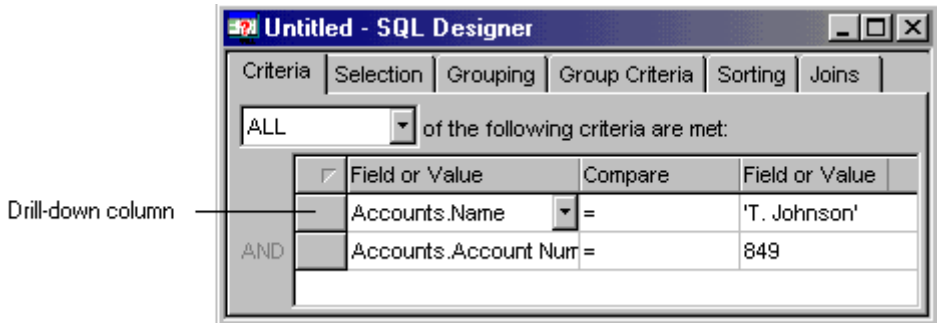
Figure 11.3 SQL Designer: Grouped



Drill-down column

The drill-down column is the leftmost column of the grid and doesn't have a header description. If the query contains a nested selection criteria, a *drill-down* arrow appears next to the nested row. Clicking this arrow "drills down" into the expression. If a grouped expression has already been drilled down, a *drill-out* arrow appears in upper left cell of the grid. Clicking this arrow "drills out" the expression.

Figure 11.4 SQL Designer: Drill-down



Ctrl+U and Ctrl+D also allow you to drill up and drill down respectively.

To ungroup selection criteria,

Change the operator in the grouped expression to its opposite. For example, if you have the previous grouped expression:

(City = 'Freeport') OR (Company = 'Unisco')

Change the OR to an AND:

(City = 'Freeport') AND (Company = 'Unisco')

Query operators

The following are the operators available in the SQL designer's Criteria and Group Criteria pages. The Joins page uses only the standard boolean operators.

Table 11.1 Query operators

Operator	Description
>, <, >=, <=, <>	Standard boolean operators for a comparison test.
LIKE	Adds a LIKE clause to the query. Tests whether the data matches the specified pattern.
NOT LIKE	Adds a NOT LIKE clause to the query. Tests whether the data value does not match the specified pattern.
IN	Adds an IN clause to the query. Tests whether the data matches at least one value in the list of values. To create this list of values, enter fields and/or values separated by commas into the second Field or Value column.
NOT IN	Adds a NOT IN clause to the query. Tests whether the data does not match any value in the list of values. To create this list of values, enter fields and/or values separated by commas into the second Field or Value column.
BETWEEN	Adds a BETWEEN clause to the query. A BETWEEN clause tests whether the field or value falls within a specified range of values. For example, you can use this to return the salespeople whose sales are between $50,000 and $200,000.
NOT BETWEEN	Adds a NOT BETWEEN clause to the query. A NOT BETWEEN clause tests whether the field or value is outside a specified range of values.
IS NULL	Adds an IS NULL clause to the query. Tests whether the field or value contains a NULL value.
IS NOT NULL	Adds an IS NOT NULL clause to the query. Tests whether the field or value does not contain a NULL value.

Selection page of the SQL designer

The Selection page in the SQL designer enables you to create summary data. It also allows you to specify a customized output name for a field or summary data in the query results.

Selecting a field

Select a field from the Field drop-down list box. Fields will be available for each table that appears in the table pane. A Field may also be dragged from a table window in the table pane and dropped onto the Field column.

Specifying an output name

In the Output Name column for a field or summary, you may enter a name you wish to appear as the title for that field or summary data rather than using the default.

Producing summary data

Right-click in the grid and select Summary from the context menu. The grid will have three columns: Output Name, Summary, and Field. Select the appropriate function from the Summary column's drop-down list box. You can also drag a field from a table window in the table pane and drop it onto the Field column.

When you add a summary, the SQL designer automatically groups on all of the non-summary fields to satisfy SQL syntax requirements.

Removing duplicate rows

When the Remove Duplicates box is checked every row in the query results will be unique. Checking this box adds the DISTINCT keyword to the SQL statement.

Deleting a row

After selecting the row to delete, right-click and choose Delete Row from the context menu.

Grouping page of the SQL designer

The SQL designer Grouping page enables you to create a grouped query. A grouped query groups the data from the source tables and produces a single summary row for each row group.

Creating a grouped query

To create a grouped query,

1. Select the field or fields you wish to group by from the Output Fields list box.
2. Click the Add button to move the field to the Grouped On list box. The query will be grouped based on fields that appear in the Grouped On list box.

To have a field appear in the Output Fields list box, select the field in the Table Pane.

To remove a field from the Grouped On list box, select the field and click the Remove button.

Group criteria page of the SQL designer

The SQL designer Group Criteria page enables you to specify group selection criteria used in the HAVING clause that the SQL designer adds to the query. A HAVING clause selects and rejects row groups.

The group criteria can be a simple expression, an SQL expression, or a two summary expression. Right-click in the grid and select the appropriate criteria type from the context menu.

Adding group selection criteria

To change the type of selection criteria, right-click in the grid on the Group Criteria page and select one of the following from the grid's context menu.

SQL Expression

Enter an SQL expression directly. For example,

```
SUM (Qty * Price) > 1000
```

Simple Having Summary Expression

A Simple Having Summary Expression summarizes the comparison of two fields for each row of data.

When defining a Simple Having Summary Expression the grid has four columns: Summary, Field, Operator, and Field.

A Field may be dragged from a table window in the table pane and dropped onto a Field column.

To enter a Simple Having Summary Expression,

1. Select the appropriate summary value from the Summary column.
2. Enter the first field you wish the summary to compare. This can be done by either dragging a field from a table window in the table pane and dropping it onto the Field column or selecting a field from the drop-down list box.
3. Select the appropriate operator from the Compare column drop-down list box.
4. Enter the second field you wish the summary to compare as described in step 2.

Two Summary Expression

A Two Summary Expression selects and rejects row groups based on the result of the comparison of two summaries.

When defining a Two Summary Expression, the grid has five columns: Summary, Field, Operator, Summary, and Field.

A Field may be dragged from a table window in the table pane and dropped onto a Field column.

To enter a Two Summary Expression grouping criteria:

1. Select the appropriate summary value from the Summary column for the first summary to compare.
2. Enter the first field you wish to summarize for the comparison. This can be done by either dragging a field from a table window in the table pane and dropping it onto the Field column or selecting a field from the drop-down list box.
3. Select the appropriate operator from the Operator column drop-down list box. This operator defines the type of comparison between the two summaries.
4. Select the appropriate summary value from the Summary column for the second summary to compare.
5. Enter the second field you wish to summarize for the comparison as described in step 2.

Combining group criteria

To specify how the selection criteria are combined to form more complex selection criteria, select from the drop-down list box above the criteria grid.

- **ALL** Specifies that all selection criteria in the grid must be true for the combined criteria to be true.
- **ANY** Specifies that at least one of the selection criteria in the grid must be true for the combined criteria to be true.
- **NONE** Specifies that all of the selection criteria in the grid must be false for the combined criteria to be true.

- **NOT ALL** Specifies that at least on of the selection criteria in the grid must be false for the combined criteria to be true.

Deleting a row

After selecting the row to delete, right-click and choose Delete Row from the context menu.

Sorting page of the SQL designer

The Sorting page enables you to specify a sort order for the query.

To sort query results

1. Select the field you wish to sort by from the Output Fields list box.
2. Click the Add button to move the field to the Sorted By list box.

Toggling the sort order

Double click on the field in the Sorted by list box. To change sort order, you may also select the field and then to the left of the list box click on the A-Z for ascending order and Z-A for descending order.

Sorting on multiple fields

Simply add more fields to the Sorted By list box. The query will be sorted based on order of the fields that appear in the Sort By list box.

Adding a field to the Output Fields list box

Select the field in the Table Pane.

Removing a field from the Sorted By list box

Select the field and click the Remove button.

Reordering fields in the Sorted By list box

1. Select the field to move in the Sorted By list box.
2. Click on the gray up arrow to left of the list box to move the field up in the sort order. Click on the gray up/down arrow to the left of the list box to move the field down in the sort order.

Joins page of the SQL designer

The SQL designer Joins page lets you create multi-table SQL queries (joins). See “Creating joins in the SQL designer” on page 11-12 for steps on how to create a join.

Including Unmatched Rows

These check boxes allow you to specify full, left, and right outer joins. If only the first check box is checked, a left outer join is added to the query. If only the second check box is checked, a right outer join is added to the query. If both check boxes are checked, a full outer join is added to the query. If neither box is checked (the default), an inner join is added to the query.

Note The SQL1 standard doesn’t include an outer join in its specifications. Some SQL servers have restrictions on outer joins and some don’t allow outer joins. Please see your server documentation for information on its support for outer joins.

Join list box

This box appears above the grid and allows you to specify a particular join. When you select a join in this box, both tables in the join appear. When a join is selected, the grid contains the field information for that particular join.

Joins grid

The Joins grid contains three columns: Field, Operator, and Field.

To have a field appear as a choice in the Fields column drop-down list boxes, select the field in the Table Pane.

The operator column allows you to specify a comparison operator for the join. You may choose from =, <, >, <=, >=, and <>.

Deleting a join

To delete a join, in the table pane, right-click on the join line between the two joined tables, and choose Delete Join from the context menu.

Deleting a row

After selecting the row to delete, right-click and choose Delete Row from the context menu.

Creating joins in the SQL designer

In the SQL designer, fields can be linked by dragging one or more fields from one table to the fields on another table. Graphically, a join is indicated by a single line that connects the two table windows at their table name.

Each linked field pair in the join is added as a separate row to the Joins page grid. When a join line is selected in the table pane, the join list box (above the Join page grid) will contain the two joined tables.

To create a join

1. In the first Field column, select the field you want to match from the first table.
2. In the Op column, select the appropriate type of match. You may choose from =, <, >, <=, >=, and <>.
3. In the second Field column, select the field you want to match from the second table.

Chapter

12

Designing reports

Reports provide non-editable views of data for formatted print or screen output. You can create reports to answer questions that may involve elaborate queries across a range of databases. A report can focus and manipulate data in many useful ways.

Tip In addition to the Designers outlined in this chapter, dQuery/Web offers "No-Click Reports", a quick and easy way to generate reports. For a demonstration, see “No-Click Reports” on page 3-17.

This group of topics shows you how to

- Use the Report wizard to automatically generate reports (using the wizard is the recommended way to begin creating a report)
- Understand the Report designer structure and objects
- Modify a report, changing its appearance and functionality
- Perform aggregate calculations
- Use multiple streamFrames that point to the same or different rowsets
- Create a variety of specialty labels by using the Label wizard

For information on linking a report to tables, see “Linking a form or report to tables” on page 6-5. For information on creating master-detail relationships, see “Creating master-detail relationships (overview)” on page 6-7.

Before you can link a report to a client/server database, the database must be assigned an alias in the BDE. For details on linking *dBASE SE* to a client/server database, see “How to connect to an SQL database server” on page 2-3.

Note After you have created a report with the look and functionality you want, you can save that report as a custom report (.CRP) and use it as a template for subsequent reports. For instructions on how to create a custom report, see “Creating a custom form, report, or data module class” on page 5-11.

Report wizard

You can quickly create useful reports by using the Report wizard. You specify which table or query contains the data you want to display in the report, and the wizard links to it automatically. The rest of the wizard’s options let you do the following:

- Display detail rows or just summary information.
- Specify fields to be included in the report.
- Group the report by specific fields. You can nest subgroups within groups.
- Choose aggregate operations that can be applied both to a group and to the entire report.

- Specify layout style, including a drill-down option, which displays summary information at the top of the page and details farther down.
- Specify a report title.
- Include the date.
- Include page numbers, which causes the report to display one screen full at a time.

The Report wizard does so much that for many reports you won't need to go any further. You can, however, add complex query statements to your reports by writing code or using the SQL designer to generate SQL statements. And you can add advanced reporting capabilities, as needed, in the Report designer.

It's easiest to begin creating a report by using the Report wizard. You can then modify the design in the Report designer. By using code, you can add a great deal of analysis to your reports and provide more sophisticated and useful pictures of the data in one or more tables.

To use the Report wizard

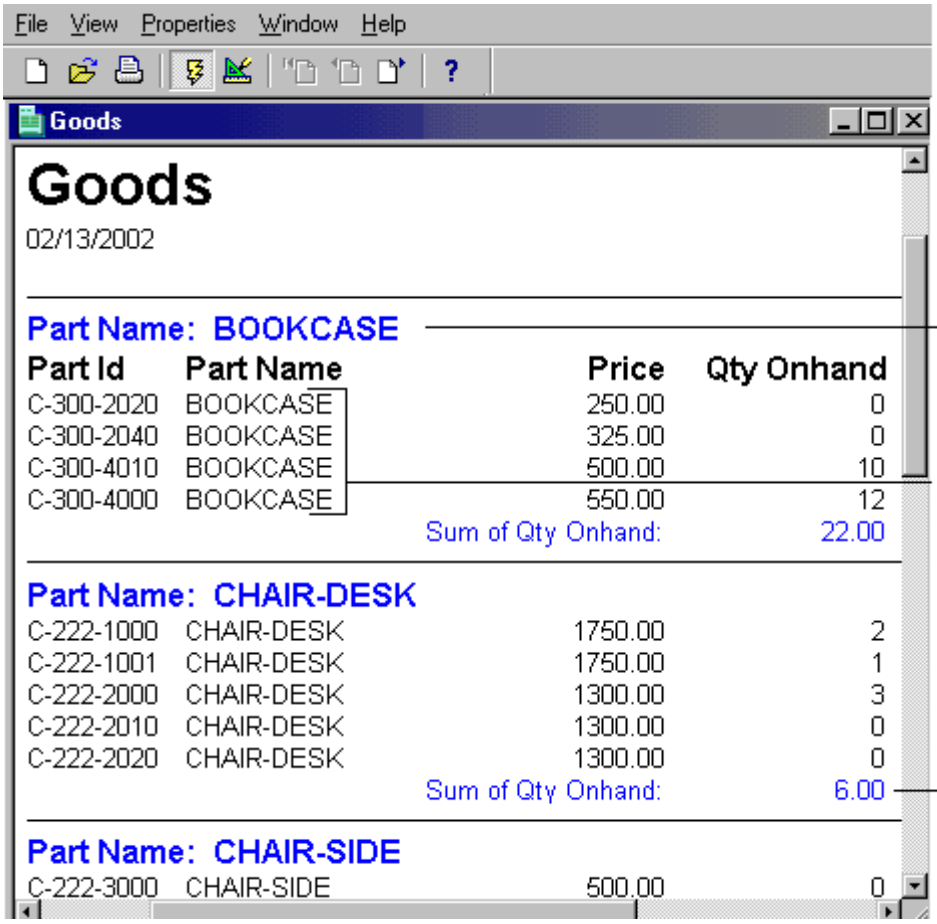
1. Choose File | New | Report. Or, double-click the leftmost Untitled icon on the Reports page of the Navigator. The New Report dialog box appears.
2. Click the Wizard button.

For help on any wizard page, click the Help button on that page.

Example of a report created with the Report wizard

This example uses a GOODS.DBF table that might be used by a Purchasing Department to track current inventory levels. The report answers these questions: What are the total quantities on hand of each furniture type, and what is the cost per unit. Here is the final report:

Figure 12.1 Wizard-generated report on a GOODS table



The report displays a heading for each furniture type, listing the individual styles below each heading. Grouping by furniture type lets you do subtotals, and several other calculations, for each type.

Aggregate operations analyze the values of a selected summary field within a group or over the entire report. You can use any field in a table, even if it is not included in the report. Also, you do not have to specify a field for grouping to use it as a summary field.

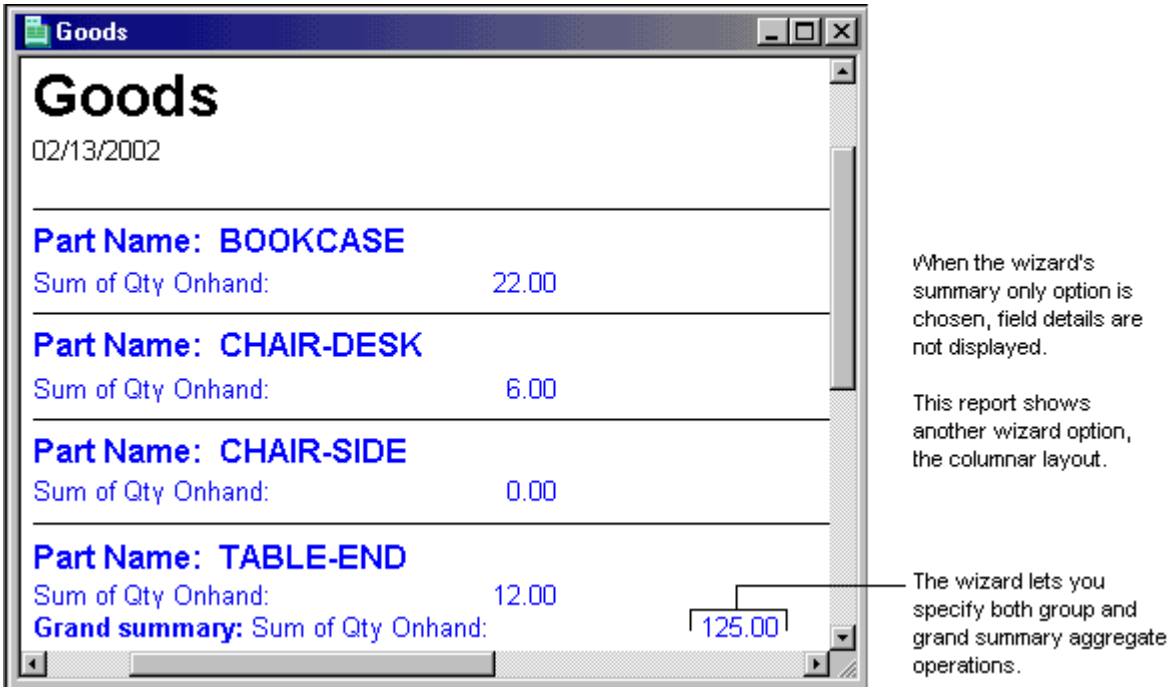
The Qty Onhand field was selected from step 5 of the wizard, and Sum was selected as the Aggregate Operation. This totaled the values in the Qty Onhand field for each grouping of rows, so that a total of the Qty Onhand column appears in each Part Name group.

Because the report is grouped by Part Name, the Part Name column is redundant. To delete a column, see “Deleting columns (fields) from a report” on page 12-6.

Wizard-generated Summary Report

This variation of a wizard-generated report on GOODS.DBF, is the most direct way of answering the question: What is our inventory of each furniture type and what is our total inventory for all units. The finished report looks like

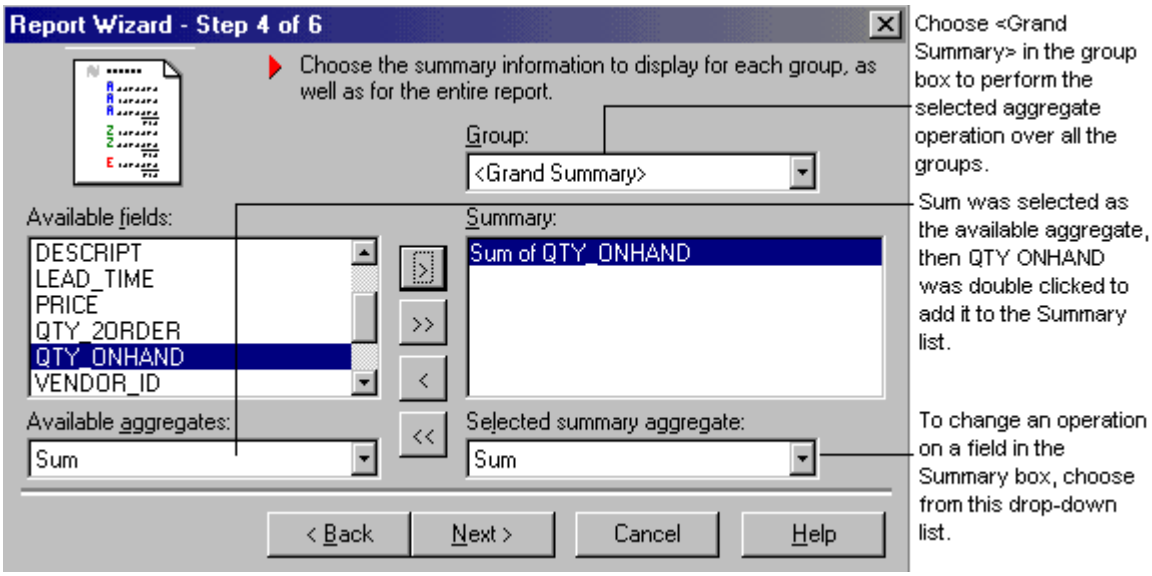
Figure 12.2 Wizard-generated Summary Report



In the wizard, the Qty Onhand field was specified as the summary field for the Part Name group. Sum was chosen as the aggregate operation for this field.

To create the grand total, <Grand Summary> was chosen from the Group drop-down list in step 5. Qty Onhand was selected in the Available Fields box, Sum was chosen from the Available Aggregates list, and the right arrow (>) button was clicked to add the Qty Onhand field to the Summary box, as shown below. The Qty Onhand field's associated aggregate operation then appears in the Selected Summary Aggregate box.

Figure 12.3 Adding grand total in the Report wizard



Report designer elements

The Report designer provides a visual design surface where you can modify reports created with the Report wizard and build new reports from scratch.

The Report and Group panes

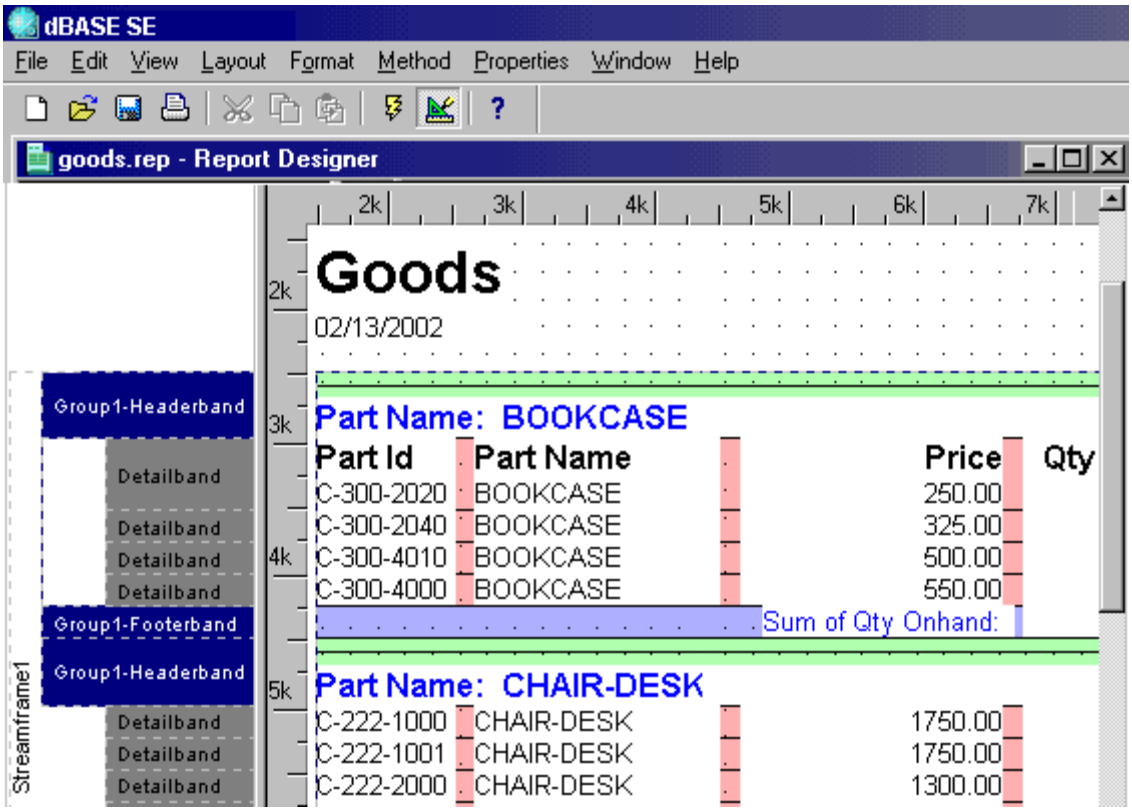
You can view the report in two panes:

- The Report pane, where you visually design the report
- The Group pane, which displays the hierarchy of the groups in the report.

When you first open the Report designer, the split bar between the panes appears at the far left edge of the Report designer window. To open the Group pane, drag the split bar to the right

.

Figure 12.4 Report in Design mode with Group view displayed



The Group pane shows the hierarchy of objects in the report.

- The dotted-line frame labeled Pagetemplate1 is the report object that determines the appearance of the page, such as the background color. Here is where you would place a report title. A report may include more than one pageTemplate object, so that different pages can have different layouts. For example, one pageTemplate can be for a right hand page, and another pageTemplate for a left hand page.

  When creating a report, you place data access components on the pageTemplate object.

- The inner dotted-line frame with the vertical label Streamframe1 is a streamFrame object. This object displays rowset data that is streamed from linked tables (specified in its *streamSource* property). One or more streamFrame objects may be contained within the pageTemplate object. Whatever is placed in the streamFrame area of a report will be displayed when the report is printed.

  If you're using standard user interface components (see "Standard page" on page 7-4), place them on the streamFrame.

- The Group1-Headerband displays the label of the grouped field. Groups are contained in a streamSource object and rendered in a streamFrame object.
- The detailBands are the streamFrame objects that contain the rowsets streamed from the linked table. Each detailBand contains data from an individual row in the table.

The Report pane shows the report appearance with the corresponding structures shown in the Group pane marked.

- The outer dotted area represents the margins of the actual report page (the pageTemplate object).
- The inner dotted line represents the data rows of the report (the streamFrame object).
- The individual fields of the row are columns in the report (the detailBand objects).

Modifying report in the Report designer

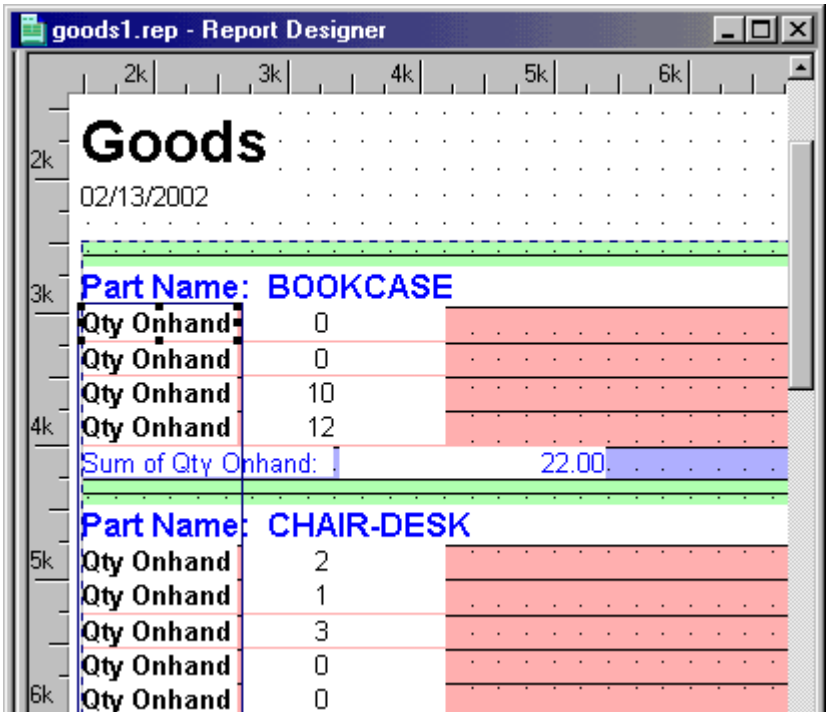
This section illustrates how to use the Report designer to modify the design and adjust the appearance of a report.

Deleting columns (fields) from a report

To delete a column,

1 Click in the column beneath the column heading to select the column, and press Delete.

Figure 12.5 Selecting a column to delete from a report



2 Click the column heading itself, and press Delete.
The remaining columns stay where they are.

If you make a mistake and delete the wrong object, choose Edit | Undo Delete.

Adding columns (fields) to a report

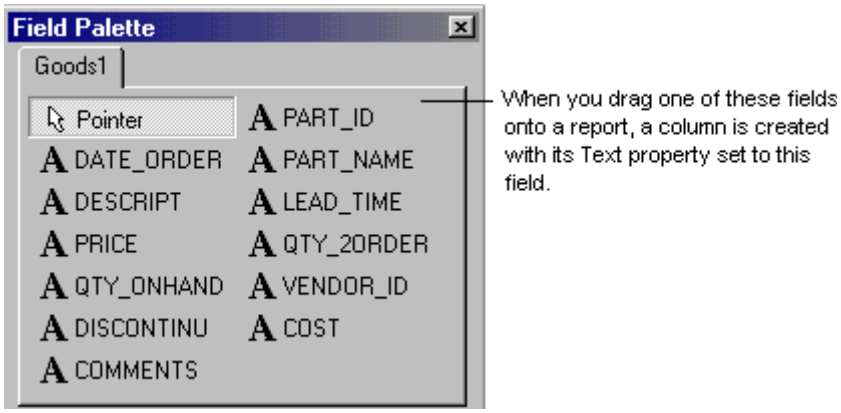
Once you have created a live Query object on a report (usually by dragging the desired table icon to the designer surface), the Field palette is populated with live fields linked to the table data. When you drag a field from the

palette to the design surface, a detailBand is created to display the field as a column in the report. The column is linked to the table's field by a codeblock in the *text* property of the detailBand's Text object.

To add a column,

1. Make sure the report has a Query object whose *active* property is set to *true* and that returns a rowset from the desired table.
2. Display the Field palette (View | Tool Windows, and check Field Palette–Report/Label Designer). The palette shows the active fields of the linked table as Text components.

Figure 12.6 Field Palette containing active fields



3. Drag the desired Field component from the palette to the report.

When you drag a live Field component to a report, the field's name is automatically added as a column heading. To specify placement (or omission) of the automatic column label, choose Tool Windows | Customize Tool Windows and specify your preference on the Field Palettes page.

If you need to move the added column, remember to reposition its heading, as well.

Suppressing duplicate field values

To suppress duplicate field values, set the *suppressIfDuplicate* property of a detailBand's Text object to *true*.

Displaying default values in a blank report field

To display a default value in a report to substitute for blank values in a field,

1. Determine the field in which you want to display the default value, and select that field's Text object in the Inspector (streamSource1.detailBand.<text object>). This example uses the NAME field from the query object, query1.
2. Enter an appropriate codeblock into the *text* property of the Text object (click the tool button in the property box). For example, the following code displays No Value for every blank value in the field:

```
{||this.form.query1.rowset.fields["NAME"].value == "" ? "No Value":
this.form.query1.rowset.fields["NAME"].value}
```

Adding a floating dollar sign to field values in reports

To add a floating dollar sign to the values in a field,

1. Select the field (represented on the report by a Text object).
2. In the Inspector, select the Text object's *picture* property under the Edit category.
3. Click the tool button to display the Template Property Builder.
4. Select the Numeric page.

5 Choose the @$ symbol from the Template Symbols box.

You can also do this by using the Code Block Builder to create the code.

Adding page numbers

To include a page number on each page of a report, drag the PageNumber component (from the Component palette's Custom page) to the report page, positioning it where you want the number to appear. (This is a custom component installed with the *dBASE SE* samples.)

To create a page number from scratch,

1 Place a Text component on the page.

2 Beside the *text* property, select Codeblock from the drop-down list, and then type

```
return this.parent.parent.reportPage
```

(You can do this in the Code Block Builder, available by clicking the wrench tool beside the *text* property.)

Drill-down reports

Drill-down reports display summary information for a report at the beginning of the report. The details of the report appear toward the bottom. Hence the name *drill-down*—users can drill down from the summary at the top to the details at the bottom.

The Report wizard offers you the option to create a drill-down report. You can also create a drill-down report in the Report designer.

Controlling drill-down reports in the Report designer

In the Report designer (or if you print the report), the summary information of the report is in the headerBand and footerBand of the reportGroup class. The details of the report are in the detailBand.

In a non-drill-down report, the bands are rendered in this order (this example groups on STATE):

1 headerBand for report's reportGroup
2 headerBand for state of "CA"
3 detailBand 1 for state of "CA"
4 detailBand 2 for state of "CA"
5 detailBand n for state of "CA"
6 footerBand for state of "CA"
7 headerBand for state of "PA"
8 detailBand 1 for state of "PA"
9 detailBand n for state of "PA"
10 footerBand for state of "PA"
11 footerBand for report's reportGroup

However, in the drill-down report, the bands are rendered in this order:

1 headerBand for report's reportGroup
2 footerBand for report's reportGroup
3 headerBand for state of "CA"
4 footerBand for state of "CA"
5 headerBand for state of "PA"
6 footerBand for state of "PA"
7 detailBand 1 for state of "CA"
8 detailBand 2 for state of "CA"
9 detailBand n for state of "CA"
10 detailBand 1 for state of "PA"
11 detailBand n for state of "PA"

...and so on

The *drillDown* property

You can control the way the drill-down feature works by setting the *drillDown* property on the reportGroup class. This property is an enumerated type, with the following possible values:

Table 12.1 Values for the *drilldown* property

Value	What happened
0 None (not a drill-down report)	
1 Drilldown (standard drill-down report)	All the headers and footers are rendered first, and then the details
2 Drilldown (repeat header)	The same as 1, but the headers are rendered again with the details
3 Drilldown (repeat footer)	The same as 1, but the footers are rendered again with the details
4 Drilldown (repeat header and footer)	The same as 1, but the footers and headers are rendered again with the details

Adding standard components to a report

By adding components from the Report's component palette, you can extend a report's functionality so that it can do some of the same things a form can do.

Important When working with components in Reports, keep these points in mind:

- Report components may be placed only on a pageTemplate, not on a band.
- When you copy and paste a component, its object hierarchy may vary in the following ways:
  - Its parent will be the parent of the currently selected component; or,
  - If the currently selected object can hold a component (a pageTemplate), then that object will be the parent; or,
  - If nothing is selected, then the pageTemplate will be the parent.
- Mouse events of a component whose parent is a band are not available.
- All components available for use on reports have *canRender*( ) and *onRender*( ) events.
- Components on a pageTemplate can be referenced directly as well as through an elements array (as in forms).
- If you do not want a report component to be printed, set the component's *printable* property to *false*.

Changing the report's appearance

You can place data from any source in streamFrames that can be sized and positioned anywhere on the report page. You can create a variety of borders around streamFrames, labels, or fields. You can control all aspects of the appearance of text. And you can specify colors for the entire report's background, for text, and for streamFrames.

Creating report borders

dBASE SE offers a choice of several different styles of borders. You can create borders around all report fields and columns and labels at once or around individual objects to set off things like grand totals. You can also place borders around streamframes (giving groups a boxed appearance).

To set borders around each column in the report, set the report's form.PageTemplate1 *gridLineWidth* property to a positive number. (*gridLineWidth* is in the Inspector's Miscellaneous category.)

To create a border around an individual object, including a streamframe, select the object and set its *borderStyle* property to one of the styles in the drop-down list beside the property. (*borderStyle* is in the Inspector's Visual category.)

For no border, set *borderStyle* to zero.

Setting background color in reports

To set a report's background color,

1. Select the report's PageTemplate object in the Inspector (form.PageTemplate1).
2. Click the wrench tool beside the *colorNormal* property to display the Color Property Builder.
3. Select a color, or create your own, and choose OK.

Setting background image in reports

To set a report's background image,

1. Select the report's PageTemplate object in the Inspector (form.PageTemplate1).
2. Click the wrench tool beside the *background* property to display the Image Property Builder.
3. Click the wrench tool beside the Image box of this dialog box to browse for an image file. As soon as you select it, you can see the background image in the Report designer.

Performing aggregate (summary) calculations

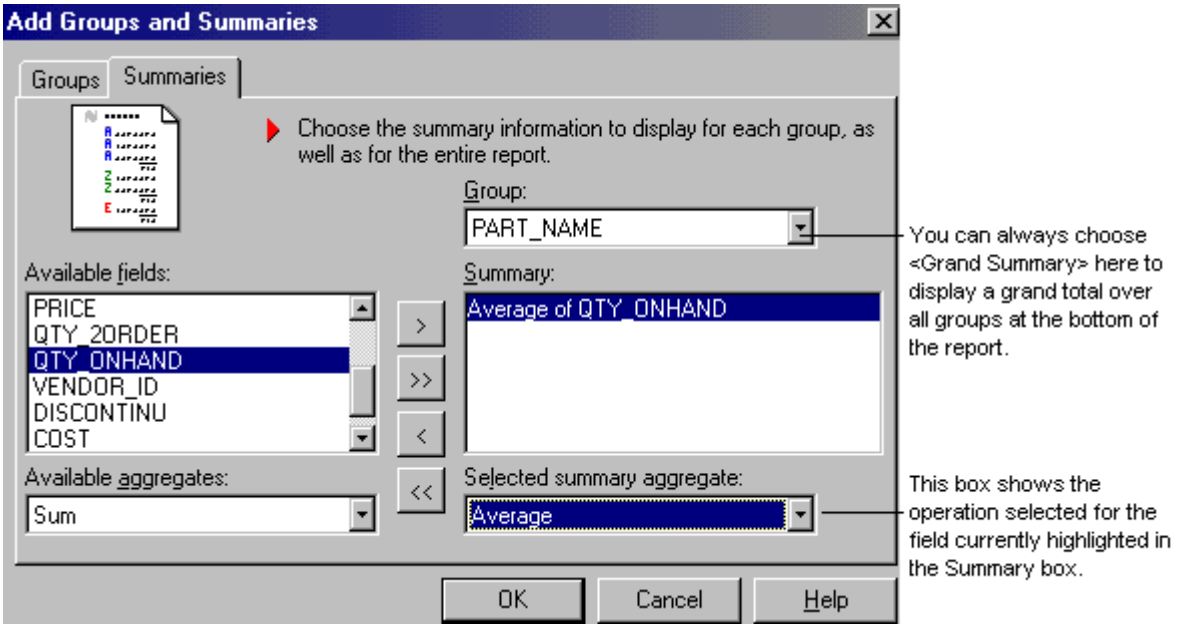
You can perform aggregate calculations (sum, minimum, maximum, count, average, standard deviation, variance) on fields within groups in a report and for the report as a whole.

To do aggregate calculations you must first specify a group of fields on which to perform the summary operation (unless you are summarizing data for the entire report).

To perform an aggregate operation,

1. Choose Layout | Add Groups And Summaries
2. On the Group page of the dialog box, select the field or fields on which you want to group the data. If you are going to do only a grand summary over the entire report (such as a grand total), then skip this step.
3. Click the Summaries tab of the dialog box.

Figure 12.7 Aggregate calculation on a Report



4. From the Group drop-down list, select the group on which you want to perform an aggregate operation. (Grand Summary is always available.)
5. From the Available Fields list, select the field on which you want to perform the operation.

6. From the Available Aggregates drop-down list, select an operation.
7. Click the arrow button to move the field and its operation to the Summary list.
8. Repeat for as many fields as you want.

If you want to change the aggregate summary operation for a field already in the Summary box, select that field in the Summary box and use the drop-down list below it to select a new operation.

If you need very sophisticated calculations beyond the operations offered in the Available Aggregates box, you could also create summary calculations in fields by writing methods for their events. Select the object linked to the field, and type your code into one of its report-specific events, such as

- *onDesignOpen*, activated on opening a report
- *preRender*, activated before a report runs
- *canRender*, activated before a component is rendered, to determine whether the component will be displayed.

Designing a report with multiple streamFrames

The streamFrame object makes it possible to create a rectangular area of any size or position to display data rows from a linked rowset (which is set in the streamFrame's *streamSource's* rowset property).

For example, the Label wizard adds several streamFrames to a report, with their common *streamSource* pointing to the same rowset of customer addresses. By sizing each streamFrame to match sheets of adhesive labels, the wizard can create a report that prints a set of mailing labels.

To add a second streamFrame to an existing report,

1. From the Component palette, Report page, drag the streamFrame component to the design surface of the report.
2. The streamFrame object appears closed, as a box. You can drag this box diagonally to any desired size.

In addition, by using the Source editor, you can create additional streamFrames and set each streamFrame's *streamSource* property to point to different rowsets. For example, you could add two streamFrame objects side-by-side on a report, with one frame displaying sales representatives grouped by city and the adjacent frame displaying prospective customers grouped by city.

Creating printed labels

The easiest way to create mailing labels and many other types of printed labels is by using the Label wizard. Choose File | New | Labels or double-click the Label icon on the Reports page of the Navigator, then choose Wizard. After you select the table or query file you want to use, the wizard does the following:

- Sets up a common address format for you, or you can create your own format
- Sorts the labels on a field you specify
- Sets up the page for the type of label sheets you have (many choices)
- Lets you create a calculated field

Labels are a type of report, given the extension .LAB. Label files you create are listed on the Reports page of the Navigator.

If you choose to modify labels after using the wizard, or if you design them from scratch, you'll be working in the Label designer. The Label designer is similar to the Report designer, except that you have additional choices on the menu.

Chapter

13

Introduction to designing tables in *dBASE SE*

The foundation of any application is the system of tables that store the data. *dBASE SE* gives you powerful tools for creating and managing databases, whether your application needs a simple table or complete access to an enterprise client/server database system.

This section is a conceptual introduction to designing and creating tables:

- Table terms and concepts
- Table design guidelines
- Table structure concepts

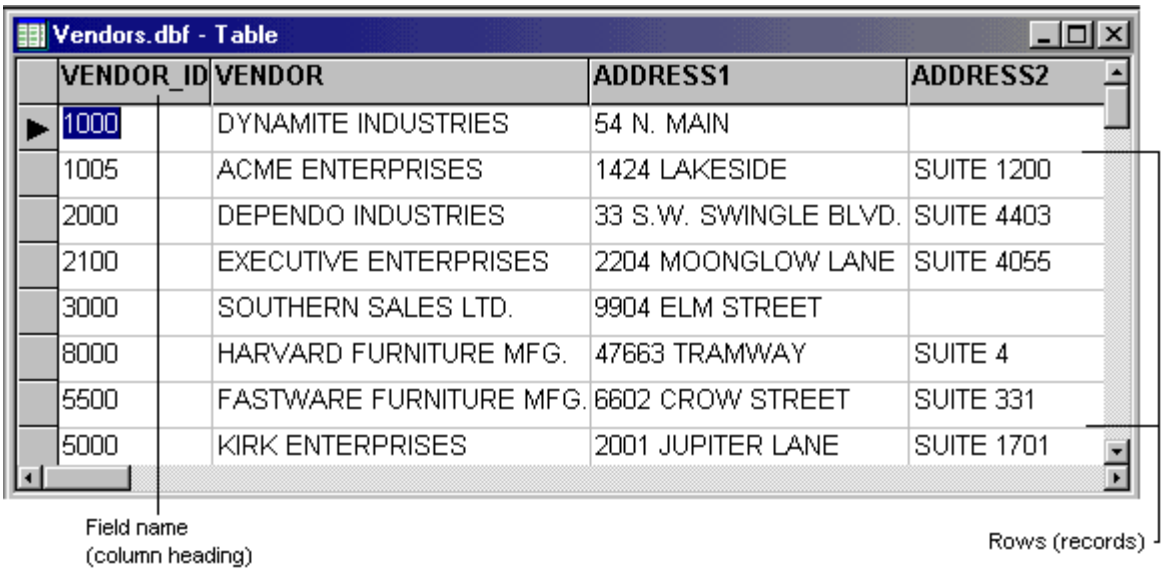
Terms and concepts

You should know these essential terms:

- An ***application*** is a complete system of tables and related forms, queries, reports and other components that handles a data management need.
- A ***database*** is a collection of one or more *tables* that store and classify information, plus related files such as index, graphic, and memo files. Each dBASE table in a database is a distinct file with a .DBF extension.
- A ***table*** consists of one or more horizontal ***rows*** (sometimes known as records) that contain information about a specific person, place, or thing.
- Each row contains one or more ***fields***. A field contains one category of information, such as a person's name, a phone number, or an invoice date.
- A ***field type*** describes the kind of information stored in the field; for example, date, character, logical, numeric.

Each row includes a *row number* field (or RECNO(_) in the dBL language) that identifies that row uniquely in the table.

Figure 13.1 Components of a Table



When you first create the table, you choose a *table type*. Your choices are standard tables (.DBF and .DB) and other supported databases for which you have configured a BDE alias (see, “Supported table types” on page 14-1). Then you define each field’s name, field type, width, and decimal (if a numeric or float field). You can also create an index on the field, which lets you arrange rows in a useful order.

The *dBASE SE* interface adjusts automatically to accommodate the type of table you are working with. For example, if the table you are working with supports data entry constraints, you can specify them in the Inspector while designing a Table. Otherwise, data entry constraints are unavailable in the Table designer.

Table design guidelines

The following sections illustrate typical design issues using a hypothetical small business, a shop that sells diving equipment.

Identifying the information to store

To develop a system of tables for an application, begin by identifying all of the *relevant* information you need to manage—unnecessary data wastes disk space and distracts users from the task at hand.

You might start by looking at the order form you use day to day. Write down all the information you think you need, without attempting to organize it yet, as shown in the following example:

- Products ordered
- Customer name, address, phone number, credit standing
- Order number
- Shipping information, including when it was shipped
- Products purchased
- Purchase date and time
- Salesperson taking the order
- Customer signature
- Special notes about the customer

Next, review this list to see what’s really relevant and what you can do without. For example, the name of the salesperson taking the order might not be important, or you might decide you don’t need to track the exact time of purchase.

Classifying information

After you identify the information you need, review the list and begin to classify the information into distinct groups. Identify the separate entities (such as persons, places, and things) and activities (such as events, transactions, and other occurrences). In general, each table should contain only one kind of entity or activity. The fields in each table identify the attributes of that entity or activity.

Reviewing the list in the previous topic reveals two separate entities (customers and products) and one separate activity (orders). Each of these components has unique attributes. When you reorganize the list according to these categories, you might come up with a new list similar to the following one:

- Customers, including customer name, address, city, state, postal code, phone number, credit standing, signature, notes, and so on
- Orders, including order date, order number, sales date, amount paid, and so on
- Products ordered, including product name, sales price, quantity ordered, and so on

Determining relationships among tables

To better define your tables, you need to determine how they relate to each other.

Single versus multiple tables

Each table should have a specific purpose. It's often better to create several small tables and link them together, rather than try to store everything in one large table. Keeping everything in a single large table usually forces you to store redundant data.

For example, if you stored the complete customer information with each order, you would be entering the customer's name and address with every order. Not only does this procedure invite errors, but it makes it difficult to update information if something as simple as a customer's phone number should change. In addition, redundant data wastes disk space.

Use multiple tables to minimize the amount of data that appears more than once. By storing the name and address information once in a Customer table, you have only one location to update if that information changes. When a customer places an order, the order information goes in a separate table that can be linked to the name and address table.

In our sample case, three distinct tables have emerged to contain data: one for customers, another for orders, and one for the items ordered. We can call the tables Customer, Orders, and Lineitem.

One-to-one and one-to-many relationships

With multiple tables in an application, it's important to understand the relationships among entities and activities. In each relationship, is there a one-to-one correspondence, or does an entry in one correspond to many entries in another? Or is there no direct relationship?

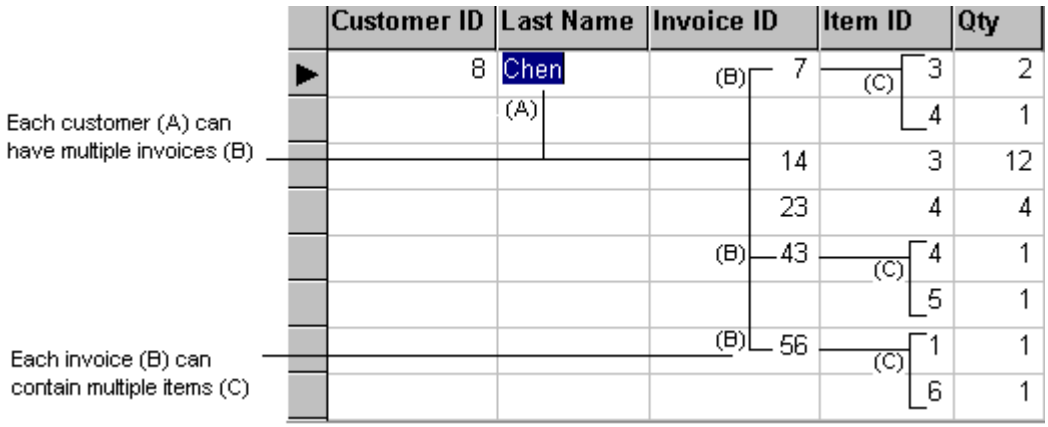
For example, a customer can place several orders, and an order can contain one or more items. These are *one-to-many* relationships. Each order is associated with one customer, a *one-to-one* relationship.

For example, a query might relate three tables:

- From the Customer table comes the customer name, NAME
- From the Orders table comes the order number ORDER_NO
- From the Lineitem table comes the stock number STOCK_NO and selling price, SELL_PRICE.

The query results show each customer name followed by many orders, and under the orders, a list of the items and prices in the order (Figure 13.2).

Figure 13.2 One-to-many relationships



Parent and child tables

When you relate two tables in a query, form, report, or data module, you establish one as the parent and the other as the child table. As you select a row from the parent table, you see the corresponding child row or rows.

Linking tables in a parent-child relationship lets you easily find rows in the child table. For example, you can set up the Customer table as the parent, and Orders as the child. Then, when you move to a new row in the Customer table, the row pointer in the Orders table moves to the orders for that customer automatically. Similarly, the Orders table becomes the parent to the Lineitem table, so that selecting an order also selects the items in the order.

The parent and child tables are linked on a common field, called the linking field. In our example, a query links the Customer and Orders tables on the CUSTOMER_N (customer number) field. In *dBASE SE*, the linking field in the child table must have an index.

Minimizing redundancy

Using multiple tables reduces redundant information. In addition, don't store information that you can easily calculate, unless the calculation requires excessive processing effort or you need an audit trail. For example, if you were creating tables for our hypothetical dive shop, you might want to store the total invoice amount, but not the total sales tax, which *dBASE SE* can easily calculate.

In general, indexed fields that are used for linking should be the only fields that contain redundant information in related tables. Identical data in index fields is necessary for linking tables. In this example, it is the way to identify the same customer in both tables.

Choosing index fields

Indexes make it easier and faster to process information in a table. With multiple tables, indexes are also necessary to link related tables together. Most tables should have at least one index, to organize rows and link to related tables, but too many indexes can slow performance.

To identify which fields to index, ask the following questions:

- What will users know when they search for information? For example, in the dive shop tables, users might want to search for a customer name, customer number, order number, or order date. Consider indexing on these fields.
- What are the common threads that tie the information together? For example, a customer number could be a common field between the Orders table and the Customer table, and the order number could be a common field between the Orders table and the Lineitem table.

For more about indexes, see "Creating and maintaining indexes" on page 14-8.

Defining individual fields

For each field, you define its name, type, size, decimals (if a numeric or float field), and index (optional). The specifics of field types are discussed in the following section. Here are some overall guidelines:

- Use one piece of information per field. For example, put city, state, zip code, and country data into separate fields, because you might want to process the information in each field separately. However, do not split certain information, such as street number and street name, unless you need to process rows by street name or street number separately.
- Keep field sizes to a minimum, without being excessively restrictive, to conserve disk space. If you intend to total a numeric field, you must define a field large enough to hold the total, not just individual values.
- For indexed fields, use abbreviated codes instead of long character fields wherever possible. For example, instead of duplicating the entire customer name in every order, use a short customer code to simplify data entry, indexing, and linking. This results in more efficient indexes and makes it easier to update information.
- Define fields in a logical order in the table. The order you define is the default way in which users will see the table. In general, put indexed fields toward the beginning of the table, and put similar information together in a sensible sequence.
- Use descriptive, unique field names. Be consistent when naming fields that contain similar data. Standardize field names shared across tables if possible (this is not permitted with some SQL databases).

Table structure concepts

This section provides a general overview of basic table structure, with specific reference to the dBASE 7 table type.

Table names

See your database software documentation to determine valid file names for its tables. For example, an Access table has no extension requirement because it is stored within an Access database with an .MDB extension. On the other hand, .DB is the required extension for Paradox tables and .DBF for dBASE tables.

The table name should indicate its purpose and be easy to remember. For example, if a table contains employee information, you might call it EMPLOYEE.DBF.

Table types

The *table type* determines the file format of a table. See “Supported table types” on page 14-1.

The table type you define depends on the way you plan to use the table. If you expect to use the table only with *dBASE SE* applications, the dBASE Level 7 format is recommended for its flexibility and rich feature set. If the table is to be shared with other applications, consider the most useful format for all applications involved.

The dBASE Level 7 format offers all the features of the previous dBASE file formats, including expression indexes and extensive table-, row-, and field-level security.

The *dBASE SE* interface adjusts automatically to accommodate the type of table you are using. For example, if the table with which you are working supports it, you can specify data-entry constraints in the Inspector while working in the Table designer. Otherwise, data-entry constraints are unavailable in the Table designer.

Field types

Each field has a defined *field type*, which determines the kind of information it can store. For example, a character field accepts all printable characters including spaces. You can define up to 1,024 fields in a table.

A dBASE (.DBF) table can contain the following field types.

Table 13.1 dBASE field types for level 7 tables

Field type	Default size	Maximum size	Index allowed?	Allowable values
Character	10 characters	254 characters	Yes	All keyboard characters
Numeric	10 digits, 0 decimal	20 digits	Yes	Positive or negative numbers
Float	10 digits, 0 decimal	20 digits	Yes	Positive or negative numbers. Identical to Numeric; maintained for compatibility.
Long	4 bytes	N/A	Yes	Signed 32 bit integer, range approximately +/-2 billion. Optimized for speed.
Double	8 bytes	N/A	Yes	Positive or negative number. Optimized for speed.
AutoIncrement	4 bytes	N/A	Yes	Contains long integer values in a read-only (non-editable) field, beginning with the number 1 and automatically incrementing up to approximately 2 billion. Deleting a row does not change the field values of other rows. Be aware that adding an autoincrement field will pack the table.
Date	8 bytes	N/A	Yes	Any date from AD 1 to AD 9999
TimeStamp	8 bytes	N/A	Yes	Date/Time stamp, including the Date format plus hours, minutes, and seconds, such as HH:MM:SS
Logical	1 byte	N/A	No	True (T, t), false (F, f), yes (Y, y), and no (N, n)
Memo	10 bytes	N/A	No	Usually just text, but all keyboard characters; can contain binary data (but using binary field is preferred)
Binary	10 bytes	N/A	No	Binary files (sound and image data, for example)
OLE	10 bytes	N/A	No	OLE objects from other Windows applications

The field type determines what you can do with the information in the field. For example, you can perform mathematical calculations on values in a numeric field, but not on values in a logical field.

The field type also determines how the data appears in the field. For example, a date field, by default, displays dates in the MM/DD/YY format (such as 02/14/01). The display of field data is also affected by the settings of the Windows control panel and the settings defined by using the BDE Administrator.

Other table types, such as SQL tables, may have different field types. Refer to your server documentation for specific details.

Chapter

14

Creating tables

This chapter describes the *dBASE SE* Table wizard, designer, and other tools for designing table structures. Here you will find procedures for creating structures, indexes, and performing other database design tasks. It assumes you are familiar with the basics of table design presented in Chapter 13, “Introduction to designing tables in dBASE SE”, and covers the following topics:

- Supported table types
- Using the Table wizard
- Using the Table designer
- User-interface elements in the Table designer
- Restructuring tables (overview)
- Creating custom field attributes
- Specifying data constraints
- Creating and maintaining indexes
- Referential integrity

Supported table types

dBASE SE provides a Table wizard and Table designer to quickly create tables in any supported table format. Although a particular database application may provide the fullest support for its native format, you can conveniently lay out the basic structure of its tables in the *dBASE SE* Table designer and view any table in Run mode.

- All table access is handled through the Borland Database Engine (BDE), which includes drivers to support the following table, and database formats.
- BDE-standard (no other software or BDE alias required):
  - dBASE
  - Paradox
- Other desktop database formats:
  - FoxPro 2.5
  - Microsoft Access 95/97

  The software application must be installed and running, with aliases assigned in the BDE Administrator. Alternatively, BDE’s ODBC socket supports any ODBC database. For example, if Microsoft Access is not installed, you can connect to an Access database via ODBC. For details, see BDE Help (BDEADMIN.HLP).

- SQL enterprise client/server database formats:
  - Oracle
  - Sybase
  - Informix
  - Microsoft SQL Server
  - IBM DB/2
  - InterBase

  The database server system must be installed and running, with aliases assigned in the BDE Administrator.

A BDE alias is a short name used as a shortcut to a client/server database or to a directory containing database files. BDE aliases are required for Access, Foxpro, and all SQL client/server systems. You may also use BDE aliases for dBASE and Paradox tables for convenience or application portability, although it is not required. See “Configure the Borland Database Engine (BDE)” on page 2-4 for information on setting a BDE alias.

To create SQL tables, you must first be able to access your SQL database. See “How to connect to an SQL database server” on page 2-3 for instructions.

Although you can create tables in any supported format, this section shows how to use the Table wizard and designer to quickly create tables in the dBASE Level 7 table format, which is the most feature-rich and convenient. Some of the dialog boxes and capabilities might not apply to a particular database you are using. Please see the documentation for your database for guidance on implementing tables in its native format.

Note The terms "database" and "table" are often confused. A database consists of a set of files, including indexes, memo, and graphics files, and one or more tables that may be related by key fields. A table consists of an ordered set of rows (records), each row containing a set of defined data fields. The larger, client/server database management systems are considered more database-oriented. The smaller "desktop" database applications are sometimes said to be table-oriented, although when related tables and index files are stored in a directory, that directory may be considered a database.

Using the Table wizard

To use the Table Wizard to create a new table,

1. If you intend to create a table type other than dBASE or Paradox, click the Tables tab in the Navigator, and select an alias from the Look In drop-down list. Your new table will be of that alias’s database type.
2. Choose File | New | Table (or double-click the Untitled icon on the Tables page of the Navigator). The New Table dialog box appears.
3. In the New Table dialog box, choose Wizard.
4. In step 1 of the wizard, select the fields you want from the available tables.
5. Click Next and select the table type.
6. Choose Run Table to save the table
7. Choose Design Table to continue the design process
8. In step 2, select a table format type. If you have selected a database alias in the Navigator as described in step 1 of this procedure, that alias appears as the default table type and cannot be modified. To change it, you must exit the wizard and choose a new BDE alias from the Look In box on the Tables page of the Navigator.

Click the Help button on any step of the wizard for details about the options available with that step.

Using the Table designer

This section outlines a short procedure for creating tables in the Table designer. For a demonstration on designing a table using dQuery/Web, see “Creating Query objects” on page 3-7

To create a new table by using the Table designer,

1. Choose File | New | Table.
2. In the New Table dialog box, choose Design.

3. The Table designer appears showing a default template for the first field.
4. Set the table type in the Inspector. You can select a BDE-standard table type or the table types of those databases for which you have created BDE aliases.
5. In the Table designer, type a name (no spaces for dBASE files) in the Name field. (You have to name a field before you can specify any of its other attributes.)
6. Specify values for the remaining attributes (Type, Width, Decimal, Index) by typing what you want, or by selecting a value from the drop-down list, or by clicking the spinbox arrows. You can tab through these to select the default values.

To create additional fields, press Return when you have finished specifying a field. Or press the Down arrow key. Or right-click and choose Add Field from the context menu.

You can generate new fields in rapid succession by naming each, then pressing the Down arrow key. After naming all the fields in your table design, you can go back and set or reset the attributes for each field.

Warning! Later on in your work, do not use functions such as CHR(_), LTRIM(_), RTRIM(_), TRIM(_), or IIF(_) that vary the field width in the key expression.

Table designer tips

- To add, insert, or delete fields, right-click in the Table designer window to display a context menu, and choose the appropriate command. See “Adding and inserting fields” on page
- To reorder the sequence of fields, place the insertion point in the field number box—it becomes a hand—and move the field to the desired position in the list.
- For information on .DBF field types, see Table 13.1, on page 13-6.
- For more information on elements of the Table designer, see the next section.

User-interface elements in the Table designer

This section provides a detailed description of the user interface elements and common tasks of the Table designer.

To open the Table designer to modify an existing table, right-click the table in the Tables tab of the Navigator, and choose Design Table from the context menu, or select the table and click the Design button on the toolbar.

- The Table designer lists the fields defined in the table, along with the attributes for each field.
- **Field** contains a number that identifies the field in the table. Field numbers are consecutive, automatic, and read-only. They determine the default order in which fields appear in the Table window.
- **Name** is the name of the field (up to 31 characters for *dBASE SE*). You can enter letters, numbers, and underscores, but no other characters. The first character must be a letter. Paradox and most SQL tables allow spaces; dBASE tables do not.

Note Do not use reserved words for the field name (ex. DATE).

- **Type** is the field type. Select the type you want from the list. The type you select determines what kind of data the field will contain. It also determines whether you can set the width, decimals, and index options for this field.
- **Width** is the field size. In the case of dBASE tables you can change field size for character, numeric, and float fields only (all others have fixed width). Never use functions that vary field widths.
- **Decimal** is the number of digits allowed to the right of the decimal point (for float and numeric fields only). In the case of dBASE tables, float and numeric fields, by default, have no decimals selected. You can set decimals to a maximum of 2 less than the width value you define. The total width must be 20 characters or less. This includes decimal settings, the decimal point, and an optional minus sign.
- **Index** determines whether to index rows using the values in this field (you can set an index on character, date, float, and numeric fields in dBASE tables). Select Ascend to index this field in ascending order (for character fields, this is ASCII order, or the order determined by your language driver). Selecting Descend indexes this

field in descending order, and None (the default) omits this field from indexing (or removes an existing index associated with this field).

If you select Ascend or Descend for a dBASE table, the Table designer creates an index for the field in the multiple index file (.MDX) associated with the table.

To set a primary key on a dBASE 7 or Paradox table, choose Structure | Define Primary Key. Some SQL types also support this.

You can also set other field attributes or create custom field attributes by selecting the field and opening the Inspector. See Help.

Resizing columns

You can resize or move columns and move rows in the Table designer.

- To resize a column, point to the column border. When the pointer changes to a double-headed arrow, the column is outlined and you can drag the border until the column is the size you want.
- To move columns, point to the title of the column you want to move. When the pointer changes to a hand, the column is outlined and you can drag it to its new location.
- To set multiuser locks or default table type and other properties, choose Properties | Desktop Properties | Table tab.

Note If you want to see rows that have been marked for deletion when the table is in Run mode, the Deleted option must be unchecked in the Desktop Properties dialog box. The rows will appear, and the work deleted will appear, in the status bar for those records marked for deletion. There is not a "delete flag" for each record.

Getting around in the Table designer

In the Table designer, each horizontal row of properties represents one field (or column) in the table you are designing or modifying. To add, change, or delete data, first select the field by clicking with the mouse or by using keyboard shortcuts.

To go to a specific field number,

1. Choose Structure | Go To Field Number or press Ctrl+G.
2. Type the number of the field to go to and click OK.

Adding and inserting fields

You can add a new field to the table by either adding a row at the end of the fields list or by inserting a row anywhere in the list.

To add a new field to the end of the fields list, choose Structure | Add Field (or right-click anywhere in the Table designer and choose Add Field from the context menu).

To insert a new field between other fields, select a field, and choose Structure | Insert Field, or right-click and choose Insert Field from the context menu. The new field’s row of properties appears above the one you selected.

Moving fields

To move a field, changing its order in a table, point to the field number in the leftmost column. When the pointer changes to a hand, drag the row up or down to its new location.

Deleting fields

To delete fields from a table,

1. Click anywhere in the property row of the field you want to delete.

2 Choose Structure | Delete Current Field (or right-click and choose Delete Current Field from the context menu).

The Table designer deletes the field definition. If the table contains rows, the data in this field is deleted as soon as you save the table structure.

Note Shortcut keystrokes can also be used to add (Ctrl+A), insert (Ctrl+N) and delete (Ctrl+U) fields.

Saving the table structure

Save the table design to keep the structure you've created. If you haven't yet saved a new table design, doing so creates the table and any associated files (such as .DBT and .MDX files in the case of dBASE tables).

To save changes to a table design, do one of the following:

- If it's a new table, choose File | Save.
- To save an existing table under a new name, Choose File | Save As.

If you are saving for the first time, or if you choose Save As, the Save Table dialog box appears.

If you are saving an older dBASE .DBF file, it will be saved in the new dBASE Level 7 file format with all the extended capabilities built in.

Note The table might not be usable in previous versions of dBASE.

Type a valid file name. Choose a destination drive, directory, and database, if needed, and then choose OK. *dBASE SE* creates or updates the table and any associated files.

Note You may not use a file-name extension ending in a "T".

Abandoning changes

Abandon changes to a table design if you want to cancel creating a new table or discard the changes you have made to an existing table.

To abandon changes,

1 Choose File | Close to close the Table designer.

2 Choose No when asked to save changes.

Restructuring tables (overview)

It's easy to change the structure of a table, even if the table contains row data.

If the table is empty, you can make any valid changes you want to the table structure except change the table type. If the table contains rows, however, you need to be more careful about the changes you make—and you should make a backup copy of the table before attempting to change its structure.

When you change the structure of a table, the Table designer makes a backup copy of the old table, creates a new table with the revised design, and attempts to copy all the data from the backup table to the new table. However, each time you change the structure of this table, the backup copy that the Table designer created is overwritten. That is why you should make your own backup copy with a unique name or in another directory.

This section assumes you are using BDE-standard table types (dBASE or Paradox). You can also change the structure of the table types of other databases connected via BDE aliases. For information on restructuring these tables, see the documentation of the respective manufacturer.

Important guidelines for restructuring

When you change the structure of a table, the Table designer uses the field name and field position to determine how to transfer information to the new structure.

Warning! If it cannot find a corresponding field in the new table, the Table designer *does not copy the data from the fields in the backup table*; instead, the information is lost when the backup table is deleted.

To prevent losing data that you want to keep, save the table structure frequently as you make changes and confirm that they are completed successfully.

If you change the type of a field, the Table designer does its best to convert data to the new type. Some conversions are relatively straightforward, such as converting date, logical or numeric fields to character. However, radical conversions (such as a memo field to a date field) might produce results you don't want. In addition, the Table designer does not copy data that is invalid in the new field type. For example, attempting to copy the value "123ABC" from a character field to a numeric field fails because letters aren't valid entries in numeric fields.

Modify structure rules

Before allowing changes to the structure of a dBASE table, *dBASE SE* makes a backup of the original table assigning the file a .DBK extension. *dBASE SE* then creates a new table file with the .DBF extension and copies the modified table structure to that file. When you've finished modifying a table structure, *dBASE SE* copies the content of the backup file into the new structure. If data is accidentally truncated or lost, you can recover the original data from the .DBK file. Before modifying the structure of a table, make sure you have sufficient disk space to create the backup file plus any temporary storage required to copy records between the two tables (approximately twice the size of the original table).

If a table contains a memo field, MODIFY STRUCTURE also creates a backup memo file to store the original memo field data. This file has the same name as the table, but is given a .TBK extension.

You shouldn't change a field name and its width or type at the same time. If you do, *dBASE SE* won't be able to append data from the old field, and your new field will be blank. Change the name of a field, save the file, and then use MODIFY STRUCTURE again to change the field width or data type.

Also, don't insert or delete fields from a table and change field names at the same time. If you change field names, MODIFY STRUCTURE appends data from the old file by using the field position in the file. If you insert or delete fields as well as change field names, you change field positions and could lose data. You can, however, change field widths or data types at the same time you insert or delete fields. In those cases, since MODIFY STRUCTURE appends data by field name, the data will be appended correctly.

dBASE SE successfully converts data between a number of field types. If you change field types, however, keep a backup copy of your original file, and check your new files to make sure the data has been converted correctly.

If you convert numeric fields to character fields, *dBASE SE* converts numbers from the numeric fields to right-aligned character strings. If you convert a character field to a numeric field, *dBASE SE* converts numeric characters in each record to digits until it encounters a non-numeric character. If the first character in a character field is a letter, the converted numeric field will contain zero.

You can convert logical fields to character fields, and vice versa. You can also convert character strings that are formatted as a date (for example, mm/dd/yy or mm-dd-yy) to a date field, or convert date fields to character fields. You can't convert logical fields to numeric fields.

In general, *dBASE SE* attempts to make a conversion you request, but the conversion must be a sensible one or data may be lost. Numeric data can easily be handled as characters, but logical data, for example, cannot become numeric. To convert incompatible data types (such as logical to numeric), first add a new field to the file, use REPLACE to convert the data, then delete the old field.

If you modify the field name, length, or type of any fields that have an associated tag in the production (.MDX) file, the tag is rebuilt. If any indexes are open when you modify a table structure, *dBASE SE* automatically closes those indexes when saving the modified table. You should re-index the table after you modify its structure.

In addition to these guidelines, remember that if you delete a field in a table that contains rows, you lose the information in that field permanently. You can recover the information only if you have made a backup of the table.

Changing the structure

To change the structure of a table,

1. Open a table in Design mode.

   If you are working in a shared environment, you see a prompt to open the table exclusively. Choose Open Exclusive to open the Table designer.
2. Make a working copy of the table (choose File | Save As and specify a new name for the table). The working copy now has focus.
3. Change the field definitions you want. You cannot change the table type.
4. When you finish, choose File | Save. In addition to saving your changes, the Table designer also copies associated files (such as .MDX and .DBT files).

Note Open the restructured table in Run mode to verify that your data is in the condition you want. If not, you can revert to your original table if you worked from a copy.

Printing the table structure

To print the table structure for future reference,

1. Open the table in the Table designer.
2. Choose File | Print.
3. Choose the print options you want and click OK.

Table access passwords

In addition to restricting access to networks and servers, you can limit access to sensitive tables by setting passwords directly on those files. The dBASE file format provides extensive table-,row-, and field-level access restrictions. For more information on *dBASE SE* security features, see Chapter 16, “Setting up security”.

Creating custom field attributes

Custom field attributes specify how a field will be displayed in a form or report, irrespective of the form’s default control settings. You can create custom field attributes in the Table designer to control special properties and events on forms and reports. Whenever a field is *dataLink*ed to a control, all custom field attributes are copied to the control.

Custom field attributes are named field properties that contain a string value. These attributes form an active data dictionary that functions at both design and runtime. Attributes are listed under the Inspector’s Custom category.

Properties assigned to fields by creating custom field attributes are not streamed. Therefore, you can change the attribute in the table without having to change your code. If you later change the field’s attributes, the changes are automatically applied to the control *dataLink*ed to the field. No change to a report or form is necessary.

By using custom field attributes, you can cause a table’s field to have its own special font, color, or format that will be reproduced on any form or report whose controls are *datalink*ed to it. For example, you can assign a picture attribute to a PHONE field. When you *dataLink* an entryfield control to the PHONE field, that entryfield control will automatically take on the *picture* property that was assigned as the field’s picture attribute.

To create custom field attributes in the Table designer,

1. Select the field for which you want to create a custom attribute and open the Inspector.
2. Right-click and from the context menu, choose New Custom Field Property...
3. In the dialog box, enter a name for the new field attribute. For example, if your table contains a phone number field whose data you want to appear in forms in a particular phone number format, you would type picture to add the *picture* property (which provides data format templates, not images).
4. Now enter a value for the new field attribute. If you used the *picture* property, you might add a template value for the property, such as 999-999-9999 for a USA phone number template.
5. The new field attribute appears in the Inspector, listed under the Custom category.

To edit or delete custom field attributes,

1. In the Inspector, select the custom field attribute.
2. Right-click to display the context menu.
3. Choose Modify or Delete Custom Field Property from the context menu.

No checks are performed on the attribute name; be sure not to create attributes, such as "Name," that will cause undesired property name conflicts. Attributes with names not used by the component simply become custom properties of the component.

Specifying data-entry constraints

If supported by the database type, you may be able to specify data-entry constraints—rules that govern the values you can enter in a field. If you want to make sure that the values users enter in a field meet certain conditions, specify a data-entry constraint for that field.

You can specify data-entry constraints for each field in the Inspector when you create or modify a table that supports them, such as a dBASE or a Paradox table.

The Inspector displays different data-entry constraints depending on the field type.

Table 14.1 Data-entry constraints

Validity check	Meaning
Required	Every row in the table must have a value in this field.
Minimum	The values entered in this field must be equal to or greater than the minimum you specify here.
Maximum	The values entered in this field must be less than or equal to the maximum you specify here.
Default	The value you specify here is automatically entered in this field. You can replace it with another value.

Creating and maintaining indexes

Rows in dBASE tables can be organized either by indexing or by sorting. Both methods arrange rows in a specific order, but in completely different ways. Relational databases require index files; sorting creates a separate table with a different organization.

This section describes both indexing and sorting in a dBASE table. It covers the following topics:

- Indexing versus sorting
- Simple indexes and complex indexes
- Design concepts and guidelines for indexes
- Adding, modifying, and deleting indexes
- Sorting data to a separate table
- Creating indexes for Paradox tables

Note The material in this section applies to dBASE, Paradox, and SQL indexes. However, specific guidelines and procedures might differ. If you're using SQL tables, see your database documentation.

Indexing versus sorting

Indexing and sorting are two approaches for establishing the order of data in a table. You use them to answer different needs in an application. In general, you index a table to establish a specific order of the rows, to help you locate and process information quickly. Indexing makes applications run more efficiently. Use sorting only when you want to create another table with a different natural order of rows.

Indexing orders rows in a specific sequence, usually in ascending or descending order on one field. Indexing creates a list of rows arranged in a logical order, such as by date or by name, and stores this list in a separate file called an *index file*. A dBASE index (.MDX) file can have up to 47 indexes, but only one controls the order of rows at any time. The index that is controlling the order is the current master index.

Note *dBASE SE* stores indexes in multiple index (.MDX) files, and recognizes older .MDX files. You can design and maintain multiple indexes using the Manage Indexes dialog box.

Sorting creates an entirely separate copy of the current table with the rows in a different order. You're likely to use sorting infrequently, only when you want to create a separate table with a different natural order.

Here is a summary of key differences between indexing and sorting:

- **Creating tables.** Indexing creates an index file that consists of a list of rows in a logical row order, along with their corresponding physical position in the table. Sorting a table creates a separate table and fills it with data from the original table, in sorted order.
- **Arranging rows.** Both indexing and sorting arrange rows in a specified order. However, indexing changes only the logical order and leaves the natural order intact, while sorting changes the natural order of the rows in the new table.
- **Processing operations.** Certain operations are much faster using indexes, such as searching for data, running queries, and so on. Some operations, such as linking tables, require indexes.
- **Using functions.** With indexes, you can order rows using fields and *dBASE SE* methods. With sorting, you can use fields only, in ascending or descending order.
- **Adding rows.** If you add rows to an indexed table, the index is updated automatically so that the rows appear in the correct order. If you add or change rows in an already-sorted table, you might need to sort it again.
- **Mixing field types.** With indexing, you must convert field values to a common field type, for example, converting the sale date to a character type. With sorting, you can order rows on fields with different field types; for example, you can sort on customer number (a character field) and sale date (a date field), without converting them to a common field type.
- **Mixing order.** With indexes, the entire index is either ascending or descending. With sorting, you can mix fields sorted in ascending and descending order.

In general, use indexing to make processing more efficient in data entry forms, queries, and reports. The only significant costs are that index files require extra disk space, and processing time is required for ongoing automatic maintenance.

Sorting or exporting rows

Sorting a table copies its contents to a separate table and arranges rows in the order you specify in the new table.

Tip In general, use sorting only when you want to export data to another application or table type. Sorting is useful whenever you want to create a separate table for reporting or other purposes. Use indexing instead when you want to make data entry, querying, and reporting tasks faster and more efficient.

When you sort, the *source table* is the table containing the rows you want to copy, and the *target table* is the new table (and new table type, if you want) to contain the copied rows. Sorting does not change the data in the source file.

When you sort a table, all fields in the source table appear in the target table. You select the fields on which to sort rows.

dBASE SE sorts data in case-sensitive alphabetic order, using the sort order specified by the language driver in the BDE Administrator. Sorting starts with the first character in the key and proceeds from left to right. Punctuation comes before numbers, numbers before letters, and uppercase letters before lowercase letters.

Note Make sure you have enough available disk space to store the table on the target drive.

To create a sorted table or export table data to another table type,

1. Open the table you want to sort in Run mode.
2. Choose Table | Sort Rows to Table. The Sort Rows dialog box appears.
3. Specify a target table. This is the path name of the new sorted file. Click the Target Tablename tool button to display a Save File dialog box. If you want to export the table data to another table type, choose the new table type in the box at the bottom of the Save File dialog box.
4. Select the field(s) on which to sort rows, and click the > button to move them to the Order By list.

The order in which the selected fields appear in the Order By list determines the order of the sort. The target table contains all fields from the source table.

5 Select each Order By field, then specify the sort order.

6 When you have finished, click OK. *dBASE SE* creates a new table. If the target file exists, *dBASE SE* asks whether to overwrite it. The rows you selected are copied to the target table and sorted as you specified, starting with the first Order By field.

dBASE index concepts

Before you create indexes on dBASE (.DBF) tables, you need to be familiar with a few general concepts.

- **Multiple index (.MDX) files.** When you create an index, it is stored in a file with the file-name extension.MDX. Each index has a name (sometimes called a *tag*) that defines the index uniquely in the .MDX file.

  A table's main .MDX file is called the *production index* file. The production index file opens automatically when you open a table, so its indexes are automatically available—though no index sets the row order until you select it as the master index. As you update rows in a table, the affected indexes in the production index file are also updated. If you use any non-production .MDX files, they must be opened explicitly by entering statements in the Command window.

  The production index file has the same name as the table plus the .MDX extension.

- **Key expressions**. A key expression is a field name, or a combination of field names, functions, or operators, that determines how an index orders rows in a table. It must be a character, numeric, date, or float field, or an expression that evaluates to one of these types. The key expression can be up to 220 characters in length.
- **Primary key**. The dBASE 7 table format supports primary keys, enabling you to create primary distinct indexes. Any field can be a primary key and you need not create the primary key before creating a secondary maintained index.
- **Simple indexes**. A simple index uses a single field name for the key expression.
- **Complex indexes**. A complex (or composite) index uses a combination of one or more fields, or a dBASE expression.
- **Ascending and descending order**. Rows can be ordered in ascending order, lowest to highest (the default), or descending order, highest to lowest. For character fields, the order is ASCII or the order established by the language driver installed by the BDE.

Note Keeping a large number of indexes affects performance, because *dBASE SE* must update each one as the table is revised. If you need to improve performance, consider removing rarely-used indexes from the production index file.

Planning indexes

When you design indexes for a table, consider how you will use and process data. Indexes affect and support features that an application provides: data entry, queries, and reports. Asking the right questions at the beginning can save you redesign efforts later.

- Using indexes in data entry
- Using indexes in queries
- Using indexes in reports
- Using indexes to link multiple tables

Using indexes in data entry

Because indexes affect the order in which rows appear, they let users find and update information quickly. To make data entry more efficient, consider these questions:

- What is the order in which users expect to see the data? For example, they might expect to see a list of companies in alphabetical order, a list of purchase orders by purchase order number, or a list of invoices in chronological order. Indexes should reflect the *expected* order of information in a table. If users expect the same information in different sequences, you can create multiple indexes—one for each sequence. For example, in the Orders table, you might want separate indexes for the order number, order date, and customer number.
- To find rows in a table, what kind of information might users know already? For example, to locate an invoice, users might already have the invoice number, approximate date of the invoice, or the company that submitted the invoice. To speed up the search process, you might want to create indexes for the most common ways a user looks for information.
- What kinds of calculations are users going to perform on data in the table? For example, users might want to calculate the average sale per state or the total sales per month. The word "per" is a clue to an index you might want to create—in the first example, indexing the state field and, in the second example, indexing the sales date field. An index can put similar rows in consecutive order so that users can quickly search for the first row in the series and stop processing after the last row in the series. For example, if users want to calculate the total payments to a vendor, consider creating an index for the vendor number or name.

Using indexes in queries

Indexes can increase the speed at which a query is processed. Indexes are also required for defining links among related tables. To make queries more efficient, consider the following issues:

- What kinds of questions are users going to ask? For example, will they want to know the number of items in stock for a particular product? If so, consider creating an index for the product name or identification number.
- What kind of information might a user know before attempting the query? For example, a user might know the name of the product, its identification number, or its type. Consider creating indexes for commonly known information.
- If the index is solely for occasional or ad hoc queries, consider generating an index at query time instead of maintaining an index separately on an ongoing basis. When the query is finished, you can delete the index to recover disk space.

Using indexes in reports

Indexes affect the order in which rows appear in a report. In addition, they can trigger subtotals and totals in a report (when key values change). To make reports easy to design, consider the following issues:

- What is the order in which users expect to see information in the report? For example, do users want to see a chronological list of invoices billed? An index can ensure that rows appear in the expected order.
- What kinds of calculations will the report make? For example, a report might show the total number of sales by salesperson, or the average sale by customer. The word "by" is a clue to an index you might want to create—in the first example, indexing on the salesperson field and, in the second example, indexing on the customer number. Using an index makes it easier to calculate running totals. If a report includes subtotals within totals, consider using a complex index.
- If the index is solely for occasional or ad hoc reports, consider generating an index at report time instead of maintaining an index on an ongoing basis. When the report is finished, you can delete the index to recover disk space.

Using indexes to link multiple tables

Indexes are required for linking related tables together in a multiple-table query. To link tables, consider the following issues:

- What are the relationships among the tables—one-to-one, one-to-many, many-to-many? For example, an Orders table and a LineItem table are in a one-to-many relationship. The Orders table is the parent table and the LineItem table is the child table.
- With related tables, which fields are common among them? To link tables together, you must have an index for the child table on a field that also appears in the parent table. For example, the Orders table and LineItem table both have an ORDER_NO field, and the LineItem table has an index on this field.

- Can you use codes instead of long character fields? For example, to link orders in the Orders table to customers in the Customer table, the application uses the customer number, a short character field that uniquely identifies each customer.

Creating a simple index

A simple index consists of a single field.

The key of a simple index is just the name of a field. For example, in the Customer table, if you index on the CUSTOMER_N field, the key is the field name, CUSTOMER_N.

You can create a simple index using either the Table designer or the Manage Indexes dialog box, as shown in the next two sections.

Using the Table designer to create a simple index

To create a simple index in the Table designer, choose an index order for the field you want to use—ascending or descending.

Using the Manage Indexes dialog box to create a simple index

To open the Manage Indexes dialog box, in Table design mode, choose Structure | Manage Indexes. The Manage Indexes dialog box appears.

To create a simple index,

1. Choose New. The Define Index dialog box appears.
2. Choose fields from the Available Fields list and add them to the Fields Of Index Key list at the right.
3. Choose Ascending or Descending order.
4. Choose Specify From Field List for a simple index.
5. Enter a name for the new index.

You can use letters, numbers, and underscores, but the first character must be a letter. The name you use must be unique within the index file. For a simple index, use the field name.

Check your vendor documentation for other limitations.

By default, *dBASE SE* indexes rows in ascending order. The exact sort order depends on the driver specified in BDE.

When you choose OK in the Manage Indexes dialog box, *dBASE SE* builds any indexes you created or changed and removes any indexes you deleted.

Note You might have to wait while the indexes are created, particularly if the table has many rows or if key expressions are long and complex.

Selecting an index for a rowset

Depending on the table type, a rowset may be displayed in a form in different default orders. When you first open a dBASE table, it appears in natural order. When you first open a Paradox table, the natural order is the primary key order.

For dBASE tables, the production .MDX file opens automatically with the table, but the indexes it contains are not in effect until you select one.

To order rows that appear in a form in a specific way, select the index you want:

1. Open the form in the Form designer.
2. Select the active Query object.
3. In the Inspector, select the *rowset* property.
4. Click the *rowset* property tool button. The Inspector displays the rowset object’s properties.

5 Set the *indexName* property to one of the available indexes.

Index tasks

In addition to creating and selecting indexes, there are several other index maintenance tasks.

Modifying indexes

You can modify an existing index to make it more useful or efficient. For example, if you create a simple index for a dBASE table in the Table designer, you might want to make it a complex index by adding fields or expressions. Or, you might learn after using the index for a while that a different key is more suitable.

To modify an index,

1 Open the table in the Table designer.

2 Choose Structure | Manage Indexes. The Manage Indexes dialog box appears.

3 Select the index you want to modify, and click the Modify button. The Define Index dialog box appears.

4 Make your changes, then choose OK.

Deleting indexes

You can delete an index you no longer need to save space and improve performance. Deleting an index does not delete any rows in the table—it deletes only the separate index that arranges rows in a particular order.

To delete a simple index, open the table in the Table designer, and choose None as the index type for the field.

To delete any other index,

1 Open the table in the Table designer.

2 Choose Structure | Manage Indexes. The Manage Indexes dialog box appears.

3 Select the index you want to delete, and click the Delete button.

4 Choose OK.

5 Save the table.

The index you deleted is removed from the production index file. If you delete the only index in the file, the .MDX file is deleted as well.

Indexing on a subset of rows for dBASE tables

In most cases, indexes include all rows in a table. For special circumstances, however, an index might contain only some of the rows in a table. Indexing on a subset of rows can make it easier to process information in that table. For example, you might want to work with budget information that applies to your sales department only. In this case, you could create an index that includes only those rows whose DEPT_ID is SALES.

To create an index that includes only the rows you want, first determine which rows you want to include, then state this in the form of a valid dBASE expression. For example, if you want to create an index of customers in your South sales region only, you could use a For *condition* expression such as SALES_REG = "SOUTH" to create the index. Thereafter, when you use this index, you see and process customers from the South region only.

Hiding duplicate values

Indexes can contain multiple rows with the same value in an indexed field. For example, a Lineitem table can contain multiple entries with the same ORDER_NO or STOCK_NO.

In certain cases, however, you might want to have a unique index, which finds only the first occurrence of a value in the indexed field and ignores subsequent rows with the same value. This kind of index is useful when subsequent rows repeat information in the first row.

For example, in a Lineitem table, if all products with the same STOCK_NO were sold at the same price, you could use a unique index to hide duplicate index values, so that only the first row with the price would appear.

If you check Include Unique Key Values Only in the Define Index dialog box, only the first row with a duplicate value in the indexed field is included in the index. Subsequent rows with duplicate values in that field are excluded.

Note In dBASE and Paradox indexes, if there is a primary or distinct index, rows may not have duplicate values in the indexed field. Duplicate values cause an error when trying to save. In SQL indexes, uniqueness is required if the index is defined as a unique index.

Creating complex indexes for dBASE tables

Complex indexes on dBASE tables use a combination of one or more field names, plus valid dBASE expressions. Use a complex index when no single field uniquely identifies each row, or when you need the flexibility of an expression to define the index condition.

Indexes on .DB tables also can use multiple fields; such indexes are called *composite indexes*. However, unlike complex indexes in dBASE tables, you cannot use functions or operators in the .DB index expression.

Rules for dBASE complex indexes

For complex dBASE indexes, the complexity of the index expression varies according to the way the index is used. The following rules apply when defining complex indexes:

- An index value can be up to 100 characters long. The text of the key expression can be up to 220 characters long.
- The complex index must be a valid dBASE expression. Note that a single field name is a valid expression.
- The expression must evaluate to a character, date, numeric, or float value.
- It usually, but not always, contains at least one field name.
- For multiple character fields, *concatenate*, or combine, fields using the plus sign (+), as shown in the following examples:

  ```
  LAST_NAME + FIRST_NAME + M_INITIAL
  CUSTOMER + ORDER_NO
  ```

- You can concatenate fields of different data types by converting them to a single type. In the following example, the key expression concatenates the CUSTOMER_N field, which is a character field, and ORDER_DATE, which is a date field. The DTOS( ) function converts the date value to a character string in the format YYYYMMDD. This order—year first, then month, then day—ensures accurate indexing.

  ```
  CUSTOMER_N + DTOS(ORDER_DATE)
  ```

- For converting number fields, use the STR( ) function. Include the width and number of decimal places of the numeric field(s), to ensure accuracy of the index. For example, suppose you are creating an index that includes a character field LNAME, and a numeric field called AMOUNT that is 10 places wide with 2 decimal places. Use the following syntax:

  ```
  LNAME+STR(AMOUNT,10,2)
  ```

Creating the dBASE complex index

To create a complex index for a dBASE table,

1. With the table open in the Table designer, choose Structure | Manage Indexes. The Manage Indexes dialog box appears.
2. Choose New in the Manage Indexes dialog box. The Define Index dialog box appears.
3. Select the combination of fields on which you want to index from the Available Fields list and move them to the Fields Of Index Key list. Or type a key expression, such as STATE+CITY, to create a complex index on the STATE and CITY fields. The key expression can use multiple field names, functions, and operators.
4. Click OK to exit the dialog box and save the index.

Key expressions

The following table shows several examples of key expressions and the fields used.

Table 14.2 Sample dBASE key expressions

Key expression	Fields used	Notes
CUSTOMER_N	CUSTOMER_N	
CUSTOMER_N + ORDER_NO	CUSTOMER_N, ORDER_NO	
CUSTOMER_N + DTOS(SALE_DATE)	CUSTOMER_N, SALE_DATE	DTOS converts date field to character for indexing.
UPPER(LAST_NAME)+UPPER(FIRST_NAME)	LAST_NAME, FIRST_NAME	UPPER changes character field to all caps.

The first example uses a single field as the key expression. Complex indexes, on the other hand, can use a combination of one or more fields, plus functions and operators.

- CUSTOMER_N + ORDER_NO is a complex key expression using multiple fields and the concatenation operator (+).
- CUSTOMER_N + DTOS(SALE_DATE) is a complex key expression consisting of multiple field names and a function.
- UPPER(LAST_NAME)+UPPER(FIRST_NAME) converts characters to all caps before concatenating them. The UPPER function prevents sorting problems when capitalized entries are mixed in with lowercase ones.

Primary and secondary indexes

dBASE SE lets you create primary and secondary indexes for any table type that supports them.

- A primary index is the main index in a table. For DBF tables, only level 7 tables support primary indexes; any expression index may be created as the primary index. For all other table types, the primary index consists of one or more consecutive fields, starting with the first field in the table.
- A secondary index is supplemental to the primary index in a table.

Some table types let you specify whether or not a secondary index is case-sensitive. Case sensitivity affects the sort order and the uniqueness of values. In *dBASE SE*, you can create case-sensitive indexes only, although *dBASE SE* maintains case-insensitive indexes when you edit tables that use them.

Each table should have one primary index, although it is not required. In a Paradox table, the primary index is stored in a file with a .PX extension.

Unique keys

Primary indexes require unique values—they do not permit duplicate key values. For example, if a dBASE table has a primary index on ORDER_NO, you cannot add two orders with the same order number—only one can exist in the table. In a composite index, individual field values can be duplicates, but the combined value of all key fields must be unique. (Secondary indexes do permit duplicate values.)

When you create the primary index, use a field that will contain a unique value for each row, such as a customer number field.

A table can have only one blank (empty) value in the keyed field, because subsequent blank values are considered duplicates. Therefore, key fields usually require entries.

Note Some field types, such as memo, OLE, binary, and logical, are unavailable as key fields.

Secondary indexes, maintained and non-maintained

The dBASE Level 7 and Paradox table types permit two types of secondary indexes:

- *Maintained* secondary indexes are automatically maintained when data changes in the table. *dBASE SE* lets you create maintained secondary indexes, and it updates maintained indexes automatically when you edit a table.
- *Non-maintained* secondary indexes are not automatically updated when the table is open. *dBASE SE* does not let you create non-maintained secondary indexes, but it supports any existing non-maintained indexes.

The dBASE table format lets you create maintained secondary indexes regardless of whether the table has a primary index. You can create as many single-field (simple) indexes as there are fields in a table, and you can create up to 255 multiple-field (called *complex* or *composite*) indexes per table.

Creating primary indexes

You can create a primary index in the Table designer or the Manage Indexes dialog box. If the table type you are creating does not support primary indexes, these options are not available. dBASE Level 7 and Paradox table types both support primary indexes.

To create a primary index,

1. Open the table in the Table designer.
2. Choose Structure | Define Primary Key to display the Define Primary Key dialog box.
3. Choose the Primary Key fields from the Available Fields list. Click the arrow to add (or remove) fields from the Fields Of Primary Key list box.

Note In the case of Paradox tables only, the first field in the table must be the primary key or part of a composite primary key. If the field you want to be the primary key is not currently the first one in the table, you have to move it up in the Table designer to be the first field. The dBASE format does not share this limitation.

Creating secondary indexes

You can create one or more secondary indexes in the Manage Indexes dialog box.

1. Choose Structure | Manage Indexes to display the Manage Indexes dialog box.
2. Click the New button. The Define Index dialog box appears.
3. Select fields from the Additional Fields select box and click the arrow to add each one to the fields Of Index Key box. The double-right arrow adds all the fields at once.
4. Choose Ascending or Descending order, assign a name to the indextag, and click OK.

Referential integrity

Referential integrity validates and updates the data in the linked key fields of a relational database. In a relational database, a field or group of fields in one table (the child table) refers to the key of another table (the parent table). Referential integrity rules ensure that only values that exist in the parent table's key are valid values for the specified fields of the child table.

You can establish referential integrity only between like fields that contain matching values. For example, you can establish referential integrity between two tables that both have a field that holds the customer number. The field names do not matter as long as the field types and sizes are identical.

dBASE SE lets you establish referential integrity for any file type that supports it, such as dBASE and Paradox table types. Some SQL-server tables also offer referential integrity. See your SQL-server database documentation to determine if your table type supports referential integrity.

The way referential integrity is used depends on the way you have set up indexing for the tables in a relational database. This section assumes you are familiar with the concepts of index creation and management.

Defining referential integrity

You can establish referential integrity between tables in the current database. If no database is specified, you can establish referential integrity between tables in the current directory.

To define a referential integrity relationship,

1. In the Navigator, Tables tab, use the Look In box to select a database alias or a directory containing tables (such as .DBF or .DB type) that support referential integrity.
2. Choose File | Database Administration. The Database Administration dialog box appears (To see SQL databases listed in this dialog box, you must first have given them a BDE alias.)
3. Specify a Table Type that supports referential integrity, such as dBASE or Paradox, then click Referential Integrity. The Referential Integrity Rules dialog box appears
4. Choose New. The New Referential Integrity Rule dialog box appears. All tables in the current database or directory appear in the Parent Table and Child Table drop-down lists
5. Choose a parent table from the Parent Table list. The table's key fields appear in the Primary Key Fields area of the dialog box.
6. Choose the child table from the Child Table list. Fields available for referential integrity appear in the Available Child Fields list.
7. Specify whether the tables are in a one-to-one or one-to-many relationship in the Relationship panel. The relationship you choose changes the available child fields.
   - One-to-one relationships can be defined between the primary key field in the parent and the primary key field in the child, or any field in the child that has a unique index.
   - One-to-many relationships can be defined between an indexed field that is not the primary key in the child and the primary key field in the parent.
8. Choose the child table's field in the Available Child Fields list and click the Add Field arrow. The field name appears in the Related Child Fields area of the References panel.

   You can establish referential integrity with a complex (or composite) key. If the parent table has a complex key, add fields from the Fields list to match all of the fields in the parent's key.
9. Select the update and delete behavior you want (see below).
10. Optionally change the rule name *dBASE SE* provides in the topmost box.
11. Choose OK to save the referential integrity relationship.

Note If you attempt to define referential integrity on a table that already contains data, some existing values may not match a value in the parent's key field. When this happens, the operation fails and you receive an error message.

Update and delete behavior

You can specify the following rules for updating and deleting data in a parent table that has dependent rows in a child table:

- **Restrict:** You cannot change or delete a value in the parent's key if there are rows that match the value in the child table.

  For example, if the value 1356 exists in the Customer No field of Orders, you cannot change that value in the Customer No field of Customer. (You can change it in Customer only if you first delete or change all rows in Orders that contain it). If, however, the value doesn't exist in any rows of the child table, you can change the parent table.
- **Cascade:** Any change you make to the value in the key of the parent table is automatically made in the child table. If you delete a value in the key of the parent table, dependent rows in the child table are also deleted.

The availability of cascading updates and deletes varies according to the table type:

- **dBASE Level 7**: Cascading updates or deletes permitted
- **Paradox**: Cascading updates only
- **Oracle**: Cascading deletes only
- **Sybase**: No cascading updates or deletes permitted
- **InterBase**: No cascading updates or deletes permitted
- **Microsoft SQL Server**: No cascading updates or deletes permitted

Changing or deleting referential integrity

You can choose any referential integrity name from the list of named referential integrity relationships in the Referential Integrity Rules dialog box to either modify or delete it.

- Choose Edit to open the Edit Referential Integrity Rule dialog box with the selected referential integrity relationship filled in. You must be able to obtain exclusive access to all tables involved in the referential integrity when you modify it.
- Choose Drop to delete the selected referential integrity relationship.

Chapter

15

Editing table data

To browse, change, or add to data in a table, open the table in Run mode. You can use any of three different layouts: grid, form, and columnar.

Though the various data sources supported by *dBASE SE* (see “Supported table types” on page 14-1) have different capabilities, limitations and structures, the same basic procedures for running tables from within *dBASE SE* apply to all.

This section describes how to use *dBASE SE*’s built-in data-editing capabilities to

- Scan information
- Find or replace information
- Perform data entry (add information)
- Delete and undelete information
- Save or abandon changes
- Operate on a subset of information
- View and edit special field types

A few words of caution

Like any file, tables and the information they contain can be corrupted or destroyed if used improperly.

If, for example, you design an application that prevents entry of a number greater than 10 in a field called NUMITEMS, and the existing data is contained in an older .DBF table, someone could circumvent your data constraint of "not greater than 10" by opening the table in Run mode, adding a new row with a value of 11 in the NUMITEMS field, and saving the table.

Most table types, including the new .DBF7 format, allow the table developer to enforce rules at the table level in order to prevent such problems from occurring. But even with such safeguards, developers should caution users who have access to tables that database integrity can be compromised by editing table data directly in *dBASE SE* or in any other application—including spreadsheets and word processors.

Developers can also take other simple precautions, such as placing data in folders at levels casual users may not venture into, or naming data directories with numbers instead of with tempting titles like "databases."

It also helps to understand exactly when and how your databases can be opened and modified from within *dBASE SE*.

Running a table

To view or edit a table, open a table in Run mode using any of the following methods.

- Menu: Choose File | Open (Alt+FO). The Open File dialog box appears. If it’s not already selected, choose the Tables (*.dbf, *.db) item from the Files Of Type list and locate a table on your local drives, or select an alias from the Database list. Choose the View Table Rows option at the bottom of the dialog box. Select your table, and click Open.

- Project Explorer: Select a table, right-click, and choose "Run" from the context menu.
- Navigator: Choose the Tables tab, then use the Look In drop-down list to choose a local folder or alias (or use the Browse button to choose an unlisted folder). Then double-click a table icon. You can also open a selected table by clicking the Run button on the toolbar or by pressing F2.
- Command window: type

  use <tablename>

  where <tablename> is a local table file name or aliased :database:table reference, for example, :MSSQL1:mytable. You may include the full path to the file name. Then type

  edit

The table appears in a grid of rows and columns. Each column is a field. You can browse or edit all the data in the table.

Protected tables

When you open a protected table, complete the Group, User, and Password fields in the Login dialog box, and choose OK. The database administrator assigns groups, users, and passwords for table protection.

Also note that some tables may have read-only protection or other access and editing restrictions.

Search the Help index for "security" for more information about protected tables, access rights, and encryption.

Table tools and views

When you run a table or query, additional items appear on the View menu, and a Table menu becomes available.

Figure 15.1 Table-editing toolbar

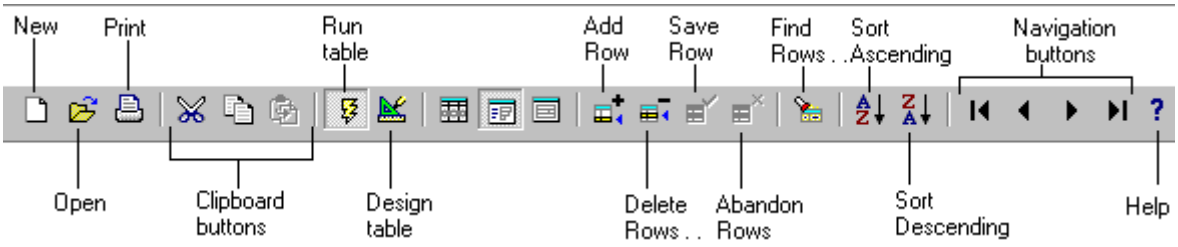
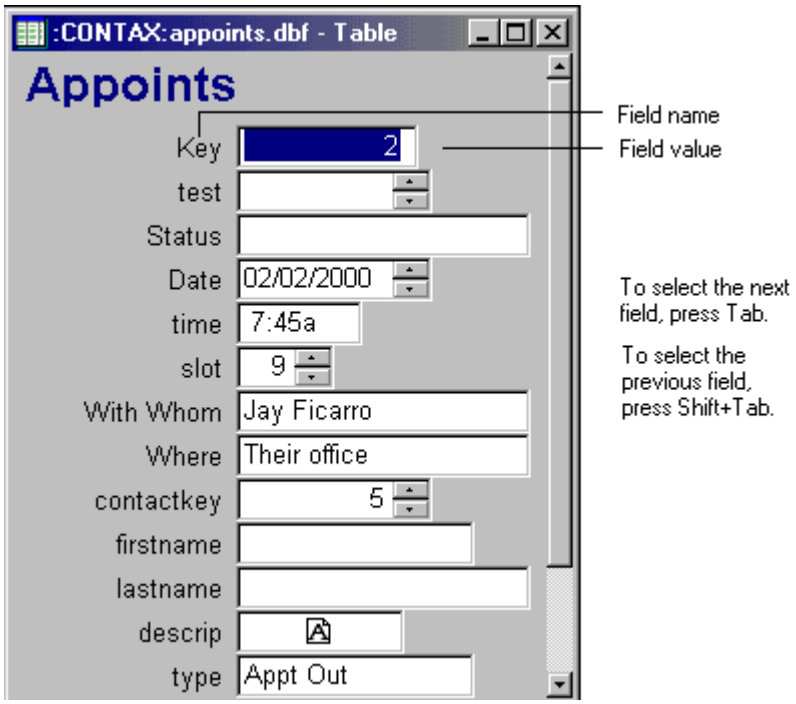


Table and query views

The default view when running tables or queries is a grid view. Other options are a columnar view, which displays a single row on each page with fields arranged vertically, and a form view.

Figure 15.2 Columnar view



You can choose your preferred view any time in two ways:

- Choose the desired view from the View menu.
- Click the appropriate button on the toolbar: Grid Layout, Columnar Layout, or Form Layout.

dBASE SE remembers your last view choice and next time you open a table, it will be in the view you last chose.

Adjusting the view

In columnar view, fields remain at their default size and cannot be widened. You can, however, enlarge the view window vertically to see more fields at once.

- In grid view, you have other options:
- You can resize columns and rows by pointing to a column or row border, then dragging when the pointer changes to a double-headed arrow.
- You can move columns by dragging field titles to new positions.

Viewing only selected table data

To open a table with only specified rows available for viewing or editing, create and run a query—an SQL statement that requests specified information from one or more tables.

To view or edit data selected by a query, open the query in Run mode using any of the following methods.

- Menu: Choose File | Open (Alt+FO). The Open File dialog box appears. If it's not already selected, choose the SQL (*sql) item from the Files Of Type list and locate a query on your local drives. Then choose the Run SQL option at the bottom of the dialog box. Select your query, then click Open.
- Project Explorer: Select the query you want to run, right-click it, and choose Run from the context menu.
- Navigator: Choose the SQL tab, then use the Look In drop-down list to choose a local folder (or use the Browse button to choose an unlisted folder). Then double-click a query icon. You can also open a selected query by clicking the Run button on the toolbar or by pressing F2.

The query results appear in a grid of rows and columns. What you edit here is reflected in the table or tables that contain the data.

For more information on creating, modifying and running queries, search for "SQL" or "queries" in the Help index.

Table navigation

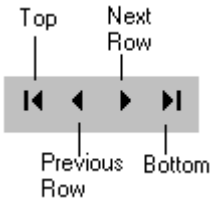
dBASE SE uses a row pointer to identify the current row. Use the following methods to move the row pointer in a table:

Table 15.1 Navigating rows using the menu, mouse or keyboard

Move to	Menu	Mouse	Keyboard
Next row	Table | Next Row	Click next row	PgDn (columnar)
Previous row	Table | Previous Row	Click previous row	PgUp (columnar)
First row	Table | First Row	Scroll to top of table, if necessary, and click first row	Ctrl+Home
Last row	Table | Last Row	Scroll to bottom of table, if necessary, and click last row	Ctrl+End
Specific row	Table | Find Rows	Click row	Ctrl+F
Previous page	Table | Previous Page	Click in the scroll bar	PgUp (grid)
Next page	Table | Next Page	Click in the scroll bar	PgDn (grid)

In addition, you can use the navigation toolbar buttons.

Figure 15.3 Navigating rows using the toolbar



Note Your Desktop Properties settings might cause the exclusion of certain rows in a table. For example, if Deleted is selected on the Table page in the Desktop Properties dialog box, the row pointer skips deleted rows. Similarly, if you've specified a scope in the Table Rows Properties dialog box, the row pointer ignores rows outside the scope. If you've specified a filter for the table, rows not meeting the filter condition are ignored. For details on these and other Table Property settings, see Help (search for "table properties").

Data entry considerations

When you're faced with day-to-day data entry tasks, consider the following:

Should I run a table or use a form? Running a table offers quick direct access to tables from within *dBASE SE*. This is handy for occasional editing, data entry, and maintenance. However, forms offer more control over the data-entry process, including the ability to edit multiple linked tables in the same window and to programmatically enforce entry validation and data integrity. If a data-entry form doesn't already exist, you can use the Form wizard to quickly create one. For ongoing data entry and maintenance, consider designing a form.

Editing all rows or selecting only the information you need. You sometimes want to work with only a subset of rows in a table, especially if the table has a large number of rows. For example, you might want to change orders for the current month only. Consider the following approaches:

- Use queries to select the rows you need to change and ignore the rows that don't apply to the task at hand. One advantage to using queries is that they allow you to store the conditions you specify and use them with multiple tables.
- Use a conditional or unique index that includes only the rows you want.

Working with parent and child tables. Deleting rows or changing the values in linked fields or key fields can cause *dBASE SE* to lose track of data. For example, if you delete an order in the Orders table but not in the associated rows in the LineItem table, you end up with orphaned rows that could skew calculations. Similarly, you might inadvertently change the order number in the Orders table but not in the LineItem table, which also results in orphaned rows.

If the table you are editing is part of a parent-child relation, consider using a query to link the parent and child table, rather than editing the single table. The query helps you see and preserve connections between related rows in the tables.

Ordering rows. During data entry, you can use the natural order of the table or you can use an index. When searching for rows to update, using an index could be the most efficient means, particularly in a table with many rows.

Selecting a view for entering data. When you run a table, three views are available: grid (default), form and columnar. Choose the one that best suits your data entry task.

Repeated values. If you are entering the same value repeatedly, consider using the Replace option to update a number of rows with the same value quickly.

Note The Data Entry page of the Desktop Properties dialog box offers a number of data entry configuration options, including Bell, Confirm, Delimiters, and Type-ahead. For details on these options, click the Help button on the Data Entry page.

Finding and replacing data

dBASE SE provides tools that let you search for information in a table and update rows with new information.

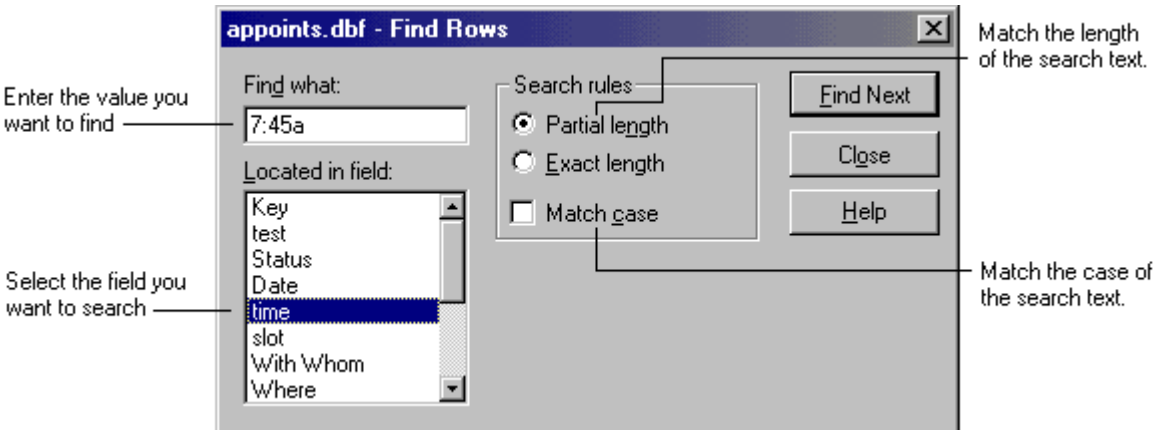
Searching tables

In addition to scrolling through rows, you can quickly find the row you want by searching for a value in a field you select. For example, you could quickly find a specific customer order by selecting the ORDER_NO field and typing the number of the order you want to find. You can search character, numeric, float, date, and memo fields.

To begin a search, click the Find Rows button, or choose Table | Find Rows.

The Find Rows dialog box provides options you can use to focus and speed up your search. The options you use depend on the search value you specify, the way information is organized in the table, how specific the search needs to be, and how much of the table you want to search.

Figure 15.4 Find Rows dialog box



- **Find What** In your search text, you can specify any printable character, including spaces. The search string can be as long as the width of the search field. In general, the longer the search string, the greater the precision

required. If you can't find a match with the current search string, shorten it to increase your chances of finding a match.

- **Located in Field** You can search for text in any field, whether or not it has been indexed. Searching is fastest when you search on an indexed field. Before you start your search, select the index you want to use as the master index.
- You can also search non-indexed fields, such as memo fields. Doing so might be slower than an indexed search, particularly in tables with many rows.
- **Partial Length** There is no requirement that the length of the search string be identical to the field value. This rule is checked by default.
- **Exact Length** To be a match, the search string must appear in the field just as you type it.
- **Match Case** Match Case requires that the field value match the search string exactly, including uppercase and lowercase letters.

Once you have selected the options you want, click Find Next. If a match is found, the row pointer moves to the matching row and the row appears highlighted. If no match is found, a message appears.

If you don't find the match on the first try, shorten the search string or adjust other search options as needed and try again.

Replacing data in rows

You can find text in a table and replace it with different text. For example, if you change the name of a product, you can search a table and replace the old name with the new one. The replacement can occur in a different field than the find string is in.

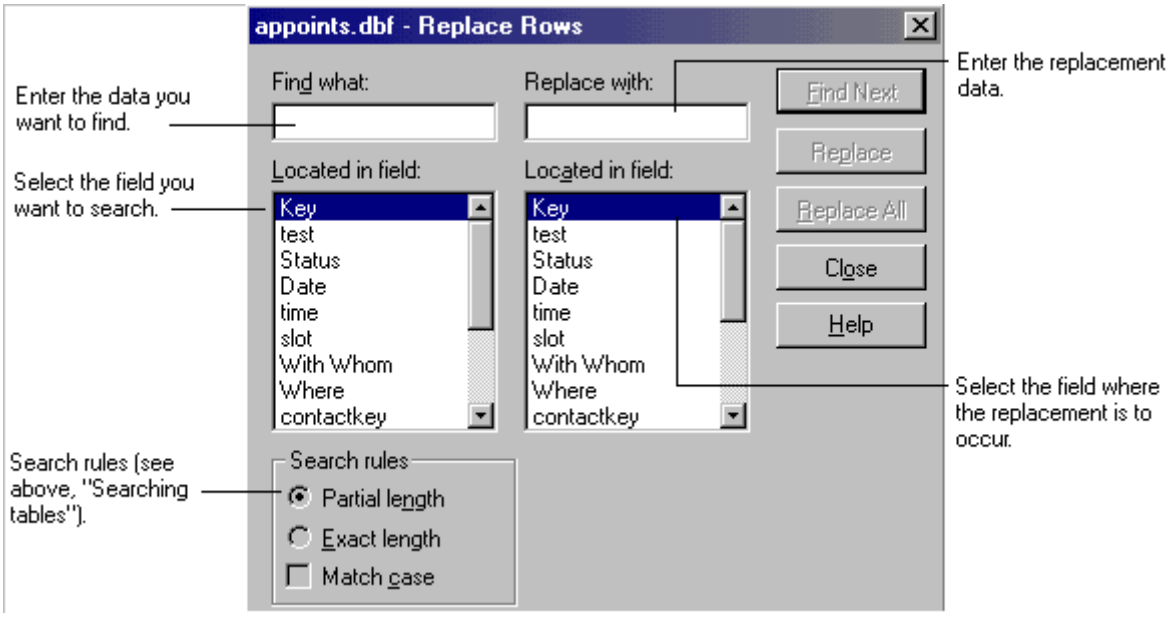
For example, suppose you assign a salesperson to a different sales territory and want to update the SALESPERSON field for all customers in that territory. Rather than update each customer row individually, you could simply select all the customer rows in that territory, then replace the SALESPERSON field in those rows only.

Important Updating indexed fields in the master index can yield unpredictable results because changing the key value changes the row position—as well as the row pointer—in the index. Use a different master index instead. In addition, changing key values in related tables can result in orphaned rows in the child table. Therefore, carefully consider the implications of updating rows, and make a backup copy of your tables before proceeding.

To replace rows,

1. Select the table, then choose Table | Replace Rows. The Replace Rows dialog box appears.
2. Complete the dialog box.
   - The replacement value you specify must match the data type of the selected replace field. Make sure that the value fits in the field.
   - In character fields, if the text is too long for the field, it is truncated.
   - In number fields, if the value exceeds the field size, the fields fill with asterisks.
   - In memo fields, the existing memo text is overwritten with the replacement text. The replacement text must be in character format.
3. Once you've specified the replacement text, do one of the following:
   - Choose Find to find the next occurrence of the search text. Then choose Replace to replace it, or choose Find to leave it alone and go to the next occurrence.
   - Choose Replace All to replace all occurrences of the search text.

Figure 15.5 Replace Rows dialog box



Adding rows to a table

When you add rows to a table, they are appended to the end of the table. An empty row is added at the end of the table where you can enter data. If a table contains 100 rows before you append, the new row becomes row number 101 and the current row for editing.

To append a row, do one of the following:

- Choose Table | Add Row.
- Click the Add row button on the tool bar.
- In grid view, go to the last row, and then press the Down arrow.

An empty row appears. Now you can enter data. See “Data entry considerations” on page 15-4 and “Viewing and editing special field types” on page 15-10.

Note If you’re adding rows to a table with an active index, each row appears at the end of the table. When you finish entering data in the row, *dBASE SE* updates the index and moves the row to its correct position as indexed. The last row you added remains the current row for editing.

Deleting rows

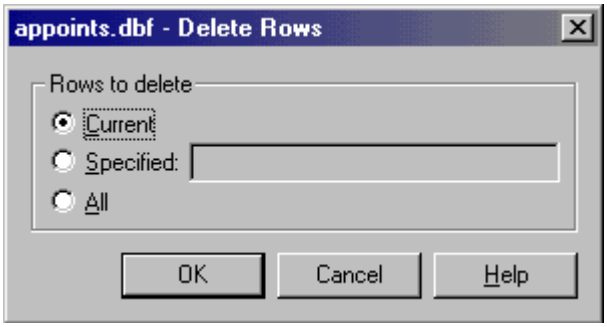
To delete a row in grid view, select the row by clicking the button at the left of the row, then press Ctrl+U. In columnar view, navigate to the row and press Ctrl+U.

You can also delete rows through the Delete Rows dialog box:

1 Find the row in the columnar view or select the row in the grid view.

2 Choose Table | Delete Rows, or click the Delete button on the toolbar. The Delete Rows dialog box appears.

Figure 15.6 Delete Rows dialog box



3 Click OK to delete the currently selected row, or specify a particular row number, or choose all to delete all rows. Click OK.

Saving or abandoning changes

dBASE SE saves changes to a row automatically whenever you do one of the following:

- Move the row pointer to another row.
- Toggle the table view between grid and columnar view.

Note If Autosave is selected in the Table page of the Desktop Properties dialog box, *dBASE SE* writes changes to disk automatically. Otherwise, it accumulates changes and saves them to disk periodically.

- To save a row manually, do one of the following,
- Close the Table window, or
- Click the Save Row button

To abandon changes to a row, click the Abandon Row button.

Performing operations on a subset of rows

Sometimes you need to work with only a subset of the rows in a table. You can save time by selecting only the rows you want to work with, and avoid processing rows that don't apply.

This group of topics explains how to

- Select sets of rows to process
- Count rows that meet a given set of criteria
- Perform calculations on multiple rows
- All of these features are found in the Table menu.

Many of the operations described in this section—including aggregate operations for calculating data, finding rows, filtering conditions, and linking parent and child tables—can also be performed by creating queries. For information, search the Help index for "SQL" or "queries."

Selecting rows by setting criteria

To select a subset of rows from your table, you have to identify the criteria by which rows qualify for selection.

For example, if you want to calculate the average sales volume of customers in Texas, your criteria might be that the STATE_PROV field in the Customer table contain the value "TX". Rows that fail to meet this criteria are ignored.

Another row selection technique is to specify a condition by writing an expression in the Command window that defines which rows qualify for processing.

Setting For conditions

Use the For condition to select rows that appear throughout a table rather than in a contiguous group. For conditions, check all rows in the table when determining which rows qualify for processing. Processing starts at the top of the table and goes to the bottom (unless you limit it using one of the options discussed in the previous section).

dBASE SE compares each row with the condition you specify to determine whether to process a row. For example, to count the number of customer orders that exceed $10,000, you might specify the following For condition: TOT_INV > 10000.

Setting While conditions

Use the While condition to select a series of rows that appear consecutively in a table. While conditions check only the current row and subsequent rows when determining which rows qualify for processing. Processing starts at the current row rather than at the top of the table. Therefore, the row pointer must be at the first qualified row before processing; otherwise, no rows can be selected.

This method works best when you use an index whose key matches fields in the condition. That way, you can quickly find the first row in the series, then process rows sequentially. Processing ends when the key no longer matches the condition.

For example, to do a calculation only on rows of customers in Texas, it would speed up processing to apply an index on STATE_PROV, which would group all the TX rows together. Search for the first TX row, and then enter STATE_PROV="TX" in the While field to process from the current row through the last TX row.

You can also create or modify an index to include a subset of rows. In this case enter a For condition that specifies STATE_PROV="TX". When that index is active, only TX rows are selected.

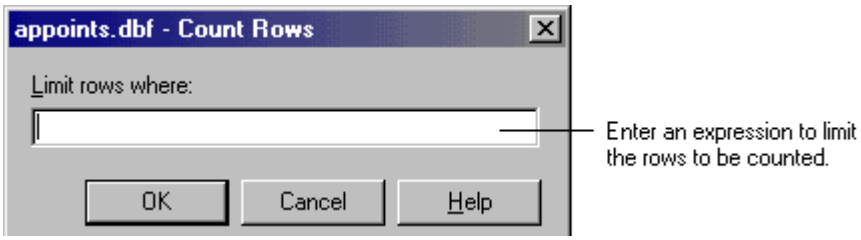
Counting rows

You can count rows to determine how many rows meet a given set of criteria. For example, you might want to know how many customers fall within a certain zip code range, or how many orders were taken on Tuesday.

To count rows,

1 Choose Table | Count Rows. The Count Rows dialog box appears.

Figure 15.7 Count Rows dialog box



2 Specify the rows to include in the count, then choose OK.

dBASE SE counts the number of rows that meet the criteria, then displays that number in a message box.

Performing calculations on a selection of rows

You can perform calculations on number fields to obtain useful information from a selection of rows. For example, you might calculate the total sales for a given month, the largest sale, the smallest sale, or the average sale amount.

You can perform the calculations shown here

Table 15.2 Types of calculations

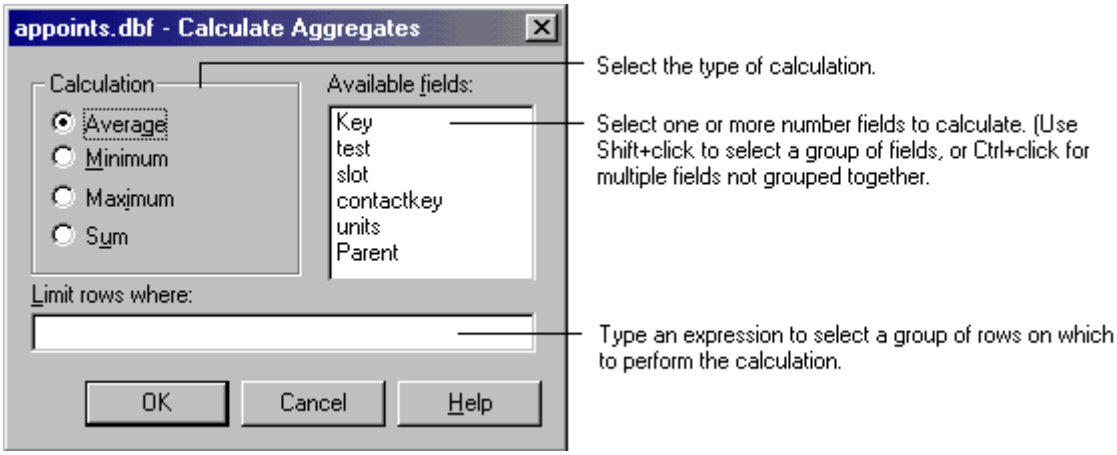
Calculation type	Result
Average	Average field value in selected rows
Minimum	Minimum field value in selected rows
Maximum	Maximum field value in selected rows
Sum	Sum total of field values in selected rows

Most calculations work on numeric and float fields only, but Maximum and Minimum can also be used with date and character fields.

To calculate values,

1 Select the table, then choose Table | Calculate Aggregates. The Calculate Aggregates dialog box appears.

Figure 15.8 Calculate Aggregates dialog box



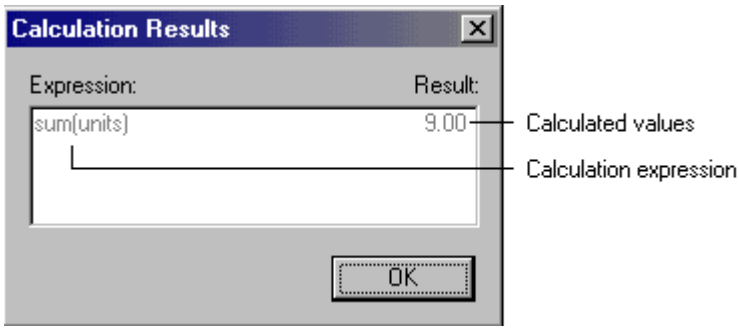
2 Choose the type of calculation you want to perform, then select one or more of the listed fields. (Use Shift+click to select several contiguous fields, or Ctrl+click to select several noncontiguous fields.)

   Optionally, you can type an expression in the Where box to select a group of rows on which to perform the calculation

3 Click OK.

dBASE SE performs the calculation and displays the results in the Calculation Results message box.

Figure 15.9 Calculation Results dialog box



Viewing and editing special field types

Supported field types and field-editing rules can differ greatly from database to database. In many cases, entering and editing data is intuitive enough: select a field and type in characters or numbers, depending on the

field type. If you enter a value that doesn't match the basic field type rules (like entering a character in a numeric field), you'll receive an error message.

Most databases also offer a number of non-character field types, however, and viewing and modifying data in these types is a bit different.

This group of topics examines binary (image or sound), OLE, and memo field types, all of which are available in both the older .DBF and the new .DBF 7 table formats. Other table formats may offer other ways to edit similar field types.

Viewing the contents of special field types

Memo, image, sound, and OLE fields are represented in a table by icons. You can view the contents of these types of fields in three ways:

- Select the field and press F9.
- Double-click the field.
- Select the field and choose View | Field Contents.

To select a field with the keyboard, use the following keys

Table 15.3 Field selection keyboard shortcuts

Go to	All views
Next field	Tab
Previous field	Shift+Tab
Beginning of field	Home
End of field	End

Memo fields

Memo fields open in a text editor. When the text editor is open, the Format toolbar becomes available, and you can format text in the memo field.

Binary fields

Your tables can contain any supported sound and image data, and the data can be stored, viewed (or played, in the case of sound files), added, or replaced any time you run a table.

.DBF and .DBF 7 tables support most popular image formats (for a complete list, see the Image page in the Navigator). The supported sound format is .WAV.

Importing an image or sound into a binary field

To add an image or sound to a binary field,

1. Double-click a binary field. The Specify Binary Field Subtype dialog box appears.
2. Choose the binary type (Image or Sound). If you choose Sound, the built-in Sound Player appears. If you choose Image, the built-in Image Viewer appears.
3. Right-click on the viewer or player. A file import dialog box opens. Select a file of the appropriate type, then click OK.
4. Save the row.

If the field already contains a binary image or sound, you can export the image or sound to disk by calling the player or viewer as instructed above, then choosing Export from the right-click menu.

OLE fields

Object linking and embedding (OLE) lets you use objects from other Windows applications in your tables.

You can either link objects to or embed objects into OLE fields. Linking inserts a reference to the file from which the object originated, which means that in order to keep the object updated, both the source file and source application must remain available. If the linked object is updated, your OLE field is updated as well.

Embedding places an entire object into the OLE field. Embedding is a more portable solution, but still requires that the application that created the source be available. It can also cause significant enlargement of your table file sizes, which grow by the size of each object you embed plus some OLE reference code for each. And unlike links, embedded objects are not updated when the source object changes. Instead, they become separately editable (in the source application) objects of their own.

An OLE object can be a graphic image, a sound, a document created by a word processor, or any other object or document that can be created by an OLE-compliant server application. For example, Microsoft Word is an OLE server, and any document created in Word can be linked or embedded into an OLE field.

In any OLE exchange, *dBASE SE* then becomes the client application.

Whether you choose to link or embed an object into your OLE field, you can launch the server application and load the object for editing by simply double-clicking the OLE field in your running table.

Adding an OLE object to an OLE field

To add an object to an OLE field

1. Start the server application and open a file or create an object.
2. If you want only a portion of the object data, select it and copy it to the Clipboard using the application's Copy command or Ctrl+C.
3. Start or switch to *dBASE SE*, and run the table containing an OLE field.
4. Click the Add Row button or choose Table | Add to open a new row. If the OLE field is represented by an icon, double-click the icon to open the OLE viewer.
5. Do one of the following:
   - To link the object, right-click the OLE viewer and choose Paste Link from the popup menu.
   - To embed the object, right-click the OLE viewer and choose Paste (Ctrl+V) from the popup menu.

The linked or embedded object appears in your OLE viewer. You may need the scroll bars to scan the entire object.

A linked object is automatically updated by default, so if the source object is edited, the OLE field will reflect the changes. You can modify link attributes, however—as well as view information on the link, open the source file, change the link (useful if the source file is moved) or even break the link—by right-clicking the OLE viewer and choosing Links from the popup menu.

To edit an embedded object, double-click the OLE viewer containing the object. The server application opens with the object loaded for editing. When you're finished with your edits, update the OLE field by choosing Update from the server application's Edit menu.

You can also link or embed OLE objects by right-clicking an OLE viewer window and choosing Insert Object from the popup menu. The Insert Object dialog box lets you choose from among the OLE object types registered on your system. You can then either create a new object in the server application (for embedding) or create an object from an existing file (for embedding or linking).

Removing an OLE object from an OLE field

To remove an OLE object from a table,

1. Locate the OLE field that contains the data you want to remove.
2. Double-click to open the OLE viewer (if it's not already open).
3. Select the viewer window and choose Edit | Delete.

Chapter

16

Setting up security

dBASE SE provides built-in levels of security against unauthorized access to encrypted databases and tables. This table-level security depends on data encryption.

Sensitive tables should always be encrypted by using the database vendor's administration software. *dBASE SE*'s password dialog is presented whenever a user tries to access a form linked to an encrypted table or database. The user's response to the password dialog is passed to the encrypted table or database for verification before *dBASE SE* will display the form. See your database vendor's documentation about security administration for SQL, ODBC, or non-Standard systems.

The DBF and DB tables you create within *dBASE SE* have built-in encryption. *dBASE SE* provides direct database administration security access to set passwords for BDE-Standard DB and DBF tables, as well as the extensive user-access and privilege-level security features of DBF tables.

Setting up security strategies

dBASE SE offers two general strategies to handle access to encrypted tables of any type: individual login and preset access.

- Individual login via automatic password dialogs
  In this approach, each user is required to login every time he or she tries to access a form linked to an encrypted table or database. *dBASE SE* automatically displays a password dialog for the appropriate table type, requiring the user to enter a password or other information required by the table. Users might get different access levels, depending on their user name and password, and depending on the security features supported by the table type. The user must submit the correct information (which is passed to the encrypted database system for verification) before the user can access the *dBASE SE* form.
- Preset access via Session and Database objects
  Preset access involves hard-coding passwords or user names in *dBASE SE* forms and reports. Preset access provides an automatic, pre-determined level of access without login procedures for certain groups of users. It can be used in conjunction with individual login to provide easy read-only access for the public and login-protected access for authorized company personnel.
  - Preset access for Standard table types
    Sessions objects provide unique connections between a user and a DBF or DB table. You can add methods to these objects to restrict access to certain features of a Standard table, or to make the table read-only for certain login-levels.
  - Preset access for SQL and other table types
    Database objects link *dBASE SE* forms to SQL databases or table sets. You can set the Database object's *loginString* property for particular user names or passwords to limit users of a specific login-level to read but not write the data in an SQL database.
- Table-level security for DBF tables
  dBASE SE supports direct access to the extended security features of DBF tables, including administrator security, up to 8 user access levels, and three-level privilege security for DBF tables and individual fields. If

you intend to create tables within *dBASE SE*, DBF tables offer the most extensive and versatile security features.

- Table-level security for DB tables
  dBASE SE provides direct access to master password security for each DB table. However, you must use Borland's Paradox or Database Desktop to set auxiliary passwords.

Individual login via automatic password dialogs

dBASE SE's password dialogs are activated only by encrypted tables. Password protection alone is inadequate to protect a sensitive table unless the table is encrypted, because an intruder, having gained access to the machine or server on which the application is running, could use another application to read the data from the hard disk.

Once an authorized user gains access to your application by providing the correct password, the user might be offered a restricted choice of a variety of tables, with different access privileges, depending on the login level used to access the application. *dBASE SE* supports the full range of table- and row-level security features for DBF tables, so you can create up to 8 user access levels and 3 privilege levels, precisely controlling access by different classes of users to specific tables and even to specific rowsets in those tables.

To link any encrypted table to a form (and thereby enable automatic password protection), you need only create a Query object for that table on your form. To do this, simply drag the table icon of the encrypted table from the Navigator's Tables tab to your form surface in Form designer. (At this time, while in Design mode, you will have to supply access information to the encrypted table.) This is all you need to do to ensure that *dBASE SE* will activate the automatic password dialog. See Form Designer for guidance on adding Query objects to forms.

After your form is run and the user activates some event (a button click, for example) to access a restricted table, the Borland Database Engine (BDE) attempts to open that table. Because the table is encrypted, *dBASE SE* automatically displays the appropriate password dialog with fields for the input required by that table type. The type of security available varies according to table type.

The original *dBASE SE* form is temporarily displaced and the user is presented with a password dialog. To gain access to your application (and its underlying encrypted table), the user must provide the particular security information required by that table or database.

Preset access via Database and Session objects

Setting preset access levels is another approach to restricting access to your data.

To set levels, you have to hard-code passwords or user names in *dBASE SE* forms and reports, thus restricting access only to individuals within specified groups.

Preset access can be useful in combination with the individual login approach to provide easy read-only access for general users, and login-protected access for authorized personnel. For example, you might have employee information managed through an application that allows updating by Human Resources personnel, but permits read-only access to other employees.

To implement this strategy, you would need a full-access (read/write) password that the HR staffers would have to enter manually every time they start the application. You would then code into the form a read-only password that would admit everyone else at a limited level.

How you encode preset access levels depends on the table type, as described in the next two topics.

Preset access for Standard table types

For DBF and DB tables, security is session-based. The Session object has a *login*(_) method for DBF table security and a *addPassword*(_) method for DB table security. The appropriate method (or both if you're using both types of encrypted tables) must be called with the correct user name and password before attempting to activate a query that accesses the table.

Where this must occur depends on whether all users are sharing the same session. If everyone accessing *dBASE SE* gets exactly the same access for every application by using the same user name and password, then they can share the default session. You would need only call the session's methods once through an administrative program or form.

All the forms would thus require a Query object to access the DBF or DB tables, but no Database object or Session object, because everyone uses the default database in the default session.

On the other hand, if any two applications use different user names or passwords, then every form must have its own Session object, so that each form runs in its own session and the security is localized.

No Database object is needed because the form uses the default database of its own session. Then users must log into each session before the query is activated.

Use the Query object's *canOpen* event to call the session's security methods.

Preset access for SQL and other table types

Table and database types other than DBF or DB tables accessed via the directory require modification of the Database object's loginString property. This applies to all non-Standard applications, including SQL servers such as Borland InterBase, Oracle, Sybase, Informix, IBM DB2, and MS SQL Server; and ODBC connections such as Access and Btrieve. It also applies to remote DBF and DB tables accessed through a Borland Database Engine (BDE) alias.

A BDE alias always identifies a database. Therefore all non-Standard table security is through the Database object that provides access to that database. In some cases, logins are required to access tables in a database.

The Database object's *loginString* property is a character string that contains the name and password in this form:

```
name/password
```

You can set this property in the Inspector in the Form designer. By setting the name and password in the form's Database object, all users attempting to open that form will get whatever level of access that name and password provides.

Although possible, it's more trouble than it's worth to share a Database object among multiple forms. Each form should have its own Database object, with whatever the appropriate *loginString* is for that particular form.

Table-level security for DBF tables

The security features of DBF tables are extensive. If you intend to create private tables within *dBASE SE* for which you want to set elaborate or varied access levels, the DBF table type is your best choice.

Table-level security relies on data encryption. Data encryption scrambles data so that it can't be read until it is unscrambled. An encrypted file contains data that has been translated from source data to another form that makes the content unreadable. If your database system is protected, *dBASE SE* automatically encrypts and decrypts tables and their associated index and memo files when a user supplies the required passwords or other login information.

In addition, DBF tables allow you to define which fields within tables users can access, and the level of access, read, read/write, or full.

The first parts of this section describe how to plan your security scheme for DBF tables. Topics include

- The various levels of security
- An overview of the various aspects of the DBF security
- Planning group access for each table
- Planning each user's login and user access level
- Planning user access to tables and fields within tables

At the end of this section are procedures for setting up your DBF security scheme:

- Enter the database administrator's password
- Create user profiles
- Set user privileges for table access
- Set user privileges for fields within tables

About groups and user access

You can control access to individual DBF tables (and to fields within those tables) by carefully defining groups of users according to

- Which tables each group can access
- Which privilege levels (read, update, extend, delete) each group has at the table-level
- Which fields within tables each group can access
- Which privilege levels (none, read-only, full) each group has at the field-level

Table access

First, you'll need to define user groups and determine which group has access to which table. Try to organize users and tables into groups that reflect application use (for example, by department or sales area).

- A table can be assigned to only one group. If the user group and table group don't match, the user can't access the table.
- Typically, each group is associated with a set of tables. By associating each application with its own group, you can use the group to control data access.
- A user can belong to more than one group. However, each group that a user belongs to must be logged-in separately.
- If a user needs to access tables from two different groups in the same session, the user must log out of one group, then log in to the second. A user may have separate logins into different groups in separate sessions to access files in different groups.

User profiles and user access levels

You'll need to develop a user profile for each user in each group. As part of each profile, you'll assign to the user an *access level*. Each user's access level is matched with the table's privilege scheme (see the next section) to determine what access the user has to the table and, within each table, to each field. For example, if you establish a read privilege of 5 for a table, users with a level from 1 to 5 can read that table. Users with a level of 6 or higher can't read the table.

By establishing access levels within a group, you can give different users different kinds of access to the table and to fields within the table.

- Access levels can range from 1 to 8 (the default is 1). Low numbers give the user greater access; high numbers limit the user's access. The access value is a relative one—it has no intrinsic meaning.
- The less restrictive levels (1, 2, 3) are typically assigned to the fewest people. To limit access to data, the more privileges a level has, the fewer users you should assign to that level.
- You can assign any number of users to each access level.
- If you don't need to vary the access level of the users within a group, there is no need to change each user's default level.

About privilege schemes

Once you've established each user's access level, you set up a *privilege scheme* for each table. A DBF table's privilege scheme controls three things:

- Which group can access the table. (The user's group name is matched with the table's group name to allow table access.)
- Which user access levels can read, update, extend and/or delete the table (table privileges).
- Which user access levels can modify and/or view each field within the table (field privileges).

After a user logs in, *dBASE SE* determines what access the user has to that DBF table and its fields by matching the user's access level with the rights you specified in the table's privilege scheme.

For example, if you assigned a user an access level of 2, that user's access to the table, and to various fields within the table, are determined by the privileges you assigned to Level 2 in the table privilege scheme.

In building a table privilege scheme, note the following:

- A user's ability to access a table is a function of both the access level of the group and the user's individual access level. However, only the user's access level determines what the user can do with a table once it is opened.
- If you do not create a privilege scheme for a table, all users of the group can read and write to all fields in the table.
- Access rights cannot override a read-only attribute established for the table at the operating system level.

Table privileges

At the table level, you can control which operations each user access level (1–8) can do:

- View records in a table (read privilege)
- Change table record contents (update privilege)
- Append new records to a table (extend privilege)
- Delete records from a table (delete privilege)

When you create a table privilege scheme, all four table privileges are granted initially. That is, all table access levels are 1 by default (1 being the *least* restrictive level).

Field privileges

At the field level, you can control which operations each user access level (1–8) can do:

- Read and write to the field in the table (FULL privilege). This is the initial default.
- Read but not write to the field (READ ONLY privilege).
- Neither read nor write the field (NONE privilege). NONE blocks a user from writing to fields and from seeing fields you do not want to display.

About data encryption

A DBF table is not encrypted until you select it, edit the access levels, and save the privilege scheme.

When a DBF table's privilege scheme is saved, *dBASE SE* encrypts the table, including the production index (MDX) file and the memo (DBT) file, if any. *dBASE SE* also creates a backup copy of the original, unencrypted table. To ensure proper security, the backup files should be archived, then deleted from the system.

Even after a database system has been protected, the database administrator and application programmer maintain control over encryption of copied files.

Planning your security system

This section describes how to plan out your security system for DBF table security. It's a good idea to think through user access and table/field rights before you start creating security profiles.

Follow these general steps to set up a protected database system for DBF tables:

1. Plan your user groups.
2. Plan each user's access level.
3. Plan each table's privilege scheme, including both table privileges and field privileges.

4 Implement your security scheme (see Setting up your DBF table security system).

Planning user groups

Take time to think through the various groupings into which you can divide your users, based on who needs access to which tables. For example, an administrative staff might need to access tables that a sales staff does not, or vice versa. Other groups may overlap; for example a marketing group might need to see some of the administrative tables and some of the sales tables.

It helps to develop a worksheet, to map group access needs in advance. The following table shows one way of organizing this information; use whatever method works best for you.

Table 16.1 Setting user groups

Table	Group	User name
CUSTOMER	SALES	AMORRIS
		BBISSING
PRODUCT	ALL	AMORRIS

Planning user access levels

Next, think about how much access each user needs to the table.

Although there are 8 access levels, you might choose to standardize on just 3 levels; one for full access, one for typical use, and one for minimal access. The next table shows the sample worksheet, expanded to show user access levels.

Table 16.2 Setting user access levels

Table	Group	User name	Level 1 (full access)	Level 4 (typical access)	Level 8 (minimal access)
CUSTOMER	SALES	AMORRIS	X		
		BBISSING		X	
		LJACUS	X		
		FFINE			X
PRODUCT	ALL	AMORRIS	X		
		BANDERS		X	
		BBISSING		X	
		CDORFFI		X	
		LJACUS	X		
		FFINE			X

Planning DBF table privileges

Next, plan each DBF table's privilege scheme.

For each table operation, determine the most restricted access level that can perform the operation. All levels less restricted than the specified one can perform that operation; all levels more restricted than the specified level cannot.

The following worksheet illustrates one way to plan which user access levels grant which table rights.

Table 16.3 Setting table privileges

Table	Read	Update	Extend	Delete
CUSTOMER	8	4	4	1
PRODUCT	8	4	4	1
ORDERS	8	4	4	1

Planning field privileges

The last planning step is to determine which user access levels can read and/or write to fields. Consider developing a worksheet similar to the following one.

Table 16.4 Setting field privileges

Field name	Full access	Read only	No access
PAYRATE	Levels 1–2	Levels 3–6	Levels 7–8
FIRSTNAME	Levels 1–6	Levels 7–8	
LASTNAME	Levels 1–6	Levels 7–8	
SSN	Levels 1–2	Levels 3–6	Levels 7–8

Setting up your DBF table security system

Once you've planned out your security scheme for DBF tables, you're ready to set it all up. Follow these steps to implement the security scheme:

1. In *dBASE SE*, define the database administrator password.
2. Define the user profiles, including group membership and access level.
3. Define table privileges.
4. Define field privileges.
5. Set the login security scheme.
6. Save the security information.

This section describes how to set the database administrator password, how to enter and edit user profiles, and how to set up table privilege schemes.

Defining the database administrator password

Before setting passwords, make sure any open tables have been closed. Follow these steps to enter the database administrator password:

1. Choose File | Database Administration. The Database Administration dialog box appears.
2. In the Database Administration dialog box, make sure that the Current Database field is set to <None> and the Table Type field is set for dBASE (DBF) tables.
3. Click the Security button. The Administrator Password dialog box appears.
4. In the Administrator Password dialog box, enter a password of up to 16 alphanumeric characters. You can enter characters in upper- or lowercase. The password does not appear onscreen.

   The first time you set the administrator password you are prompted to reenter the password to confirm. (Thereafter, the system gives you three chances to enter the password correctly before the login terminates.) The Security dialog box appears.

Warning! Once established, the security system can be changed only if the administrator password is supplied. Keep a hard copy of this password in a secure place. There is no way to retrieve this password from the system.

Creating user profiles

The Security Administrator dialog box is where you create user profiles and establish an access level for each user.

Follow these steps to add a user profile:

1. In the Security dialog box, select the Users tab and click the New button.
2. Enter a user login name (1–8 alphanumeric characters) in the User field. The entry is converted to uppercase. Required.
3. Enter a group name (1–8 alphanumeric characters) in the Group field. The entry is converted to uppercase. Required.
4. Enter a password for this user (1–16 alphanumeric characters). Required.
5. Select an access level for this user (from 1 through 8; see About groups and user access). Lower numbers give the greatest access; higher numbers are the most restricted.
6. Enter the user's full name (1–24 alphanumeric characters). This entry is optional. Because this item is not used in validating a login, you can use it any way you want. Frequently, the full name is used to add a more complete user identification. Alphabetic characters you enter in the Full Name option are not converted to uppercase.
7. Click OK to save the user profile.
8. The Security dialog box reappears with your new user info added to the list on the Users tab.
9. Repeat the preceding steps for each user.

Changing user profiles

To change a user's profile,

1. Open the Users tab of the Security dialog box.
2. Select the user name of the user you want to change, and click the Modify button.
3. Make the desired changes, then click OK.

Warning! Be careful when editing the group name, deleting the group, or deleting all users from a group. If you edit the group name, there is no way for its users to access tables associated with the original group. And if you delete the group or all users from a group before all tables associated with the group are copied out in a decrypted form, no one can access the tables. In that case, you must create a new user for the group.

Deleting user profiles

To delete a user profile,

1. Open the Users tab of the Security dialog box.
2. Select the user name of the user you want to delete, then click the Delete button.
3. To confirm the deletion, click the Yes button.

Establishing DBF table privileges

Follow these steps to define table and field privileges for a table:

1. Open the Tables tab of the Security dialog box.
2. Select a table.
3. Assign the table to a group.
4. Establish the most restrictive access level for each table privilege.

5 Select field privileges for each user access level.

In general, for DBF tables you can use the Tables tab of the Security dialog box to

- Assign a table to a specific group.
- Set table access privileges.
- Set field access privileges for each user access level.

The sections that follow describe these steps in detail.

Selecting a table

To select a table,

1 Open the Tables tab of the Security dialog box. You use the Tables tab of the Security dialog box to create and modify DBF table privilege schemes. The DBF table privilege schemes are saved in the table structure.

2 In the Table field, type the name of the desired table. (Or click the Tools button and select the table.)

3 Click the Modify Table button. The Edit Table Privileges dialog box opens.

Assigning the table to a group

A DBF table can be assigned to only one group. The group name is matched with a user group name to enable data access.

To select a group for the DBF table, click on the down arrow to display a list of the available groups from the Group list in the dialog box. (These groups were created when you created user profiles.)

Setting DBF table privileges

For each type of table operation (see the table below), specify the most restricted access level that can perform that operation.

Table 16.5 Setting DBF table privileges

Privilege	Access granted
READ	View the table contents
UPDATE	Edit existing records in the table
EXTEND	Add records to the table
DELETE	Delete records from the table

To set table privileges, select a value (1–8) for each operation (Read, Update, Extend and Delete) in the dialog box. Remember that lower access numbers indicate the greatest access; higher numbers indicate the greatest restriction.

Note You cannot specify access levels that are logically incompatible. For example, you cannot prohibit Level 6 from having read access, and also permit Level 6 to have update access. To have update access, Level 6 also needs read access.

Setting field privileges

With DBF tables you can establish access for each field by user access level. The following table describes the available field privileges.

Table 16.6 Setting field privileges

Privilege	Access granted
FULL	View and modify the field. This is the default.
READ-ONLY	View the field only (no update capability).
NONE	No access. The user can neither read nor update the field, and the field does not appear.

Note Table privileges take precedence over field privileges. For example, if a table privilege is set for Read but not Update, the only meaningful field privileges are Read-Only or None. You must restrict *table* privileges to protect your data against table-oriented commands like DELETE and ZAP. Restricting field privileges to Read-Only or None without restricting table privileges doesn't protect data against these commands.

The Fields list in the dialog box lists all of the fields in the current table. The Rights buttons display the field privileges for the selected field for access levels 1 through 8. Initially, all field privileges are set to Full.

Follow this procedure to change a field privilege:

1. Select the field.
2. Click the Rights buttons that correspond to the privileges you want to grant for the field *for each access level*.
3. Repeat the process for each of the other fields in the table.
4. Click OK to save the field access privileges.

Warning! Never change the access rights of the _DBASELOCK field of any table. The rights to this field must remain Full for all access levels.

Setting the security enforcement scheme

You can choose one of two enforcement schemes:

- Force a login when a user attempts to load *dBASE SE* itself.
- Force a login when a user tries to view a form linked to an encrypted DBF table. In this scheme, anyone may use unencrypted tables, but unauthorized users are prevented from accessing protected tables.

To change the security enforcement scheme, follow these steps:

1. Open the Enforcement tab of the Security dialog box. The two radio buttons on the Enforcement page indicate the security enforcement scheme currently in effect.
2. Select the enforcement scheme you want: whether to display a password dialog when loading *dBASE SE* or only when accessing an encrypted table.
3. Click Close.

Table-level security for DB tables

Although DB tables do not offer the extensive user the access-level and privilege- level security system available to DBF tables, DB tables (unlike DBF tables) support passwords.

You can use *dBASE SE* to assign master passwords to DB (Paradox) tables. Once you have assigned a master password assigned to a DB table, it cannot be opened without supplying the password, either by you locally or by users over the Internet.

You may choose to create a single master password that opens all DB tables. A user with this password need see only one password dialog to gain access to all DB tables. Or you may set unique passwords for particularly sensitive DB tables.

Note In addition, auxiliary passwords are supported by DB tables, but you cannot access this feature from *dBASE SE*. Auxiliary passwords allow you to create multiple individual passwords for each DB table, so that you can restrict access to certain tables and certain fields. Different users can be given different passwords that will open only a

specific set of DB tables or allow read/write access to only certain fields within those tables. However, to set auxiliary passwords for field rights to a DB table you must use Paradox.

The process of assigning passwords is initially very similar to that described previously for DBF tables. To assign a master password to a DB table, follow these steps:

1. Make sure the DB table you want to secure is closed.
2. From the File menu, select Database Administration. The Database Administration dialog box appears.
3. Make sure that the Current Database field is set to <None> and the Table Type field is set for Paradox (DB) tables.
4. Click the Security button to open the Security dialog box.
5. Select the name of the table in the Table list. If the table is not in the current directory, use the Folder button to select the directory.
6. Click the Edit Table button to open the Master Password dialog box.
7. Enter the new password for the table in the Master Password field. The password can be up to 31 characters long and can contain spaces. Paradox passwords are case-sensitive.
8. Enter the password again in the Confirm password field.
9. Click the Set button to save the password.

Removing passwords from DB tables

To remove an existing password from a DB table, follow Steps 1 through 6 in the previous section. When prompted, enter the existing master password for the table. Then click the Delete button to remove the password from the table.

Chapter

17

Character sets and Language drivers

dBASE SE is an international software tool which can accommodate many languages. Each language or language category uses its own character set, and each has its own way of sorting and relating its characters. *dBASE SE* provides comprehensive language and character set support with multiple language drivers and automatic OEM–ANSI conversion as needed.

Users new to the Windows environment need to understand the difference between the OEM and ANSI character sets. Users who exchange data across national or linguistic boundaries need to understand how *dBASE SE* uses language drivers to handle data in different languages.

This section of the User's Guide explains how *dBASE SE* uses the OEM and ANSI character sets and discusses techniques for working with language drivers.

Determining the language displayed by the User Interface

Language resources are files containing human-readable text strings that are displayed in the User Interface. In *dBASE SE*, these files are identified, according to language, by a two-letter code at the end of the file name. The online documentation (such as the Help files) are identified in the same way. The codes are:

- en = English
- de = German
- es = Spanish
- it = Italian
- ja = Japanese

While most users install *dBASE SE* for a single language, it is possible to select multiple languages during installation. You may also re-run the Installer at any time to install additional languages.

As with other *dBASE SE* settings, you may indicate a specific preference. If you do not indicate a specific preference, *dBASE SE* will follow the settings of the operating system (see Windows | Control Panel | Regional Settings). This is done on an as-available basis, dropping back to English when a target language resource cannot be found. For example, if the Windows Regional Setting is set to German and the "de" (German language) version of a resource is available, that language resource will be used. A similar protocol is used for loading the Online Help.

You can indicate a specific *dBASE SE* language preference by making a selection from the Desktop Properties dialog. From the Desktop Properties | Country tab, you can access the User Interface Language picker. When you make a selection from the Country tab, your explicit preference will be written in the dBASE_SE.INI file. It will not take effect, however, until you re-start *dBASE SE*. Once restarted, *dBASE SE* will use the new language preference when possible.

At startup, locate the desired core-product language resource file. This file is identified as dBASE_xx.dll - where the "xx" is the two-letter language code. *dBASE SE* then attempts to load a language resource file by checking the following locations:

1. The dBASE_SE.INI file
2. The operating system's Regional Setting.

If a language resource file was not found at either location, *dBASE SE* will default to English and attempt to load any language resource file it finds. The language resource file that is successfully loaded determines the value of the _app. object's *language* property. The value of the _app.language property is the two-letter language code mentioned above.

The _app.language setting will be used for the first attempt at locating relevant files when other language-specific resources and documentation files are needed. For example, when _app.language is set to "it", Italian, and a user invokes the Online Help system, *dBASE SE* will attempt to locate and load a file called "dBASE_SE_IT.HLP". If this cannot be found, the system will attempt to load the file's English version.

About character sets

In the early 1980s, the developers of the IBM PC created an ordered list of symbols known as the *IBM extended character set*. This list contained all the classic ASCII 7-bit characters, together with various mathematical symbols, line and box drawing characters, and some accented characters.

While this was adequate for certain English-speaking countries, it was insufficient for most other countries. For example, there are accented characters in various European languages that are not included in the IBM extended character set. Therefore, a number of other character sets were developed. Each character set, including the original IBM extended character set, is contained in a *code page*. Each code page is designed for a particular country or group of countries, and each is identified by a three-digit number.

Some examples of the code pages supported by MS-DOS are:

- 437 English and some Western European languages
- 850 Most Western European languages
- 852 Many Eastern European languages
- 860 Portuguese
- 863 Canadian French
- 865 Nordic languages

These are known as *OEM code pages* (for Original Equipment Manufacturer). The classic IBM extended character set is contained in OEM code page 437 and is the default code page for the United States. DOS considers code page 850 to be the default for most European countries. Code page 850 contains all the letters (but not all the symbols) of code pages 437, 860, 863, and 865; consequently, many of the box-drawing and line-drawing characters contained in these code pages are omitted to make room for accented characters in code page 850.

Each character in a code page is identified with a number; this number (which can be decimal or hexadecimal) is known as a *code point*. For example, the code point of the numeric character "4" is 52 (decimal) or 34 (hexadecimal) in code pages 437 and 850.

The Windows environment uses its own character set, which is generally known as the *ANSI character set*. Although this character set shares many characters in common with the OEM code pages, it omits most of the line-drawing characters and mathematical symbols that these code pages offer. Furthermore, even characters shared in common between an OEM code page and the ANSI character set often have different code point numbers.

The global language driver determines the character set used by *dBASE SE*. If you have another product already installed on your system that uses the Borland Database Engine (BDE), your current language driver is unchanged when you install *dBASE SE*. If no BDE language driver setting is detected, however, *dBASE SE* installs the ANSI language driver by default.

About language drivers

dBASE SE uses language drivers to specify which character set to use and which language rules apply to that character set. For example, the Canadian French language driver uses a character set that is identical to code page 863, while the default driver for the United States uses a character set that is identical to code page 437. It is important to understand that *dBASE SE* uses these internal code pages instead of the code pages supplied by the operating system.

dBASE SE language drivers contain tables that define or control the following for a particular character set:

- Alphabetic characters
- Rules for upper- and lowercase
- Collation (sort order) used in sorting or indexing
- String comparisons (=, <, >, <=, >=)
- Soundex values (values that represent phonetic matches when exact spellings are not known)
- Rules for translation between OEM and ANSI character sets

dBASE SE identifies each driver with a character string known as an *internal name*. For example, the internal name of the German driver for code page 850 is DB850DE0, and the internal name of the Finnish language driver is DB437FI0. The following table lists some of the European language drivers available in *dBASE SE*.

Table 17.1 European language drivers available in *dBASE SE*

Language or country	Code page	Internal name
Portuguese/Brazil	850	DB850PT0
Portuguese/Portugal	860	DB860PT0
Danish	865	DB865DA0
Finnish	437	DB437FI0
French/Canada	850	DB850CF0
French/Canada	863	DB863CF1
German	437	DB437DE0
Italian	437	DB437IT0
Netherlands	437	DB437NL0
Norway	865	DB865NO0
Spanish	437	DB437ES1
Spanish	ANSI	DBWINES0
Swedish	437	DB437SV0
English/UK	437	DB437UK0
English/UK	850	DB850UK0
English/USA	437	DB437US0
English/USA	ANSI	DBWINUS0
W. European	ANSI	DBWINWE0

When *dBASE SE* converts data from OEM to ANSI, and vice versa, most alphabetic characters exist in both an OEM code page and the ANSI character set and are converted without problem. Most of the extended graphic symbols in an OEM code page cannot be represented in the ANSI character set at all. When such a discrepancy exists, *dBASE SE*, like other standard Windows applications, makes a guess at the nearest character, but data loss can occur.

Performing exact and inexact matches

When *dBASE SE* compares two characters, either an exact match (also known as a *primary match*), or an inexact match (also known as a *secondary match*) can be performed. An exact match is performed when SET EXACT is set to ON, and in inexact match is performed if EXACT is set OFF.

An exact match requires that the characters be exactly the same. For example, although the characters O and Ö are similar, they don't satisfy the requirement for an exact match, and a SEEK or a FIND expression treats them as different characters. In contrast, an inexact match requires only that the characters belong in the same general category. For example, since the characters O and Ö are similar in several languages, they satisfy the requirement for an inexact match with many language drivers; a SEEK or a FIND expression treats them as identical characters.

Exact and inexact matches are performed using *primary weights* and *secondary weights*, which are assigned by a language driver to each character. Exact matches use primary and secondary weights, while inexact matches use only the primary weights and ignore the secondary weights. For example, the SET EXACT command controls whether characters with umlauts match their respective characters without umlauts. The following commands open a table named VOLK.DBF and search for a record with a key value that satisfies an exact match criterion:

```
set exact on
use VOLK order NAMEN
seek "KONIG"            // Will NOT find "KÖNIG".
```

EXACT is ON and the secondary weights of O and Ö are different, so they are evaluated as different characters. The following commands open VOLK.DBF and search for a record with a key value that satisfies an inexact match criterion:

```
set exact off
use VOLK order NAMEN
seek "KONIG"            // Can find "KÖNIG".
```

EXACT is OFF and the primary weights of O and ö are the same, so they are evaluated as identical characters.

Using global language drivers

Each time you start *dBASE SE*, a language driver is activated automatically. This is known as the *global language driver*. This setting applies to reading and writing of files, table creation, table-independent character operations and the output of commands and functions. For example, the global language driver governs FOR and WHILE expression evaluations.

dBASE SE normally chooses the global language driver from the dBASE Language Driver setting in the BDEADMIN.EXE Utility. Optionally, you can also specify a global driver in your dBASE_SE.INI file with an *ldriver* key. When there is no dBASE_SE.INI entry for a language driver, the setting in the BDEADMIN Utility determines the global language driver. When you place a valid driver entry in dBASE_SE.INI it overrides the setting in the BDEADMIN Utility except when creating tables. *dBASE SE* will always set the new table's language according to the global language driver specified in the BDE Administrator Utility.

To set the ldriver option in dBASE_SE.INI:

1. Close *dBASE SE* if it is running.
2. Open the dBASE_SE.INI file (normally located in your dBASE\BIN directory) and enter one of the following in the [CommandSettings] section:

   ldriver = WINDOWS

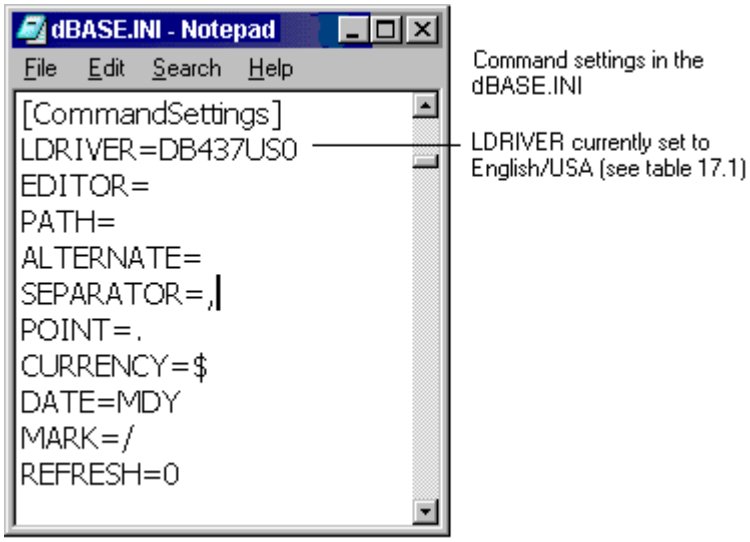
   or

   ldriver = <internal driver name>

   For example, the internal name of a European Spanish language driver for code page 437 is DB437ES1; to install this driver, insert the following setting:

   ldriver = DB437ES1

Figure 17.1 Setting LDRIVER in the dBASE_SE.INI



3 Save your changes and restart *dBASE SE*.

Use ldriver = WINDOWS to maximize compatibility with the operating system locale.

Use ldriver = <internal driver name> to specify a Borland language driver and maximize compatibility with legacy applications. For legacy applications matching the global language driver to the one previously in effect will help ensure compatible character handling and processing of data in the legacy tables.

Using table language drivers

dBASE SE assigns a language driver to a table automatically when you create it. This assignment is recorded in the LDID, a 1-byte identifier in the file header region. When you create a table from scratch, *dBASE SE* always assigns the current global language driver to the LDID. When you create a table file from another table file, either the global language driver or the language driver of the original table is assigned to the LDID of the new table. Which language driver is assigned depends on the command you use to create the file, as shown in the following table:

Table 17.2 Automatic assignment of language drivers by *dBASE SE*

Assigns global driver to the LDID of new table	Assigns original table driver to LDID of new table
CREATE	COPY FILE
CREATE...FROM	COPY STRUCTURE
CREATE...STRUCTURE EXTENDED	COPY...STRUCTURE EXTENDED
	COPY TABLE

For example, the following commands open a table file and create a new one with the LDID set to the current global language driver:

```
use CLIENTS     // LDID specifies a language driver other than global language driver
copy to CLIENTS2 structure extended  // LDID of CLIENTS2.DBF matches the language
                                     // driver of CLIENTS.DBF
use CLIENTS2 exclusive
create NEWCLIENT from CLIENTS2       // Create a new table with the global LDID
```

The following commands open a table file and create a sorted table file with an LDID set to the original table language driver:

```
use CLIENTS                 // LDID specifies a nonglobal language driver
sort on LASTNAME to CUSTOMER  // LDID of CUSTOMER the same driver as with CLIENTS
```

Identifying a table language driver and code page

Because some commands behave differently than others when a table language driver differs from the current global language driver, it is often necessary to detect which language driver is assigned to the LDID region of the table file or determine which code page the language driver uses. For example, file and field names may be valid in one language but not in another, or a key field may have characters that are not shared in common between the code pages of the language drivers.

When you use a command to open a table, and that table has a language driver that differs from the global language driver *dBASE SE* displays a warning dialog box only if LDCHECK is set to ON (installation default is OFF).

To determine which language driver is recorded in a table LDID region and which code page the driver uses, use the LDRIVER( ) and CHARSET( ) functions:

```
set ldcheck off      // Turns off automatic language driver compatibility checking.
use CLIENTS exclusive
index on COMPANY tag COMPANY
if ldriver( ) == "DB437DE0"   // If this is the German language driver...
    seek "Shönberg Systems"   // Searches are OK
else
    if charset( ) == "DOS:437" // If the driver uses Code Page 437...
        warn1.open( )          // Opens a custom warning dialog box.
    else                   // If the driver doesn't use Code Page 437...
        warn2.open( )         // Opens a different warning dialog box.
    endif
endif
```

Non-English Character Display Issues

There are two parts to this issue:

- First we must make sure *dBASE SE* uses the correct code page when interpreting text encoded in source files (.prg, .wfm, etc.), or text encoded in an incoming byte stream from a *dBASE SE* Web App.
- Second, we must make sure *dBASE SE* uses a display font that contains the characters we wish to display. For font specification, see Selecting Specialized Product Fonts.

dBASE SE interprets text according to the code page associated with the language driver you specify. You can set the language driver in the dBASE_SE.INI file through the CommandSettings section as follows:

```
[CommandSettings]
ldriver=<internal Name>
```

For a complete list of Language Drivers and their internal names, please see the topic "About language drivers". Select the language driver that best matches the language you wish to display.

In addition to using the ldriver setting, you can also use the BDE to designate a desired language driver. In the absence of an ldriver setting in the dBASE_SE.INI, dBASE uses the default language driver from the BDE configuration. This BDE setting can be used to designate alternative language drivers in much the same fashion as an ldriver setting. Please note that an ldriver setting takes precedence and will therefore override any subsequent change to the BDE configuration.

Selecting Specialized Product Fonts

In order to use TrueType fonts which do no use the Western Europe code page (1252), you must specify the language (also referred to as the "script"). Since *dBASE SE* does not list available language scripts for TrueType fonts, you must specify it in the *fontName* property--either in code or through The Inspector--using the exact TrueType font name. If you use The Inspector, choose a text component, it’s *fontName* property and, instead of choosing from the fonts list, type in the name of a desired language script. We recommend all languages be entered in English, e.g.:

- Times New Roman Greek
- Verdana Turkish
- Arial Baltic

- MS Gothic Cyrillic
- Courier New Central Europe

The following dBASE_SE.INI file settings ensure that the initial font created for a new control uses the language you want:

```
[DefaultFonts]
Application=<strFontName>,<intPointSize>
Controls=<strFontName>,<intPointSize>
```

The Application setting specifies the font used for the Navigator and Inspector, while the Controls setting specifies the default font used for forms and controls. You can also create your own custom controls to specify the font and language you want to use.

Table language drivers versus global language drivers

When the language driver of a table differs from the current global language driver, the table language driver is loaded into memory automatically when you open the table. Thereafter, the table language driver is respected by some commands, while the global language driver is respected by others.

All commands that have nothing to do with a table use the global language drivers. The following table shows the general rules when operations are performed on table data where the table language driver differs from the global language driver.

Table 17.3 Language drivers: Table versus Global

Table driver	Global driver
INDEX ON command expressions	SET FILTER command expression
FOR scope expression of INDEX ON command	FOR and WHILE expressions for every command except INDEX ON
SET KEY range checking expression	SET RELATION TO expression
SORT command expressions	
Secondary matches for expressions in LOOKUP( ), FIND, SEEK, and SEEK( ) with EXACT set OFF	
Secondary matches for SET RELATION TO expression with EXACT set OFF (uses the driver of the child table)	

For example, when you create a table file with the German language driver, an LDID identifier is written to the header region of the file. If the global language driver is set to English and you open the table in *dBASE SE*, *dBASE SE* notes the discrepancy between the table's and system's language rules. If you create an index with INDEX ON, the logical order of the index obeys the language driver of the table:

```
use VOLK                  // Created with the German language driver
index on NAMEN tag DIENAMEN  // Orders records in the German way
```

By contrast, if you create a filter with SET FILTER, the filtering condition obeys the global language driver:

```
use VOLK
set filter to NAMEN = "KONIG"  // Excludes records with "KÖNIG" in NAMEN
```

Handling character incompatibilities in field names

When you use a table file that was created with a different language driver from the current global language driver, some characters in the table file might not be recognized. This can lead to problems.

For example, the German language driver DB437DE0 has the same code page and character set as the US language driver DB437US0. However, the German language driver recognizes extended characters like ö and Ü as alphabetic characters, while the U.S. language driver doesn't. Consequently, when a field name contains such characters (as with ÜNAMEN in the example below) problems arise when you try to reference the field while using the U.S. language driver.

When such conditions exist, the following command generates an error condition:

```
replace ÜNAMEN with "Schiller"
```

You can solve this problem by surrounding the field name with the : delimiter, which treats the field name as an identifier regardless of the rules contained in the current language driver:

```
replace :ÜNAMEN: with "Schiller"
```

When you use this command, each element in the field name ÜNAMEN, including Ü, is treated as an identifier and the command executes successfully.

Converting between OEM and ANSI Text

The Source Editor Properties dialog box lets you specify whether to view the text, in an editor, in the DOS (also called OEM or ASCII) character set or the Windows (also called ANSI) character set. Only the way you view the source changes, not the actual code points.

You can also convert between character sets. For example, you may want to convert from the OEM character set to the ANSI character set if you are changing to an ANSI language driver, and your program uses extended characters.

Warning! Converting your program to a different character set changes the code points of values in your program. You may lose information if a character you convert does not exist in both character sets.

You should carefully review extended characters in your code if you convert between character sets.

Converting from OEM to ANSI

The encoding is changed when you convert between character sets. For example, if your program was written in the OEM character set and it uses the Ä character (OEM 142), the actual numeric value of the character is changed to 0196 when you convert the program to ANSI, because 0196 is the numeric value of Ä in the ANSI character set.

Alphanumeric characters usually do not cause problems when you convert from an OEM character set to the ANSI character set. Characters that typically do not convert include:

- Greek characters other than ß (ANSI 0223) and µ (ANSI 0181)
- Line-drawing and box characters

Converting from ANSI to OEM

Lowercase alphanumeric characters in the ANSI character set exist in all OEM character sets and are converted without problems. Uppercase extended alphabetic characters exist in some OEM character sets (such as OEM code page 850) but not in others (such as OEM code page 437). Following are the rules *dBASE SE* uses for conversion:

- Extended characters in the ANSI set that do not exist in the OEM character set are converted to "similar" characters. For example, the accented À (ANSI 0192) is converted to the capital A in OEM code page 437.

How to convert and view your source code

To convert between character sets:

1. Open your program or text file in the editor.
2. Select the section you want to convert or Select | All.
3. Choose Edit | Convert | To DOS Text to convert your source to OEM, or Edit | Convert | To Windows Text to convert your source to ANSI.

Note Converting between character sets does not change the character set you are using to view your text or program.

To change the way you view your text or program:

1. Choose Properties | Source Editor Properties.

2 Specify either DOS Text or Windows Text for Interpret text as.

Chapter

18

Converting prior version dBASE Applications to *dBASE SE*

How you go about migrating your dBASE applications to *dBASE SE* will depend on which prior version you are currently using.

dBASE III+/IV

To convert your application to Visual dBASE 5.7, see the section titled, "Converting a dBASE III+/IV Application to Visual dBASE 5.7.", on this page.

Note For dBASE III+ users: This procedure will only work for .PRG and .FMT files.

Once this has been completed, you need to convert the Visual dBASE 5.7 application to *dBASE SE*. See the section titled, “Converting Visual dBASE 5.7 Applications to dBASE SE” on page 18-15.

dBASE 5.0 for DOS

Convert your reports and labels in the same fashion as a dBASE IV application. See the section titled, "Converting a dBASE IV Application to Visual dBASE 5.7", on this page.

Object-based forms and menus, created using the DEFINE syntax, can be converted directly to *dBASE SE*. See the section titled, “Converting dBASE 5.0 for DOS Screens/Menus to dBASE SE” on page 18-12.

dBASE 5.0 for Windows through Visual dBASE 5.7

See the section titled,“Converting Visual dBASE 5.7 Applications to dBASE SE” on page 18-15.

Converting a dBASE III+/IV Application to Visual dBASE 5.7

Note For dBASE III+, this procedure will only work for .PRG and .FMT files.

Installing Visual dBASE 5.7

The conversion process requires you to have Visual dBASE 5.7. If you do not:

Insert the dBASE SE CD. If a menu displays on your screen, click the "Exit" button. Using Windows Explorer, navigate to the CD drive and double-click on the folder named "16bit".

Choose an install language from the five listed in the "Visual dBASE 5.7" folder:

- DE = German
- EN = English

- ES = Spanish
- FR = French
- IT = Italian

Double click the desired language folder. Double click the "vdb" folder. Open the "Disk 1" folder and double click the “Setup.exe” file to begin the installation process. If you are unfamiliar with Visual dBASE 5.7 you should use the "default" install, which installs the program according to a set of pre-determined defaults.

Overview

The following is designed to help you convert an existing application to Visual dBASE 5.7, and is not a programming guide for the language. Most of what you read here concerns how to convert your screens, reports and labels to Visual dBASE 5.7. If you have other questions, please check the online help for Visual dBASE 5.7, as well as the Knowledgebase for *dBASE SE* (which includes a Visual dBASE 5.7 section).

Suggested Steps

An application, the "Component Builder", is provided with Visual dBASE 5.7 which can be used to convert some of your code directly from DOS code to Windows code. Create a folder to use for the new code as well as copies of your tables. Make sure you copy all .DBF, .MDX, and .DBT files.

Start Visual dBASE 5.7. Click the “Current Directory” folder in the Navigator window and select the folder you will use to store the Windows version of your application.

Most applications are quite diverse, however the following steps assume a relatively basic application. When converting your DOS application to Windows you need to do the following:

1. Create or Convert your Menus
2. Convert your Screens to Visual dBASE Forms
3. Fine tune the forms
4. Convert your Report and/or Label files to Crystal Reports
5. Fine tune the reports and/or labels
6. Create any queries necessary for your reports or labels
7. Create any code needed to communicate with your forms, reports, and labels
8. Modify the menu so it calls the appropriate programs.

Note It may be possible to run your code in Visual dBASE 5.7 without doing a conversion, but doing so will cause it to look like a DOS Screen, not a Windows application.

Sample

There will be a few screen shots for a very simple address-book style application, but for the most part there are other chapters of the User's Guide that detail how to use the different design surfaces.

The sample is very simple and is only used to get you started.

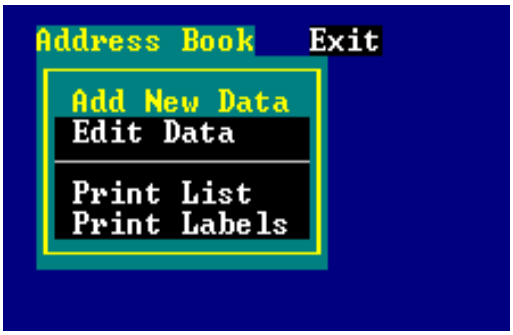
Recreating Menus

The "Component Builder" that ships with Visual dBASE 5.7 can be used to convert screens and reports, however you will need to manually reproduce your menus.

Create a sample menu in Visual dBASE 5.7

Click the “Forms” icon in the Navigator. Four untitled icons will appear in the window to the right. “Right-click” the icons to locate the one called “New menu”. Double-click (left mouse button) the "New Menu" icon to open the menu designer.

Figure 18.1 Sample Menu



Note The menu designer in *dBASE SE* has not changed much from the one used in Visual dBASE 5.7. For additional information, see Chapter 8, “Creating Menus and Toolbars”.

The menu designer will have two windows open, "Untitled - Menu Designer", which is the design surface, and "Untitled - Inspector". If the Inspector does not appear, select it from the View menu. The Inspector is used to set or change properties, and other aspects, of the menu being designed. You should be able to reproduce the menu shown in figure 1 with little effort. For the time being, we're only going to visually design the menu. A little latter, we’ll come back to tell the menu what to do when the user selects the items.

Click in the design surface window and an EntryBox with a pulsating cursor will appear.

- Type the word “File”, for the first menu option, and press the down arrow key to get a “Sub menu” entrybox. Sub menus appear beneath the main menu heading and offer it's associated options.
- Type "Add New Data” and press the down arrow key.
- Type "Edit Data" and press the down arrow key again.

For the final sub-menu option add an "Exit" option preceded by a separator. In the Inspector, click the *separator* property and use the drop-down list to change it's value from F to T. Click in the design surface. Press the down arrow key, and type "Exit". The "File" menu option is now complete.

To add a second main menu option, move the cursor back up to "File" and press the tab key. Type the word "Reports" and press the down arrow key. Continue in this fashion, adding menu options for “Print List” and “Print Labels”.

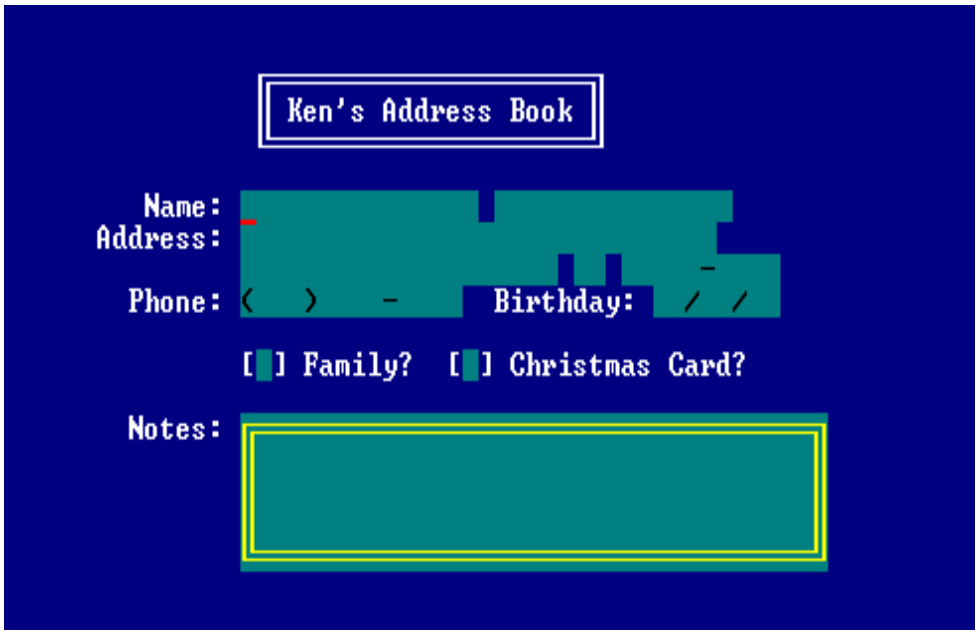
Save the menu and exit the designer. The easiest way to save the menu is by pressing Ctrl+W (Save and Exit) simultaneously. You could also use Ctrl+S to save the menu, and then close the window with the 'X' in the TitleBar. When asked for a filename, call this "Address". If you look in the Navigator you should now see "Address.mnu" listed.

Converting .VUE Files

There is no direct conversion to the Visual dBASE .QBE file format for .VUE files. To convert a Screen or Report/Label file, select a .VUE file and the Component Builder will attempt to convert it. Once it’s converted, open the .QBE in the Visual dBASE 5.7 Query designer and modify it as needed.

Converting Forms

Figure 18.2 Sample Form



To convert a screen to a Windows form, use the Component Builder installed with Visual dBASE 5.7. To run the Component Builder, you will need to change the current folder to point to the folder containing the Component Builder.

To do this:

- Click the “folder” pushbutton in the Navigator Window's "Current Directory" area.
- Select "Visualdb\utils\", the utils folder, and click "OK" (make sure you use the actual path for your installation of Visual dBASE 5.7).
- In the Command Window, type:

  ```
  do cb
  ```

  and press the Enter key. This will start the Component Builder program.

In the Command Window, type:

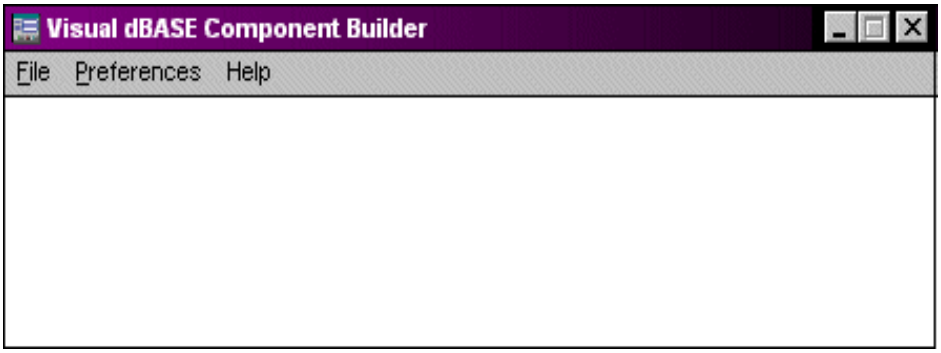
```
do cb
```

and press the Enter key. This will start the Component Builder program.

Note When the Component Builder window appears, you’ll see it’s "Help" menu. If you need assistance, this is a good place to go.

The Component Builder screen looks like that shown in Figure 18.3.

Figure 18.3 The Component Builder



1. Select the "File" menu
2. Select "FMT to WFM ..."
3. Select the folder that has your DOS application and the .FMT file to be converted
4. Click "OK"
5. Select a "Database" -- select the table or view (.VUE or .QBE) you need for the screen
6. Click "OK"
7. You will be asked where to save the new form. Select the folder where you wish to put your Windows application. Click "OK"

A screen will appear showing the progress of the conversion. This will show the different types of objects, the lines, etc. Unless there is a serious error, the Component Builder will attempt to convert everything on this screen to a Windows object. If you have complex code, some of it may converted in which case you will need to make modifications.

When the screen is done being converted, click "OK" on the screen showing the progress bar, and exit the Component Builder.

Use the Navigator to go back to the folder with your Windows application. If you click on "Forms" in the Navigator you should see "Address.wfm" in the list on the right.

You will most likely need to fine-tune the job done by the Component Builder. There are some very good instructions on using the Form Designer in Chapter 7, “Using the Form and Report designers. While the User’s Guide is aimed at *dBASE SE*, most of what is said can be applied to the Visual dBASE 5.7 form designer.

Using The Component Builder to Convert a Form from a .PRG

The Component Builder menu contains an option to convert a .PRG to a:

- .WFM (Windows Form)

  or

- .MNU (Menu) file.

We’ve had little success with this option, however it’s included for those of you who would like to try. The process is similar to using the Component Builder to convert a .FMT to a .WFM file, with the exception that you need to highlight the source code you want to convert.

The difficulty with using the Component Builder is that it does not understand a lot of the setup, contained in the program file, that most dBASE/DOS programs use to display screens. All it is looking for are the @/SAY, @/SAY/GET, and READ statements in the program.

If you choose this option:

1. Use the Navigator to go to the Visualdb\UTILS folder
2. Run the program CB.PRG
3. Select the folder with the source code

4 Select the program file you wish to convert, and click OK.

5 A new window is displayed (See Figure 18.4)

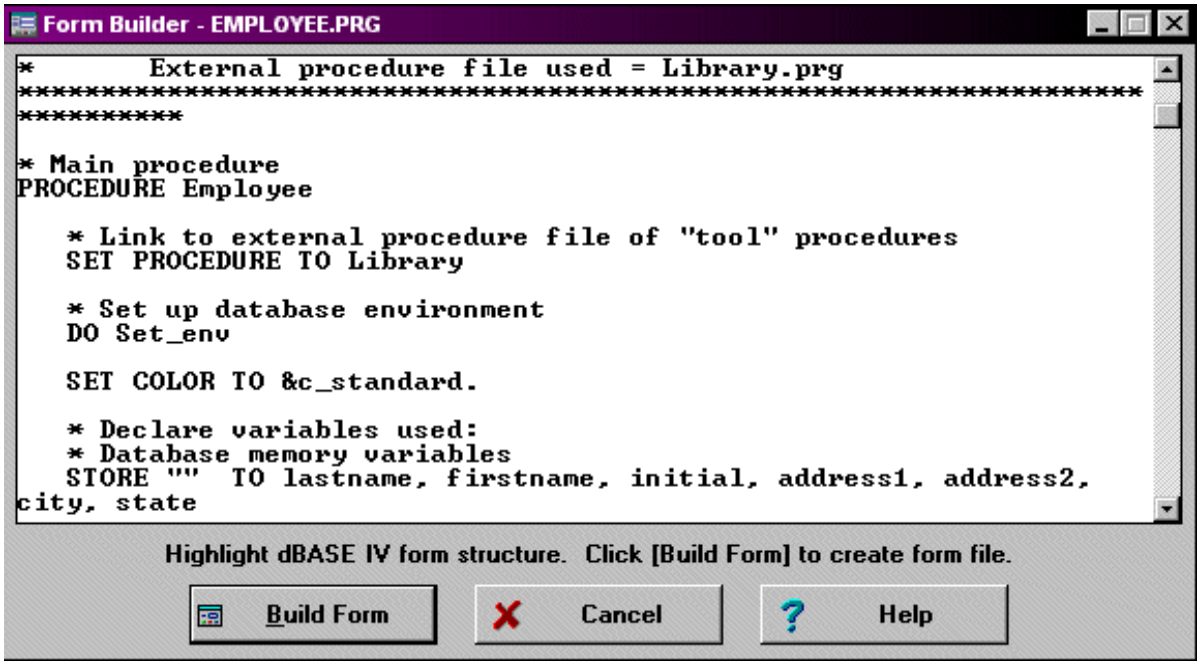
6 Highlight the parts of the code you wish to convert

7 Click the "Build Form" button

8 At this point we get an error message stating: "Marked text block required". If no error message displays, the Component Builder will continue to run.

If you receive an error message, our only suggestion would be to copy the code to another file, rename it as an .FMT file, and retry.

Figure 18.4 Component Builder window with displayed source code



Fine-Tuning The Form

Once you have a converted .WFM it will probably not display exactly as the DOS program. This is simply due to differences between Windows and DOS. To correct this, you will need to open your new .WFM in the form designer and manipulate it.

To do this:

1 Start Visual dBASE 5.7 if it is not already open.

2 Use the Navigator to navigate to the folder for your Windows application.

3 Click on the "Forms" icon on the left of the Navigator.

4 The new .WFM file will appear on the list in the Navigator window.

5 Highlight it (give it “focus”) by single clicking, then right click, selecting "Design Form" from the context menu.

Note To understand all the different palettes in the design surface, you should read Chapter 7 of the User's Guide on using the Form Designer. While the form designer in Visual dBASE 5.7 is different from the one in *dBASE SE*, they are similar enough for you to use.

With the form in the designer, you can not resize it and adjust the controls to correct the form’s display. "Lasso" the controls, and move, them by holding down the Ctrl key, clicking the mouse, and dragging until all the

controls you wish to affect are selected. Click on a single control and drag all of the “lassoed” controls to a desired location. Resize the form by clicking on the border (the edge), and dragging it in and up, etc.

You can now use the Inspector to change colors, fonts, font sizes, and other properties of the controls.

Notes about Memo and Logical fields

Memo fields: A memo field, by default, is placed on a form as an Entryfield displaying the word "Memo". The control used to handle memo fields is called an Editor.

To change the Entryfield to an Editor control:

1. Delete the entryfield that says "Memo"
2. Click the Control Palette window
3. Move the mouse over the controls until you find one that says "Editor" in the small "speedtip" window.
4. Click and drag the Editor control to your form. Move and size it appropriately.
5. Click the Inspector window, and select the "DataLink" property.
6. Click the tool button (the "wrench")
7. Click the tool button in the new window
8. Select the field and click "OK".
9. Click "OK" again

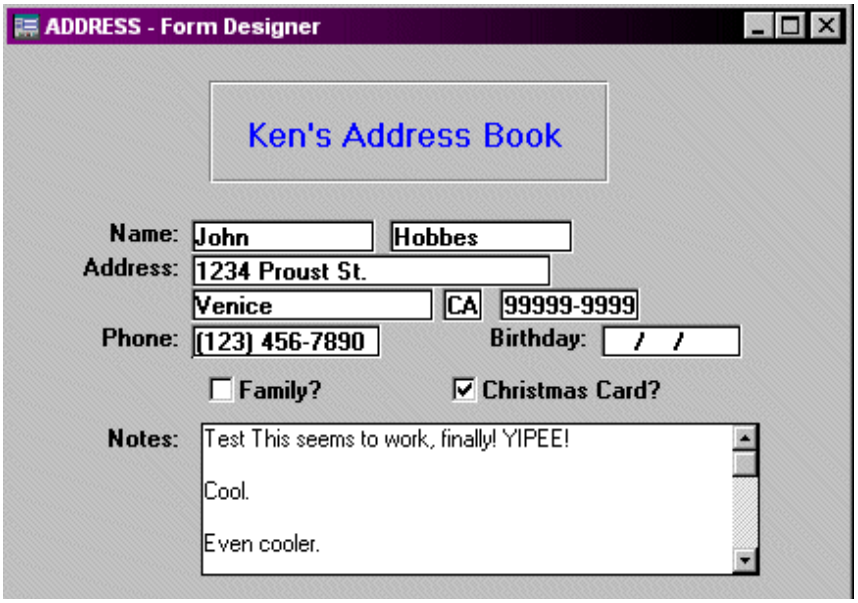
Logical fields: In Windows applications, logical fields are normally displayed as Checkboxes, checked meaning "true" and unchecked meaning "false". The Component Builder will place Checkboxes onto a form, however, as the dBASE/DOS software did not have Checkboxes, they are not really created quite right. You can delete these controls and use the Inspector to set the *text* property of the Checkbox "object" instead. Since the Checkbox object defaults to a short width, and at first the text doesn't show up, widen it using the Inspector. It will look better and allow you to manipulate a single control rather than several.

Click directly on the form's surface, to give it focus, and click the Inspector. Find the *mdi* property (it's under the "Windows Properties" heading), and change it from T to F. "MDI" stands for "Multiple Document Interface". DOS applications were generally not "MDI", and while it was possible to write MDI applications in DOS, doing so was quite complex.

If you set the form's *autoCenter* property to .T., the form will be centered on the screen. If not, the form’s display is determined by it’s *top* and *left* properties.

With a little work, your form could resemble Figure 18.5:

Figure 18.5 A Running Form



We could modify this form all day, but we're only going to add one more item - a button that allows the user to close the form.

Visual dBASE 5.7 comes with a selection of custom controls. These are simply common controls the developers designed to save us the trouble. Click on the Control Palette and click the tab marked "Custom" at the bottom of the page.

You can view the name of any one of the resulting series of pushbuttons by moving the cursor over it. Find the one named "Closebutton", and drag it to the bottom of your form. This button already has code defined that will close the form when clicked.

Running the Form

You should run the form to see how it looks, and acts. To do this, save it and click the "Run" button at the top of the screen 

You can navigate through records in your table using the arrow buttons at the top of the screen (the toolbar), and you can close the form using the close button at the bottom of the form.

To make further modifications to the form, check the Knowledgebase installed with *dBASE SE*. Among the vast amount of information available, there is a section for Visual dBASE 5.x including HOW TO documents for working with Object-Oriented Programming.

Repeat the process of converting your forms for each screen you use.

Using ACCEPT or INPUT?

Some items won't convert directly using the Component Builder. For example, the commands "ACCEPT" or "INPUT" will not work in *dBASE SE*. If you were to leave them in your programs, they would not look, or act, like a Windows application.

Suppose you wanted to use code to find a person in an address book, and view or edit their data. You would use code similar to:

```
public cSearch && make it public so that the search form can find
    && it in memory
use address order name

do while .t.   && loop until told to exit
    cSearch = space(15)   && initialize it
    accept "Last Name to find: " to cSearch
```

```
    *-- when the form is closed, the memvar will either
    *-- have a name, or it will be empty ...
    if isblank(cSearch)
        exit
    endif

    *-- if here, we have a value, let's try to find it!
    lOK = .f.  && initialize flag
    if seek(upper(trim(cSearch)))   && if we found a match
        do while trim(upper(cSearch)) $ trim(upper(last))
            accept trim(first)+" "+trim(last)+;
                " -- is this the name [Y/N]? " to cYN
            if upper(left(cYN,1)) = "Y" && found a match
                lOK = .t.
                exit
             else
                lOK = .f.
                skip
            endif
        enddo
    else                    && match not found
        lOK = .f.
    endif

    if .not. lOK   && we didn't find it, let user know
        ?
        ? "*** No match found ***"
        ?
    else          && we DID find it, display the form
        do address.fmt
    endif

enddo
  use
```

This code would need to be modified so:

- the ACCEPT command was not used
- the code displays the address form, not the old DOS .FMT file.

To start with, you want a form that prompts the user for the name of the person in the address book.

1 Click on the "Forms" icon in the Navigator, and double-click on the first "Untitled" icon on the right side. This will bring up a new form in the designer.

2 Drag a text control from the Control Palette to the design surface, and place it toward the top of the form.

3 Widen it by dragging the right edge to the right.

4 Click the Inspector, and in the *text* property field, type: "Last Name to find:".

5 Adjust the size of the text by changing the *fontSize* property to a larger value, such as 10 or 12 (you may need to readjust the Text object's *height* and *width* properties as well).

6 Drag an Entryfield object next to the text object.

7 Widen it, and set the *fontSize* property to the same size used for the Text object.

8 Adjust the *height* and *width* as needed.

9 Move it so it aligns the way you want.

10 Set the *dataLink* property to "cSearch" (type that in the Inspector without the quotes).

11 Click on the Form.

12 Bring up the Inspector and change the form's *mdi* property to F.

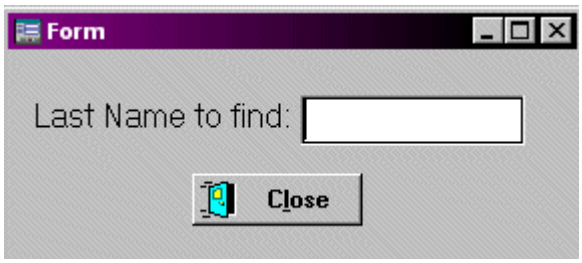
13 Set the form's *autoCenter* property to T to center the Form

14 Press Ctrl+S to save. Name the file "search".

You should now have a pushbutton the user will click to perform the search. Click on the "Component Palette", select the "Custom Tab", drag the "Closebutton" to the form, and center it under the controls. Resize the form, and it's done.

You should have a form resembling Figure 18.6:

Figure 18.6 Form with PushButton



- Modify the above code by replacing the "ACCEPT" command line with:

  ```
  do search.wfm with .t.
  ```

  This line will run the form, and the "with .t." ensures it runs as a modal form (details on "modal" versus "non-Modal" can also be found in the Knowledgebase).

- The section,

  ```
  accept trim(first)+" "+trim(last)+;
      " -- is this the name [Y/N]? " to cYN
  if upper(left(cYN,1)) = "Y" && found a match
  ```

  can be changed to use a built-in Visual dBASE dialog box that prompts for yes/no answers. This is called a Message Box, and it's called using the msgbox( ) function.

  Change both of these lines to:

  ```
  if msgbox( trim(first)+" "+trim(last)+;
      " -- is this the name?", "Confirm", 4 ) = 6
  ```

  If a match is found, this will prompt the user to confirm that the find was correct using "Yes" or the "No" buttons. (More details on the msgbox( ) function can be found in Help.)

- To replace the code displaying the message that no match was found:

  ```
  ?
  ? "*** No match found ***"
  ?
  ```

  call the msgbox( ) function once again:

  ```
  msgbox("We did not find it.","No Match Found")
  ```

  This will display only the message, and an "OK" button.

- Replace the code that calls the screen (.FMT file):

  ```
  do address.fmt
  ```

  with

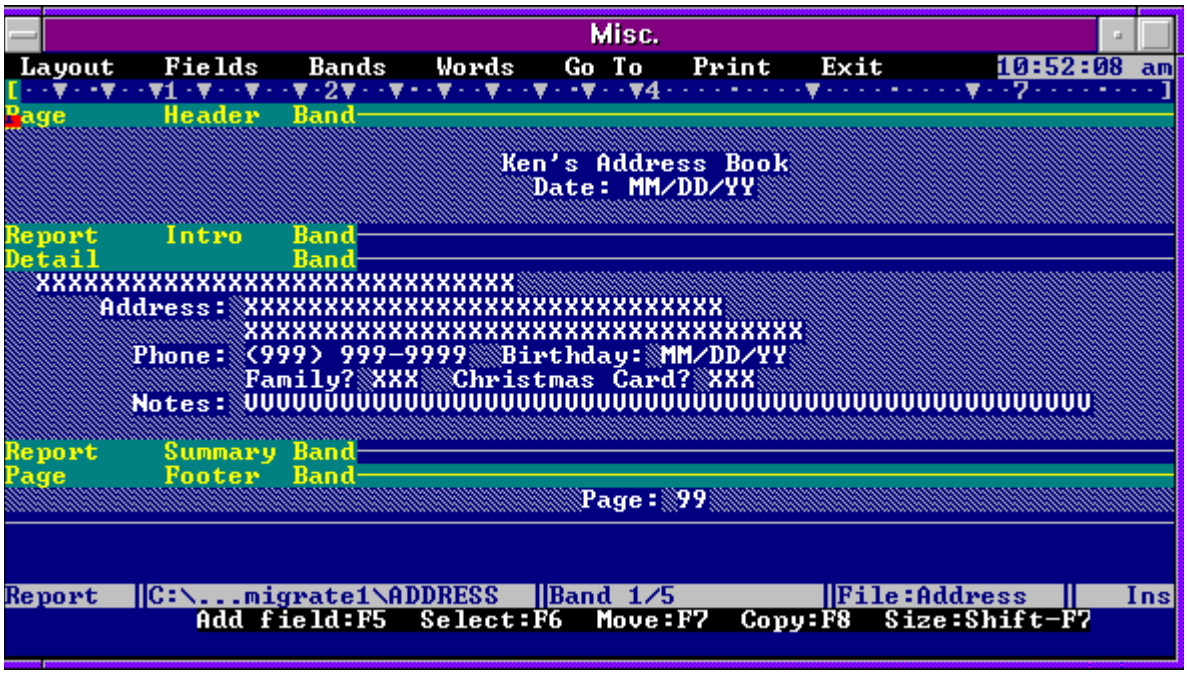
  ```
  do address.wfm with .t.
  ```

Reports and Labels

Visual dBASE 5.7 uses a special version of Crystal Reports 3.0., which does not understand the dBASE IV, or dBASE 5.0 for DOS, report and label files. However, the Component Builder installed with Visual dBASE 5.7 can convert these files to Crystal Reports.

Use the following steps to convert the report, shown below in the dBASE IV for DOS report designer, to a Crystal Reports version.

Figure 18.7 The dBASE IV for DOS Report designer



1. Using the Visual dBASE 5.7 Navigator, change your current directory to the UTILs folder containing the Component Builder.
2. Run the program CB.PRG
3. When the Component Builder screen comes up, select the File menu, and "FRM to RPT ...". The .RPT extension is the Crystal Reports report filename extension.
4. Select the folder, and then the report you wish to convert
5. Select the table needed for the report. If you are using multiple tables in a .VUE, select that file and the designer will attempt to create a .QBE file.
6. Select the folder in which to place the report.
7. A screen much like Figure 8.4 is displayed.
8. When done, click "OK". You can convert another, or close the Component Builder.

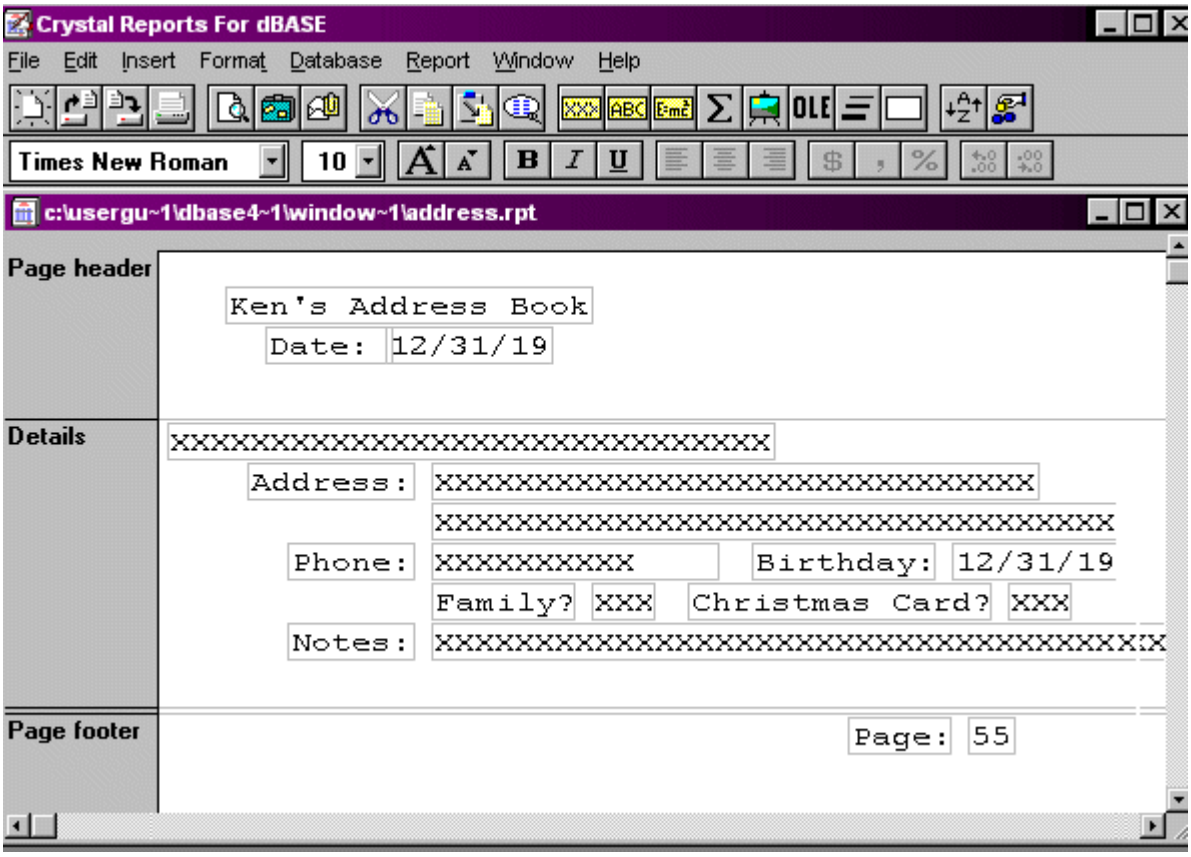
Open the report in Crystal Reports. You may need to modify the report somewhat (change the fonts, etc.).

To do this:

1. After closing the Component BuilderChange, use the Navigator to change the folder you are working in, to the folder that will contain the Windows version of your application.
2. In the Navigator, select the "Reports" icon.
3. Click on your new report
4. Right click and select the option, "Design Report".

You should see something similar to Figure 18.8:

Figure 18.8 Crystal Reports for dBASE



You can change the font for all items, or change each one individually. To change the font for all of items, use the Ctrl key along with the mouse. While holding down the Ctrl key, click each text control you want to change. Then Right click, and select "Change Font ...". You can select a font, a font size, and attributes.

Once you've changed the font, especially if you use a Windows "True-Type" style font, you may need to change the widths of some controls, and possibly re-locate a few.

With two execeptions, you can use the same steps for Label files as were used for Reports

- When you select the "File" menu in the Component Builder, select "LBL to RPL ..."
- Use the "Labels" icon when you return to Visual dBASE to modify the label file.

Converting dBASE 5.0 for DOS Screens/Menus to *dBASE SE*

dBASE 5.0 for DOS has a different form designer which uses objects not available in dBASE IV for DOS. Upgrading dBASE 5.0 for DOS applications to *dBASE SE* therefore requires a slightly different approach than those from dBASE IV for DOS.

Setting up for Conversion

Before starting the conversion:

- Create a folder containing your dBASE 5.0 for DOS application, including tables, forms, menus, reports, labels, programs, and other files you may have generated.
- Create a folder to be used for the converted Windows application. If you want duplicate copies or your tables, copy them, as well as the .DBF, .MDX, and .DBT files, to the Windows folder.

Converting screens or menus to *dBASE SE*:

1. Start *dBASE SE*
2. If the Command Window is not already visible, select it from the View menu.
3. Click in the upper pane of the Command Window and type the following:

   ```
   do :DOS5Conv:Dos5Conv.wfm
   ```

   This runs the Converter Wizard, which will guide you through the conversion process.
4. Read the instructions on the first screen and click the "Forward" button.
5. During this step, you'll identify the folder containing the files to be converted, and the folder in which to place the converted files. Click on the "Tool" button to the right of the entry areas to select the folders. Once these folders have been selected, click the "Forward" button.
6. Click the "Convert" button. The Wizard lists each source file as it's converted and, when finished, displays the list of newly created files.
7. Click the "Close" button and use the Navigator in *dBASE SE* to locate the folder, specified earlier, containing the converted files.
8. In the folder you'll see the files with the extensions .wfm and .mnu. These are the extensions for *dBASE SE* form and menu files. To open them in the Designer, right-click and choose the "Design (form or menu)" option. Forms can be run from the Navigator by double-clicking on a selected form.

Conversion considerations

A few things to note when the converter has finished:

- This converter was designed to create dBASE +/SE compatible objects from your screen and menu layouts. In doing so, our priority was to insure the layouts converted correctly. Still, your form or menu's design will most likely require some modification (changing the widths of some objects, possibly moving some objects a small amount).
- Some of your source code may not have been converted correctly. Lacking a huge language translation utility, it simply isn't possible to insure the correct conversion of all source code. Use the Source Editor to correct any conversion errors.
- The converter creates forms that use the OODML (Object Oriented Database Manipulation Language) of *dBASE SE*, which is not necessarily going to be familiar to you. When tables are opened, they are opened using Query objects, rather than the USE command, and datalinks to fields in the tables refer to the appropriate Field objects. To become acquainted with OODML:
  - Read through other parts of this User's Guide
  - Check the dBASE Knowledgebase at www.dBase.com, or from the Help menu
  - Subscribe to, and participate in, a few of the many dBASE newsgroups available at news:// news.dBase.com. They're free!
- When the converter runs, it reads all the form and menu definitions in the .DFM, .MNU and .PRG files, and creates individual files for each one. This means you end up with more files than you started with. For example, if you have a .DFM with two form definitions, you'll end up with two .WFM (Windows Form) files. The resulting filenames may look a bit strange, but this is only because the converter combines the names of the object and source files: The ACCT_REC.DFM file from the dBASE 5.0 for DOS CUA Samples folder is converted to ACCT_REC_ACCT_REC.WFM.
- When the converter creates a menu file, it checks to see if there is a form's menuFile property set to point to it, and if so, attempts to set that property in the new .WFM file. Therefore, the menus should be correctly associated with the forms.
- Properties of a form that were defined programmatically, using expressions or variables, will be converted to their default values. For example, a .DFM may have defined the top property based on a value relative to another form. In this case, the top property is likely to be set to zero when the .WFM is created by the converter. This is necessary because there is no good way for the converter to know the relative value these values refer to.

- Procedures in your forms and menus may be converted to the .wfm or .mnu file with the code itself placed inside of a "comment block". This is a block of text that appears between the comment block symbols /* and */. For example:

  ```
  procedure PUSHBUTTON1_ONCLICK
  / *
      select acct_rec
  */
  ```

  The code is present and hooked up to the pushbutton, but will not execute until you have a chance to make sure it's correct. If the procedure involves data manipulation, it will most likely fail and you'll need to replace it with OODML syntax. If the code looks correct, you can remove the comment blocks, keeping in mind that tables are not opened with the old dBASE for DOS syntax. They are opened in the new dBASE +/SE (dBL) syntax, using Query objects. The converter wizard will attempt to put referenced procedures, from a procedure file, into the form or menu.
- If you have any "#INCLUDE" statements, make sure you copy the include files to the new source location.
- If your programs, or forms, require the use of .BIN or other external binary files, chances are they will not work in *dBASE SE*. You'll need to come up with a dBL way to perform their function, or find another solution (i.e., ActiveX, or other third-party tools).
- Some menu keystrokes do not work in Windows (Ctrl-Ins, for example). Opening these menus in the Designer may result in errors pointing to these statements. The simplest way to deal with this would be to comment that line of code out (or delete it) by placing a * or // in front of the line of code. Open the menu in the Designer, and in the error dialog, click "Fix" and place the * or // characters in front of the line of code.
- If your form produces the error message, "Unallowed phrase or keyword: .oBForm", click the "Fix" button and place either a * or // in front of the statement;

  ```
  RELEASE _CmdWindow.oBForm
  ```

  to comment this line out. Then Save and Exit (Ctrl+W) the editor.

Remember, the converter is here to help bring the screen and/or menu layouts into *dBASE SE*. Expect to do some modification to the results.

An Option

1. Copy just the screen and/or menu code from files that involve the layout. Do not copy any actual code that manipulates the data. Basically, copy the screen design code, the "constructor code", to a new .DFM, and save it in a new folder.
2. Run the converter to bring over the definitions of your screens and menus.
3. Run dQuery to create a datamodule.
4. Open the new *dBASE SE* form in the form designer.
5. Place the dataModule object on the form surface, and change all the datalinks to the dataModule (if you have multiple tables, make sure you get the right ones).
6. Delete the query objects that are on the form.

In the dataModule, you can place a lot of your validation, and other code, create links between tables, and so on. This places everything in the one, re-usable, file.

Converting dBASE 5.0 for DOS Reports and Labels

To convert Reports and Labels, you'll first need to convert them to Crystal Reports. Follow the steps in the dBASE III+/IV to Visual dBASE 5.7 section of the User's Guide. Once you have converted your reports and labels to Crystal Reports, follow the instructions in the ensuing section to convert them to *dBASE SE*.

Converting Visual dBASE 5.7 Applications to *dBASE SE*

Once you have your application converted to (or written in) Visual dBASE 5.7, there are three different things you need to do to convert it for *dBASE SE*

- Convert your Forms
- Convert your Crystal Reports format to the *dBASE SE* report format.
- Convert any QBE files to dataModules
- Update your Forms to use dataModules

There are tools designed to help you do each of these conversions.

Converting Forms

To convert forms from Visual dBASE 5.x (5.5, 5.6, 5.7) to *dBASE SE*, you need to use Visual dBASE 5.7. The converter program needs to be run in Visual dBASE 5.7 to perform the conversion.

In Visual dBASE 5.7, use the Navigator to point to this folder

C:\Program Files\dBASE\SE\Converters\FormConverter5x

Note This path may differ depending on where you installed *dBASE SE*, and what language you are using.

You will need a folder containing your Visual dBASE 5.7 source code, and one in which to store your *dBASE SE* source code.

To perform a form conversion:

1. In the Command Window, type:

   ```
   set procedure to Conv1632.cc additive
   ```

2. Using the Navigator, select the folder containing your Visual dBASE 5.7 source code.
3. Open the form to be converted in the form designer.
4. Click on the Control Palette
5. Click the 'Custom' tab of the Control Palette
6. There should be a control with the letter "A" for the icon. When you move the mouse over the control it will display a speedTip that says "Conv1632". Drag this to the form.
7. Click the "Forward" button
8. Click the "Wrench" button next to the Entryfield that says "Select 5.x input file:"
9. Select the same form you have in the designer
10. Click the "Forward" button
11. Click the "Convert" button
12. Click the "Close" button

Close the form in the designer, but do not save changes. If you select "Yes" to save the changes, the converter's custom control will be saved as part of the original form.

You now have a new form in the same folder as your 5.x source code, but the file has "_fcv" added to the filename. For example, if you converted "Customer.wfm", you would have a new file "Customer_fcv.wfm".

This form file should contain the *dBASE SE* version of your 5.x form. You should try testing the form in *dBASE SE* and, once you are satisfied with it, copy or move it to your *dBASE SE* applications source folder.

If you use Custom Forms, convert all of them before converting your other forms.

Important The form converter does NOT change the source code to *dBASE SE*'s Object-Oriented Database Manipulation Language (OODML). A converted form will continue to use XBase Database Manipulation Language (XDML). If you wish to update to OODML, read the Knowledgebase which has several articles designed to assist.

Converting Reports and Labels

dBASE SE does not have the ability to run your Crystal Reports report and label files. However, a conversion tool exists to convert these to the *dBASE SE* report format.

To convert a report or label file the *dBASE SE* format, you need to start *dBASE SE* and, using the Navigator's "Look In" option, change to this folder:

C:\Program Files\dBASE\SE\Converters

Note This path may differ depending on where you installed *dBASE SE*, and what language you are using.

When you are there:

1. Click on the "Programs" tab
2. Double-click on the file "Rpttorep.prg"
3. Select "Report" or Label" at the bottom of the wizard
4. Click "Next"
5. Use the "Wrench" button next to the entryfield with the text of "Source Crystal Report or Label File"
6. Select the path and report (or label) filename
7. Select the path and report (or label) filename you wish to convert to. Note that the conversion wizard assumes the same location even with the different file extension (.REP for report, and .LAB for label).
8. Click the "Next" button
9. Click "Finish" to actually perform the conversion.
10. A list will appear with any warnings that were generated.
11. Click the "Close" button.

You can now open the new report in *dBASE SE*. Navigate to the location of the .REP file, and run it, or open it in the designer. You can also open the file in the source editor to examine the code.

To learn how to use the report designer in *dBASE SE*, please see Chapter 12, "Designing Reports".

Converting QBE Files to Datamodules

Why should you want to convert your QBE files to Datamodules, if your forms work fine using the QBE files? The answer is, you may want to update your forms to *dBASE SE* OODML methods, or you may want to create reports or labels based on a *dBASE SE* query. The report engine does not understand XDML methods of opening and manipulating data, and therefore does not understand the QBE file format.

To convert a Visual dBASE 5.7 QBE file to a *dBASE SE* Datamodule, open *dBASE SE* and, using the Navigator, point to:

C:\Program Files\dBASE\SE\Converters

As noted elsewhere, this path may differ depending on where you installed *dBASE SE*, and what language you are using.

Click the "Programs" tab and you will see the program "Qbe2dmd.prg". Double-click this file to start the converter.

In the first dialog to appear, select the folder containing your QBE files and select the file you wish to convert. Click "OK".

A new dataModule (.DMD) of the same name will be placed in the folder with the QBE file. For example, if you converted the Visual dBASE 5.7 Sample qbe "Animals.qbe", a file named "Animals.dmd" would appear in the Visualdb\Samples folder. To keep your new *dBASE SE* code separate from your Visual dBASE code, you should copy or move this file to the new location.

Note If you move this file to a new location, you may need to open the .DMD file in the source editor and edit any path statements it contains. Using the Animals.dmd example mentioned above:

```
sql = 'select * from "C:\VISUALDB\SAMPLES\ANIMALS.DBF"'
```

Will need to be changed to:

```
sql = 'select * from "ANIMALS.DBF"'
```

OR you could put the actual current path in the statement.

To edit the dataModule further, click the "Datamodule" tab in the Navigator, and then double-click it. This starts the dQuery/Web dataModule designer.

Updating Forms to Use Datamodules

This can be a bit tricky:

1. Open the form in the form designer.
2. Drag the dataModule to the form design surface.
3. In the Inspector, make sure the form is selected.
4. Find the *view* property.
5. Delete what is there.
6. Click on the Command Window.
7. Type: CLOSE TABLES
8. Select the "Window" menu, and then the name of your form.

All the datalinked controls will be blank.

For each control that was datalinked to a field:

1. Click the control
2. Click the *dataLink* property in the Inspector,
3. Click on the Tool (Wrench) button in the Inspector
4. Select the field from the dataModule in the dialog that appears,

For Images, use the *dataSource* property instead of *dataLink*.

This may take some time if your form is complex. If you have multiple tables, be careful you get the correct field from the correct table. When using multiple tables, consider opening the form in the designer and, before performing any other steps:

1. Click on a control that is datalinked
2. Select the Inspector
3. Click on the *dataLink* property
4. Write down the datalink for that control. Repeat these steps for all datalinked controls.

You can refer to your list when reassigning datalinks.

Index

Symbols

.MNU file 8-1
.POP file 8-1
.PRG files 9-1
  editing 9-1
.WFM file 5-7

Numerics

437 (code page) 17-8

A

abandon-data button 7-7
abandoning changes (Table designer) 14-5
accessing
  data (table of components) 7-6
  databases 6-1, 6-4, 6-7
  databases (automatically) 6-6
  databases (manually) 6-6
  databases (overview) 6-5
  tables 6-1
  tables (master-detail relationships) 6-8, 6-9
ActiveX controls
  setting up 7-8
adding
  columns (fields) to reports 12-6
  components to reports 12-9
  dollar sign (reports) 12-7
  fields 14-4
  fields (to a query) 11-4
  rows 15-7
  selection criteria (SQL designer) 11-5
  tables (in SQL designer) 11-3
  tables (SQL designer) 11-3
aggregate calculations (reports) 12-10
aliases
  creating 2-3
  using 3-4
AND 11-7
ANSI
  about character sets 17-2
  character incompatibilities in field names 17-7
  converting between OEM and ANSI text 17-8
  Identifying table language driver and code page 17-6
  Table language drivers vs global language drivers 17-7
  using global language drivers 17-4
Append mode 6-2
append-data button 7-7
applications
  choosing a user interface 4-1
  creating (basic steps) 5-1
  creating (introduction) 5-1
  debugging 10-1, 10-2, 10-5
  event driven 4-1, 4-2, 4-4
  how event driven programs work 4-2
ASCII 17-2, 17-8
AutoIncrement 13-5
average 12-10
  finding in reports 12-10

B

background
  color in reports 12-10
  image in reports 12-10
background (setting scheme) 7-16
BDE
  configuring 2-3
  default session 6-7
BDE Administrator 2-3
binary fields 15-11
bookmark( ) method 6-2
breakpoints
  defined 10-6
  using 10-7
Browse mode 6-2
bugs 10-1
  Debugger 10-1
  types 10-1
Building 8-4
button components 7-7

C

calculations (reports) 12-10
calculations on rows 15-9
call stack 10-8
changes
  Visual dBASE 5.x through Visual dBASE 7.0 1-2
changing
  table structure 14-6
Character field type 13-5
character incompatibilities 17-7
character sets
  about 17-2
  and exact matches 17-3
  code page 17-6
  converting between OEM and ANSI text 17-8
  global language drivers 17-4
  incompatibilities in field names 17-7
  table language driver 17-6
  table language drivers vs global language drivers 17-7
child tables
  and referential integrity 14-16
  defining referential integrity 14-16
class definition 5-8
code block builder
  defined 9-3
  from the Inspector's Events page 7-11
code blocks
  defined 9-3
  editing 9-3
  selecting (results pane) 9-5
  The Command Window 9-5
code editor 9-1
code pages
  and global language drivers 17-4
  character incompatibilities in field names 17-7
  examples 17-2
  identifying 17-6
  OEM and ANSI 17-8
  table language drivers vs global language drivers 17-7
code point 17-2
color
  in reports 12-10
colors (setting scheme) 7-16
columns
  moving and resizing (Table designer) 14-4
combining group criteria 11-9
Command window
  clearing the panes 9-4
  copying to 9-5
  defined 9-4
  editing 9-5
  loading functions 9-4
  opening 9-4
commands
  copying 9-4
  editing in Command window 9-5
  executing 9-5
  executing blocks 9-5
  history 9-4
  issuing 9-4
  multiline 9-5
  printing 9-4
  reusing 9-5
  saving to programs 9-5
  The Command Window 9-4
Component palette
  adding components 5-14
  creating custom components 5-13
  defined 7-3
  removing components 5-14
components
  aligning 7-14
  container components 7-12
  copying 7-14
  custom
    adding to palette 5-14
    creating 5-13
    removing from palette 5-14
  cutting 7-14
  data access 7-6
  deleting 7-14
  field 7-8
  layout 7-14, 7-16
  moving 7-13
  pasting 7-14
  placing 7-12
  placing in containers 7-12
  reports 12-9
  reports (table) 7-7
  resizing 7-15
  selecting 7-13
  selecting multiple 7-13
  spacing 7-16
  table access 7-6
  The Component palette 7-3
  user interface standard components (table) 7-4
configuring BDE 2-3
connecting to databases 2-3

container components 7-12
control types (field components) 7-8
controls 7-4
  user interface standard controls (table) 7-4
converting
  Converting a dBASE IV Application to Visual dBASE 5.7. 18-1
  Converting dBASE 5.0 for DOS Screens/Menus to dBASE SE 18-12
  Converting Visual dBASE 5.7 Applications to dBASE SE 18-15
  OEM and ANSI text 17-8
Converting to dBASE SE 18-1
count
  finding in reports 12-10
counting rows 15-9
Create Alias dialog box 3-6
Create Folder dialog box 3-6
Create Table dialog box 3-8
creating
  applications (basic steps) 5-1
  custom classes 5-11
  custom components 5-13
  dataModules 6-10
  drill-down reports 12-8
  forms 5-6
  forms (in Designer) 5-6
  forms (with Wizard) 5-6
  indexes 14-10, 14-12, 14-14, 14-16
  joins (queries) 11-12
  menus 8-4, 8-5
  methods 9-3
  printed labels 12-11
  projects 5-2
  Query objects 3-7
  report borders 12-9
  reports 12-5
  reports (in Designer) 12-5
  reports (with wizard) 12-1
  tables (in designer) 14-2
  tables (in dQuery/Web) 3-7
  tables (with wizard) 14-2
  using the Form designer 5-6
Criteria Page (SQL designer) 11-4
cursor position
  using to stop debugger 10-7
custom class designers 7-1
custom classes
  creating 5-11
  using 5-12
custom components
  adding to palette 5-14
  creating 5-13
  removing from palette 5-14
custom components (Component palette) 7-8
Custom Views 3-16
Customer support options 1-6

D

data
  accessing (table of components) 7-6
  displaying specific fields 3-16
  importing 15-7
data entry
  automatic 14-8
  constraints 14-8
  indexing for 14-10
  Live Data Area 3-12
  rules for updating or deleting 14-16
  validity checks 14-8
data model 6-1
  Database objects 6-4
  DataModRef objects 6-5
  Field objects 6-3
  Query objects 6-1
  Rowset objects 6-2
  Session objects 6-4
  StoredProc objects 6-5
Database objects 6-4
  aliases 3-4
  creating 3-4
  database-level security 6-4
  databaseName property 6-4
  default 6-4
  loginString property 6-4
database-level methods 6-4
databases 6-7
  accessing 6-1, 6-7
  accessing (overview) 6-5
  accessing (table of components) 7-6
  accessing automatically 6-6
  accessing manually 6-6
  connecting to 2-3
dataLink property 6-4
  and Field palette 7-8
DataModRef objects 6-5
  filename property 6-5
dataModules
  creating 6-10
  DataModRef object 6-5
  defined 6-9
  using in a Form or Report 6-10
dBASE field types 13-5
dBASE Filter - Non-Indexed 3-14
dBASE Filter dialog box 3-15
dBASE SE
  documentation 1-5
dBASE SE 2.0 1-1
debugging
  bug types 10-1
  Call Stack 10-8
  docking tool windows 10-4
  event handlers 10-8
  general procedures 10-2
  methods of controlling execution 10-5
  overview 10-1
  programs 10-1
  stepping 10-6
  The Debugger 10-1
  The Source Window 10-3
  using breakpoints 10-6
  using watchpoints 10-9
  variables 10-4
decimals
  defining 14-3
default BDE session 6-7
default values (reports) 12-7
defaults
  setting field defaults 14-7
delete-data button 7-7
deleting
  columns (fields) from reports 12-6
  components from palette 5-14
  fields 14-4
  OLE object 15-12
  rows 15-7
DEO (Dynamic External Objects) 1-5
Design mode 7-2
designers
  Design and Run modes 7-2
  Form designer 5-6
  Report 12-5
  Table designer 14-2
  tools 7-1
  undoing 7-14
  windows 7-1
designing
  forms and reports (introduction) 7-1
  tables
    determining relationships 13-3
    field types 13-5
    hints 13-2
    individual fields 13-5
    minimizing redundancy 13-4
    overview 13-1
    structure concepts 13-5
  tables (in Designer) 14-2
  tables (with wizard) 14-2
Determining the User Interface language 17-1
developer support 1-6
displaying default values (reports) 12-7
DISTINCT 11-8
docking tool windows (Debugger) 10-4
documentation
  typographical conventions 1-5
  updates 1-6
dollar sign
  adding to reports 12-7
DOS text 17-8
dQuery/Web
  basic functionality 3-1
  Create Alias dialog box 3-6
  Create Folder dialog box 3-6
  Create Table dialog box 3-8
  creating a Database object 3-4
  creating Query objects 3-7
  defined 3-1
  design surface 3-2
  EasyStart menu 3-2
  Live Data Area 3-12
  Main Menu 3-3
  New dataModule 3-4
  No-Click Reports 3-17
  One-Click Windows 3-19
  Parent-Child Link (creating) 3-11
  Save Datamodule dialog box 3-7
  Select/Add Alias dialog box 3-5
  Set Index (Key) 3-10
  Toolbar 3-3
drill down column (SQL designer) 11-7
drillDown property 12-9
drill-down reports 12-8
duplicate values
  hiding 14-13
duplicates 12-7
  suppressing in reports 12-7
Dynamic External Objects 1-5

E

EasyStart menu 3-2
Edit mode 6-2

Edit Report dialog box 3-18
edit-data button 7-7
editing
  advantage of using forms 15-4
  code 9-1
  code blocks 9-3
  Command window 9-5
  For condition 15-8
  header and bootstrap 5-8
  in-place (Text object) 7-17
  memo fields 15-11
  OLE objects 15-12
  performing calculations 15-9
  special field types 15-10
  subset of rows 15-8, 15-9
  tables
    adding rows 15-7
    counting rows 15-9
    data entry considerations 15-4
    deleting rows 15-7
    OLE fields 15-12
    performing calculations 15-9
    selected table data only 15-3
    selecting rows 15-8
    special field types 15-10
    tools 15-2
  tables vs queries 15-4
  Text object 7-17
  tips 15-4
  using an index to limit rows 15-4
  WFM files 5-8, 5-9
  While condition 15-8
editor 9-1
embedding
  in OLE fields 15-12
errors
  debugging 10-1
event handlers
  debugging 10-8
event-driven applications
  advantages 4-1
  developing 4-4
  how they work 4-2
events
  programming (using Inspector) 7-11
exact matches 17-3
excluding variable types (Debugger) 10-4
executing
  commands 9-5
  programs in the Debugger 10-5, 10-6, 10-7
  queries 11-2
Exists clause
  Criteria Page (SQL designer) 11-4
  example 11-6
  Group criteria page (SQL designer) 11-9
  selection criteria 11-5
exporting table data 14-9
expressions (key) 14-14

F

features 1-1
field components 7-8
field names
  and character incompatibilities 17-7
Field objects
  dataLink property 6-3
  defined 6-3
  value property 6-3
field order (queries) 11-4
Field palette
  and dataLink property 7-8
  displaying fields 7-8
  opening 7-8
  using 7-8
fields
  adding 14-4
  adding to reports 12-6
  binary 15-11
  choosing type 13-5
  custom attributes 14-7
  data entry constraints 14-8
  dBASE types 13-5
  default values 14-8
  defining 13-5, 14-3
  deleting (Table designer) 14-4
  deleting from reports 12-6
  image 15-11
  memo 15-11
  moving (Table designer) 14-4
  naming 14-3
  OLE 15-12
  parent/child values 14-16
  selecting (Table designer) 14-4
  selecting in queries 11-8
  setting defaults 14-7
  sound 15-11
  validity checks 14-8
  viewing special types 15-10
fields array 6-3
files
  adding .SQL to forms and reports 11-2
  adding to projects 5-3
  multiple index (MDX) 14-8, 14-10
Filter By Grid 3-16
Filter mode 6-2
filter-data button 7-7
filtering data
  dBASE Filter dialog box 3-15
  demonstration 3-14
  Filter By Grid 3-16
find and replace 15-6
finding rows 15-5, 15-6
Float field type 13-5
Form class definition 5-8
Form designer 7-1
Form wizard 5-6
Format toolbar 7-17
forms
  creating
    basic steps 5-6
    The Form Designer 5-6
    The Form Wizard 5-6
  creating custom class 5-11
  design guidelines 5-4
  global page 5-11
  master-detail relationships 6-8
    creating 6-7
    masterSource property 6-9
    navigateByMaster property 6-2
    navigateMaster property 6-2
    SQL JOIN statement 6-8
    synchronizing cursor movement 6-2
  multi-page 5-10
  multi-page navigation 5-11
  opening in Run mode 7-18
  printing 7-18
  saving 7-17
  selecting indexes 14-12
  setting colors and fonts 7-16
  using SQL files 11-2
functions
  using in command window 9-4

G

getting started
  Connecting to an SQL database server 2-3
  Installation 2-2
  What you need to run dBASE SE 2-1
global language drivers
  defined 17-4
  vs table language drivers 17-7
global page (forms) 5-11
group criteria page (SQL designer) 11-9
group selection criteria (queries) 11-9
grouped query 11-9
Grouping Page (SQL designer) 11-9
grouping selection criteria 11-7

H

handlers
  event debugging 10-8
Having clause (SQL designer) 11-9

I

image fields 15-11
images
  report background 12-10
importing data 15-7
incompatibilities
  character 17-7
indexes
  active 3-10
  assigning 14-3
  complex 14-14
  complex (rules) 14-14
  concepts (dBASE tables) 14-10
  creating 14-12, 14-16
  deleting 14-13
  hiding duplicate values 14-13
  key expressions 14-14
  modifying 14-13
  planning 14-11
    choosing fields 13-4
    indexes in data entry 14-10
    linking multiple tables 14-11
  selecting 14-12
  Set Index (Key) dialog box 3-10
  subset 14-13
indexing
  planning 14-10
  tables 14-8
  vs sorting 14-8
indexing tables 14-8
inexact matches 17-3
input pane 9-4
insert mode 9-5
inserting
  fields 14-4
Inspector

Events page 7-11
listing properties alphabetically 7-9
Methods page 7-11
Properties page 7-10
using 7-9
yellow highlighting 7-9
installation
How to . . 2-2
un-installing dBASE SE 2-3
What happens during . . . 2-2
internal name 17-8
International issues
specifying a language resource file 17-1
international issues
character incompatibilities in field names 17-7
character sets 17-2
converting between OEM and ANSI text 17-8
exact matches 17-3
identifying code pages and language drivers 17-6
table vs global language drivers 17-7
introduction to dBASE SE 1-1

J

joins 11-12
Joins Page (SQL designer) 11-11

K

key expressions 14-14
knowledgebase
Documentation updates 1-6
What is it . . 1-1
www.dbase.com 1-6

L

Label designer 7-1
Label wizard 12-11
labels
creating printed 12-11
language drivers
and Character sets 17-2
and exact matches 17-3
character incompatibilities in field names 17-7
converting OEM to ANSI 17-8
global 17-4, 17-7
identifying 17-6
table 17-6, 17-7
language resource files 17-1
ldriver 17-4
line length
configuring 9-5
linking 15-12
in OLE fields 15-12
methods to events 9-3
to databases 2-3
Live Data Area 3-12
local tables
using with remote 6-7
Locate mode 6-2
locate-data button 7-7
login name 6-6
Long 13-5

M

master-detail relationships
creating
dQuery/Web demonstration 3-11
with masterRowset and masterFields 6-8
with masterSource 6-9
with SQL JOIN 6-7
in local .DBF tables 6-8
masterSource property 6-9
navigateByMaster property 6-2
navigateMaster property 6-2
SQL JOIN statement 6-8
synchronizing cursor movement 6-2
maximum
finding in reports 12-10
lines in the input pane 9-5
performing aggregate calculations 12-10
maximum line length
code 9-5
configuring 9-5
MDX files 14-8
memo fields 15-11
menuFile property 8-1
menus
and toolbars 8-1
attaching a pulldown menu to a form 8-1
attaching popup menus to a form 8-1
changing properties on the fly 8-7
creating 8-4, 8-5
creating with designers 8-4
events 8-8
example 8-6
features demonstration 8-5
file code 8-6
keyboard shortcuts 8-4
menuFile property 8-1
methods 8-8
popup 8-1
properties 8-8
methods 7-12
built-in 7-11
creating 9-3
deleting 7-11, 7-12
editing 7-12, 9-1, 9-2
linking 7-12
linking to events 9-3
Method menu 7-12
programming (using Inspector) 7-11
unlinking 7-12
verifying 7-12
minimum
finding in reports 12-10
performing aggregate calculations 12-10
monitor execution in Debugger 10-1
moving
fields 14-4
multiline commands 9-5
multi-page forms 5-10
multiple index (MDX) files
dBASE index concepts 14-10
defined 14-8
multiple queries 6-2
multiple selection (Inspector) 7-9
multiple streamFrames 12-11

N

naming conventions
fields 14-3
tables 14-5
navigating
in tables 6-2
Table designer 14-4
navigation buttons
Data Buttons page 7-7
image-style 7-7
multi-page forms 5-11
standard-style 7-7
VCR-style 7-7
Navigator
listing SQL tables 2-4
overview 3-3
new features 1-1
No-Click Reports 3-17
Numeric field type 13-5

O

ODBC databases
connecting to 2-3
OEM 17-4, 17-6, 17-8
code pages 17-2
OLE fields 15-12
One-Click Windows
demonstration 3-19
Field Attributes tab 3-19
General tab 3-19
opening
a file named in code 9-2
Form or Report in Run mode 7-18
SQL designer 11-1
tables 15-1
operators
in queries 11-8
OR 11-7
ordering fields (queries) 11-4
outer joins 11-11
output column name (SQL designer) 11-8
overwrite mode 9-5

P

page numbers
reports 12-8
pageTemplate 12-5
palettes
Component 7-3
Field 7-8
parent tables
master-detail relationships 6-8
referential integrity 14-16
passwords
loginString property 6-6
table access 14-7
popup menu 8-1
primary indexes
creating 14-16
printed labels 12-11
printing
Form or Report 7-18
table structure 14-7
procedures
debugging 10-2
Product registration 1-6

production index 14-10
program files
  editing 9-1
programming (overview) 4-1
programs
  breakpoints 10-6
  debugging 10-1, 10-2, 10-5
  executing in Debugger
    breakpoints 10-6
    controlling program execution 10-5
    monitoring execution 10-1
    running at full speed 10-7
Project Explorer
  adding files 5-3
  creating a project file 5-2
projects 5-2
  adding files 5-3
  creating 5-2
  overview of project files 5-2
properties
  setting (Inspector) 7-10
protected tables 15-2

Q

queries
  and rowsets 6-2
  combining selection criteria (SQL designer) 11-6
  Group Criteria page 11-9
  grouping criteria 11-7
  including unmatched records 11-11
  multiple 6-2
  operators 11-8
  outer joins 11-11
  sorting 11-11
  SQL designer 11-1
  SQL designer (entering data) 11-2
  summary data 11-8
  with indexes 14-11
Query objects
  creating 3-7
  defined 6-1
  rowset property 6-1
  SQL property 6-1
Query Results window 11-2

R

referential integrity
  changing 14-18
  defining 14-16
  What is it . . . 14-16
remote tables 6-7
replacing data 15-6
Report designer 7-1
  elements 12-5
  overview 7-1
reports
  adding columns (fields) 12-6
  adding standard components 12-9
  aggregate (summary) calculations 12-10
  appearance 12-10
  background color 12-10
  background image 12-10
  components (table) 7-7
  creating 12-1, 12-5
  creating borders 12-9
  deleting columns (fields) 12-6
  detailBand (Report designer) 12-5
  displaying default values 12-7
  drillDown property 12-8, 12-9
  Drill-down reports 12-8
  floating dollar sign 12-7
  headerBand (Report designer) 12-5
  indexing for 14-11
  introduction 12-1
  layout 12-8
  master-detail relationships 6-8
    creating 6-7
    masterSource property 6-9
    navigateByMaster property 6-2
    navigateMaster property 6-2
    SQL JOIN statement 6-8
    synchronizing cursor movement 6-2
  multiple streamFrames 12-11
  opening in Run mode 7-18
  page numbers 12-8
  pageTemplate (Report designer) 12-5
  printed labels 12-11
  printing 7-18
  Report designer 12-5
  Report wizard 12-1
  Reports page of the Component palette 7-7
  saving 7-17
  streamFrame (Report designer) 12-5
  summary calculations 12-10
  summary information at top 12-8
  suppressing duplicates 12-7
  using SQL files 11-2
requirements 2-1
resizing
  columns (Table designer) 14-4
resources 1-6
restructuring tables 14-5, 14-6
results pane 9-4
row buffer 6-2
row cursor 6-2
row info (SQL designer) 11-7
rows
  adding to tables 15-7
  counting subset 15-9
  deleting 15-7
  editing subset 15-8, 15-9
  moving (Table designer) 14-4
  sorting 14-9
rowset
  events 6-2
  modes 6-2
  navigation 6-2
  streamSource property 12-11
Rowset objects
  Data Buttons page of the Component palette 7-7
  events 6-2
  fields property 6-2
  overview 6-2
  state property 6-2
  state-property button 7-7
Rules for modifying tables 14-6
Run mode
  opening in 7-18
  using Design and Run modes 7-2
running files 7-2
running tables 15-1
runtime applications
  debugging 10-3
runtime errors
  debugging 10-1
  types of bugs 10-1

S

Sample files 1-4
sample viewer 1-4
Save Datamodule dialog box 3-7
save-data button 7-7
saving
  forms and reports 7-17
  table structure 14-5
scheme (Form designer) 7-16
search and replace 15-6
searching
  tables 15-5, 15-6
secondary indexes 14-16
security
  access
    groups 16-4
    levels 16-4
    table 16-4
    user 16-4
    user profiles 16-4
  data encryption 16-5
  DB tables passwords 16-11
  DBF tables 16-3
  encryption 16-5
  field privileges 16-5
  planning 16-5
    field privileges 16-7
    planning access levels 16-6
    table privileges 16-6
    user groups 16-6
  preset access 16-2
    other tables 16-3
    SQL 16-3
    standard tables 16-2
  privilege schemes 16-4
  removing passwords 16-11
  setting up
    DB tables 16-10
    DBF table privileges 16-8
    DBF tables 16-7
    enforcement scheme 16-10
    field privileges 16-9
    passwords 16-7
    table privileges 16-9
    user profiles 16-8
  table privileges 16-5
  table-level 16-3
  tables assigning 16-9
  tables DB 16-10
  tables selecting 16-9
  user profiles
    changing 16-8
    creating 16-8
    deleting 16-8
    overview 16-4
Select/Add Alias dialog box 3-5
selecting fields (in SQL designer) 11-4
Selecting Specialized Product Fonts 17-6
selection criteria
  adding 11-5
  combining (SQL designer) 11-6
  Criteria Page (SQL designer) 11-4

Selection Page (SQL designer) 11-8
Session objects
adding 6-7
default 6-4
onProgress( ) event 6-4
tables 6-4
sessions
BDE default 6-7
setting
custom classes 5-12
properties 7-9
Properties Page (Inspector) 7-10
simple expression (queries) 11-5
simple Having expression 11-9
sorting
and language issues
character incompatibilities 17-7
character sets 17-2
exact match 17-3
query results (SQL designer) 11-11
Sorting Page (SQL designer) 11-11
tables 14-8, 14-9
Sorting Page (SQL designer) 11-11
sound fields 15-11
Source Aliasing 1-5
Source editor
miscellaneous notes 9-2
opening 9-1
setting preferences 9-1
using 9-1
source window (debugger) 10-3
SQL 11-1
designer elements 11-1
learning 11-1
queries visual 11-1
SQL databases
connecting to 2-3
listing tables in Navigator 2-4
SQL designer 11-2
adding tables 11-3
entering data 11-2
removing tables 11-3
renaming tables 11-3
Source editor and 11-2
SQL files
using in forms and reports 11-2
SQL Links 2-3
SQL property 6-1
SQL Property Builder dialog box 11-2
SQL queries
combining selection criteria (SQL designer) 11-6
group criteria 11-9
grouping criteria 11-7
including unmatched records 11-11
inner and outer joins 11-11
Joins Page (SQL designer) 11-11
operators 11-8
Selection Page (SQL designer) 11-8
sorting 11-11
summary data 11-8
SQL tables
using with local 6-7
stack
call 10-8
standard deviation
aggregate calculations 12-10
finding in reports 12-10
Standard tables
local SQL and 13-1
stepping 10-6
stopping execution (debugger) 10-7
stored procedures 6-7
StoredProc objects
and Database objects 6-5
defined 6-5
params array 6-5
rowset property 6-5
using 6-7
vs. Query objects 6-5
streamFrames
multiple 12-11
streamSource property 12-11
subscripts
See also array elements
summary calculations (reports) 12-10
summary data in queries 11-8
suppressing duplicates (in reports) 12-7
syntactical errors 10-1
system requirements 2-1

T

tabbing order
changing 5-5
guidelines 5-5
Table design guidelines 13-2
Table designer 14-4
a demonstration 3-8
abandoning changes 14-5
deleting fields 14-4
moving fields 14-4
navigating 14-4
resizing columns 14-4
saving table structure 14-5
tips 14-2
user interface 14-3
using 14-2
table language drivers 17-6, 17-7
table pane (SQL designer) 11-3
Table wizard 14-2
tables
abandoning changes (Table designer) 14-5
accessing 6-1
accessing (table of components) 7-6
adding fields 14-4
assigning field attributes 14-7
Binary fields 15-11
child 14-16
choosing type 13-5
complex indexes 14-14
creating (in designer) 14-2
creating (with wizard) 14-2
creating in dQuery/Web 3-7
creating indexes 14-16
creating secondary indexes 14-16
Data Access page 7-6
defining fields 14-3
deleting fields 14-4
deleting indexes 14-13
designing
a demonstration 3-8
defining fields 13-5
Helpful hints 13-2
minimizing redundancy 13-4
overview 13-1
relationships among tables 13-3
structure concepts 13-5
terms and concepts 13-1
editing
add rows 15-7
counting rows 15-9
data entry considerations 15-4
delete rows 15-7
performing calculations 15-9
precautions 15-1
selected data only 15-3
setting criteria 15-8
special field types 15-10
toolbar descriptions 15-2
exporting data 14-9
field types 13-5
hiding duplicate values 14-13
index concepts (dBASE) 14-10
indexes 14-12
indexing 14-8, 14-10
indexing (vs sorting) 14-8
indexing subset 14-13
linking 14-11
linking a form or report 6-5
listing in Navigator 6-5
master-detail relationships 6-8
creating 6-7
navigateByMaster property 6-2
navigateMaster property 6-2
SQL JOIN statement 6-8
synchronizing cursor movement 6-2
masterSource property 6-9
Memo fields 15-11
modifying indexes 14-13
naming 13-5
navigating fields (Table designer) 14-4
OLE fields 15-12
opening 15-1
parent 14-16
printing structure 14-7
protected 15-2
referential integrity 14-16, 14-18
relationships 13-3
restructuring 14-5, 14-6
restructuring guidelines 14-5
rules for modifying 14-6
running 15-1
saving structure 14-5
searching 15-5, 15-6
security (dBASE and Paradox) 6-4
seeing in Navigator 2-4
selecting indexes 14-12
setting type 14-2
sorting 14-8, 14-9
SQL JOIN statement 6-8
terms and concepts 13-1
using local and remote together 6-7
technical support 1-6
Text objects
editing 7-17
tool windows
docking in Debugger 10-4
tool windows (debugger) 10-4
toolbars 8-8
adding, editing and navigating 8-5
and menus 8-1
attaching to forms 8-2
changing properties "on the fly" 8-7

creating custom 8-3
defining custom 8-3
events 8-10
methods 8-10
properties 8-10
sample code 8-6
Two Summary Expression 11-10

U

undoing (in designers) 7-14
un-installation 2-3
user interface 5-6
creating forms 5-6
event-driven 4-1, 4-2, 4-4
form design 5-4
multi-page forms 5-10
specifying a language resource file 17-1
standard components (table) 7-4
tabbing order 5-5
Table designer 14-3
using
ActiveX controls 7-8
Code Block Builder 9-3
custom classes 5-12
data modules 6-10
The Inspector 7-9

V

validity checks 14-8
value property 6-3
values
displaying default (reports) 12-7
variable types 10-4
in the Debugger 10-4
variance 12-10
finding in reports 12-10
viewing 11-2
query results 11-2
special field types 15-10
visual designers 7-1

W

watchpoints 10-9
Web address (dBASE) 1-6
WFM files 5-8, 5-9
class definition 5-8
editing 5-8, 5-9
sample code 5-7
structure 5-7
Where clause 11-4
windows
designers 7-1
tool (debugger) 10-4
windows language setting 17-4
Windows text 17-8
wizards
Form 5-6
Label 12-11
Report 12-1
Table wizard 14-2

Z

z-order 5-5