Upgrade Guide

Visual dBASE® for Windows

Introduction

Copyright Agreement

Visual dBASE combines an intuitive and dynamic 4th generation language with an extensive set of design tools to create a Rapid Application Development (RAD) that is as easy to learn as it is to use. *Visual* dBASE builds on the object-oriented foundation of dBASE 5.0 by adding the tools most requested by power users, programmers and professional client/server developers. To create tomorrow's applications quickly you need a mature object and event model, visual tools that provide inheritance, advanced client/server features and an easy method to distribute your applications. *Visual* dBASE brings you these tools without tying you to a specific table format or breaking the link between visual tools and source code. In addition to an expanded and refined tool set, *Visual* dBASE embraces Windows95 while maintaining full compatibility with Windows 3.1.

This guide shows you how to upgrade from dBASE 5.0 for Windows to *Visual* dBASE. It explains the new features and syntax and helps you migrate applications.

This manual, together with extensive online Help provide the information you'll need to work productively in dBASE Desktop. Help also contains the complete language reference. You can also use this manual in conjunction with the dBASE 5.0 manuals to form a complete hard copy reference library.

Contents of the Upgrade Guide

Chapter 1, "Setting up *Visual* dBASE." Start here for an overview of the setup program and installation options.

Chapter 2, "What's new in *Visual* dBASE." A quick look at all the new features including the expanded visual tool set, programming enhancements, data administration features, and Windows95 support. The chapter also contains a brief description of the add-on *Visual* dBASE Compiler.

Chapter 3, "Upgrading applications." Tells you how to migrate your applications from dBASE 5.0 for Windows to *Visual* dBASE. Refer to the "Programmer's Guide" for information on converting applications from dBASE for DOS.

Chapter 4, "Working with the new visual tools." This chapter provides in-depth coverage of the Experts, the new Form Designer, Popup menus and the MenuBar.

Read this chapter to learn about visual inheritance and visual class design. You will also learn how to take advantage of new control and field palettes.

Chapter 5, "Database administration." A guide for creating table security schemes and designing referential integrity rules. This chapter provides information on data encryption and creating, modifying, and deleting referential integrity rules on Paradox tables, and tables on remote servers attached via SQL Links and ODBC.

Chapter 6, "Enhancements to the dBASE Language." Here is an overview of changes to the dBASE language. It covers new commands and functions used for database operations and working with encrypted tables. Read this chapter to learn about the enhanced encapsulation features and see how to use OLE automation. This chapter also shows you how to use literal arrays and take advantage of Windows95.

Chapter 7, "New commands, functions, classes and properties." This is a subset of the *Language Reference* containing changes since dBASE 5.0. The complete *Language Reference* is available in online help.

Chapter 8, "Local SQL." Learn how to mix and match dBASE commands with SQL. This chapter is a reference for the SQL syntax supported by BDE (Borland Database Engine).

Chapter 9, "Learning and extending *Visual* dBASE." The final chapter provides you with resources for learning more about *Visual* dBASE and what tools can help you extend the dBASE environment. It describes what types of training materials are available and some popular add-on products.

Chapter

1

Setting up *Visual* dBASE

This chapter describes the differences between the dBASE 5.0 installation program and the *Visual* dBASE Setup program. Refer to the dBASE 5.0 Getting Started manual for details on configuring the Borland Database Engine (BDE). BDE is the new user friendly name for IDAPI.

The README file

Disk 1 of the dBASE disk set contains a file called README.TXT. This file provides the latest information on installing and running *Visual* dBASE.

To view the README file, insert disk 1 and open the file with any text editor or word processor. You can use the Notepad accessory in Windows 3.1 or the WordPad accessory in Windows95.

Installing over dBASE 5.0

The Upgrade Edition of *Visual* dBASE only allows installation on a PC that contains a copy of dBASE 5.0 for Windows. Keep a dBASE 5.0 disk set on hand to reinstall or move dBASE to another machine. If you need to install *Visual* dBASE on a new machine, install dBASE 5.0 first and then run the *Visual* dBASE setup program.

CD-ROM setup

The CD-ROM Edition of *Visual* dBASE includes a special installation guide. The sleeve contains the CD-ROM installation guide.

Running the Setup program

To run the setup program,

1. Insert Disk 1 into your disk drive
2. The ways to invoke the setup program depend on the version of Windows and shell you are using. Here is a guide for the most common configurations:
   - In the Windows 3.1 Program Manager, select File | Run and enter `A:SETUP.EXE`
   - In Dashboard, select Dashboard Run and enter `A:SETUP.EXE`
   - In Norton Desktop, press Ctrl-R to open the run dialog and enter `A:SETUP.EXE`
   - In Windows95, select Start | Settings | Control Panel | Add and Remove Programs. The Setup wizard will locate and run SETUP.EXE for you. Windows95 also allows you to run Windows applications including the SETUP program from the MS-DOS window.
3. Select from the following options:
   - **Complete** installs all *Visual* dBASE files.
   - **Minimum** installs a limited configuration that includes all files required to run dBASE including the Borland Database Engine. Useful for laptops with limited hard drive space, the minimum option does not include samples, help, or utilities and takes under 10 MB of disk space.
   - **Custom** gives you more granular control over what files it installs. From within custom you can select the following file groups.
     - ***Visual* dBASE** includes the main dBASE program and the Borland Database Engine. It includes all required files to work with every dBASE object except Report files.
     - **Sample Files** includes the samples used in the documentation and online help. It also contains the Music application that shows you how to combine several dBASE objects into a larger application. Samples for the dBASE API for C++ programmers are provided in the EXTERN subdirectory of the SAMPLES directory.
     - **Custom Controls and Utilities** gives you a wide variety of custom controls you can use when developing forms. It also includes several utility programs written in dBASE. Custom controls provide reusable components such as OK, Cancel and Close buttons. The utilities include the Component Builder and SQL Statement Builder. The Component Builder translates dBASE III PLUS and dBASE IV files into *Visual* dBASE equivalents. The SQL Statement Builder is an interactive query tool that creates and executes SQL commands.
     - **Interactive Tutors** provide computer based training for various parts of the dBASE environment. They also include lessons on migrating from the Control Center.
     - **Crystal Reports** includes the report designer, online help for the designer and the personal trainer. The personal trainer is an interactive tutor that is tailored to the report designer.

- **Online Help** includes the complete language reference and context sensitive help for the dBASE Desktop, Visual Property Builders, designers and dialogs. There is also a help file for using SQL and connecting to remote data sources with SQL-Links and ODBC.

4 The setup program requires that you register your software by entering a name and company. Setup records both items in the [Install] section of the INI file. After the setup is complete you may edit the INI file to change the name and company that appear at startup. The [Install] section will look similar to the following:

```
[Install]
Username=Paddy Moloney
Company=Chieftain Industries
```

5 The setup program completes by creating a program manager group and prompting you to view the README file. Be sure to view the README file at this time, if you did not already view it prior to running SETUP.

After installing you may need to modify the BDE (Borland Database Engine) configuration file (IDAPI.CFG). Please refer to Appendix B of the dBASE 5.0 Getting Started manual for more information. The utility is now called "BDE Configuration Utility" but it works the same as the "IDAPI Configuration Utility" described in Appendix B.

If you plan to use ODBC drivers with dBASE you might need to copy the ODBC.NEW and ODBCINST.NEW files to your WINDOWS\SYSTEM directory as ODBC.DLL and ODBCINST.DLL. Setup installs ODBC.NEW and ODBCINST.NEW into the BDE directory. These files are an update to some of the earlier versions of ODBC.

Chapter

2

What's new in *Visual* dBASE

Visual dBASE is Borland's second generation, award winning, object-oriented, event-driven database manager. *Visual* dBASE has enhanced usability, more programming power, and more robust database support. In addition, an EXE compiler is now available separately. *Visual* dBASE provides all the tools required for RAD (Rapid Application Development), while the compiler provides rapid application deployment.

Read this chapter for an overview of the new and enhanced features you will find in *Visual* dBASE.

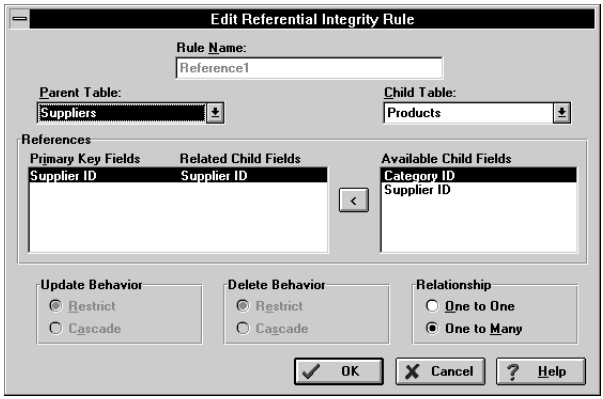
- **Robust Database Support** allows comprehensive database administration and offers client/server features including embedded ANSI SQL-92 and access to stored procedures. *Visual* dBASE is the only Xbase development tool to include an ANSI SQL-92 compliant implementation of SQL. The data administration tools help you to setup table security and design referential integrity rules.
- **Expanded Visual Tools** includes an overview of each Expert and enhancements to the Two-Way-Tools. The new Experts allow users to complete comprehensive tasks by answering a few simple questions. Enhancements to the Two-Way-Tools let you visually create and inherit from reusable components such as base form sets and custom controls.
- **More Programming Power** introduces the new and enhanced stock classes that ship in *Visual* dBASE, allowing programmers easy access to a wide range of powerful capabilities.
- **Windows95 Support** explains how *Visual* dBASE can exploit the new interface and file system of Windows95.
- **Application Distribution** describes the *Visual* dBASE compiler, Help compiler and application deployment technology. These tools integrate into the dBASE environment to compile your applications files into a single EXE file and create installation and setup systems for diskette and CD-ROM.

Robust database support

Visual dBASE brings advanced client/server features to a scalable environment. It provides the tools required for serious client/server and file-based application development. *Visual* dBASE lets you leverage the unique strengths of back-end servers and gives you more control over the DB and DBF table formats. In addition to the pass-through SQLEXEC( ) function, you can now embed ANSI SQL-92 commands in any dBASE program, creating a tight bond between the dBASE data environment and your server. If your data resides in a server supporting stored procedures you can call them directly from a dBASE program. You can also take advantage of server specific transaction isolation levels. *Visual* dBASE also contains new ANSI language drivers and interactive tools for administering relational integrity rules, setting up table security, and inspecting field attributes.

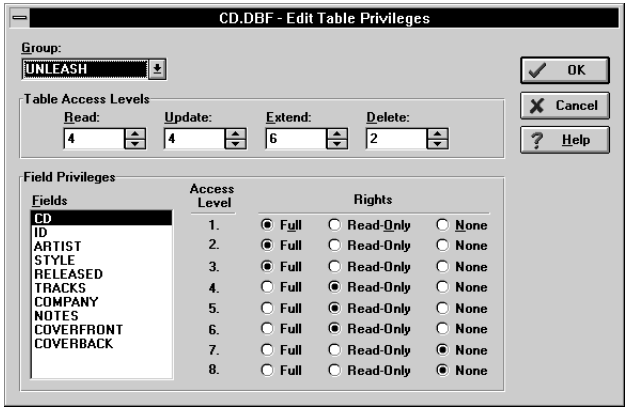
- **Embedded ANSI SQL-92** works seamlessly within dBASE programs. There is no need to SET SQL ON or create separate PRS files. Embedded SQL works with any data source including DB, DBF, the Local InterBase Server, and remote servers attached via ODBC or SQL-Link drivers. The result set of an SQL Select statement is a standard dBASE work area that you can continue to work with using the traditional Xbase DML. Unlike traditional client/server tools that create temporary tables from SELECT statements, dBASE can create a full read/write cursor into the source tables. dBASE allows you to mix and match embedded SQL, pass-through SQL and traditional Xbase DML.
- **Stored Procedures** on a server can extend the dBASE language as external functions. *Visual* dBASE uses the EXTERN command to declare both stored procedures on a server and functions in a DLL. If your server can provide information on stored procedures, an AUTOEXTERN option is available. Servers capable of providing the necessary AUTOEXTERN information include Oracle 7 and InterBase 4.0.
- **Transaction Isolation Levels** give developers the option of taking exacting control over transactions. Depending on your server, you can work with uncommitted changes, committed changes, or full repeatable reads.

Figure 2.1 Editing referential integrity



- **Referential Integrity Tools** gives database administrators a visual interface to examine and change the integrity rules occurring between tables. Administrators can set rules for parent and child tables with options for cascading or restrictive updates. Rules appear at the database level for database servers and at the directory level for Paradox tables.

Figure 2.2 Setting up table security



- **Table Security** protects sensitive tables through encryption and a password system. Database Administrators can set up to eight privilege levels at both the table and field level. The security system is compatible with the encryption schemes found in both dBASE for DOS and Paradox.
- **The Field Inspector** is a new tool for creating column integrity rules while in the table designer. Field Inspector provides an interface that is consistent with the Form Designer's object inspector. You can set field properties such as default value, maximum, minimum, and if the field is required or not. The properties available depend on the type of table you are opening and the driver you are using to open the table.
- **True NULL Support** is now available for working with Paradox and SQL tables. You can search for and update tables with NULL values, create NULL variables, and use NULL in any comparison or expression.
- **ODBC Connectivity** is now a standard feature of the Borland Database Engine. The new Borland Database Configuration utility lets you define database alias names for any standard ODBC driver. You can use this feature to work with other PC table formats such as Access, Btrieve, and FoxPro as well as database servers such as DB2 and INGRES.
- **ODBC Database Administration** is supported throughout *Visual* dBASE. You can create ODBC tables using the table designer and the Table Expert. You can examine and update Referential Integrity rules in ODBC databases. If your ODBC driver supports Stored Procedures, you can work with them just as you can work with Stored Procedures through SQL-Link drivers.
- **ANSI Language Drivers** provide international customers with seamless integration into multi-lingual SQL and ODBC systems. The new drivers also offer a significant

performance boost in applications that previously required OEM/ANSI conversion routines.

- **The Upsizing Expert**, sold separately, helps developers migrate tables and integrity rules between local, network, and database servers. Since the Upsizing Expert works through the Borland Database Engine, it is fully compliant with all the data sources that are available to *Visual* dBASE. Contact Borland at 1-408-431-1000 for information on availability and pricing.

Expanded visual tools

Visual dBASE has new Experts for creating tables, reports, and labels. There are also many new options for customizing the Form Expert. Experts present the user with a logical series of steps for defining a new file and use the same consistent interface that allow the user to move back and forth through steps to change any option before creating the new file.

- **The Table Expert** guides you through the table design and creation process. You can use the Table Expert to select common table and field definitions. The Table Expert allows you to mix and match fields from any of the table templates. You can modify table templates to customize the Table Expert for your organization. In addition to native DB and DBF support, the Table Expert offers the ability to create tables using ODBC and SQL-Link drivers.
- **The Form Expert** is now fully customizable. The Form Expert is capable of creating a wide range of layouts. It provides four basic layouts: columnar, form, browse, and one-to-many. Layouts can appear as either single- or multiple-page forms. Control associations let you change what control appears for each field type. You can also apply custom color and font schemes. The schemes are similar to the color schemes found in the Windows control panel.
- **The Report Expert** guides you through designing the detail, grouping, summary, and grand total sections of a report. It includes an easy way to get complex statistical information such as standard deviation by region and distinct count within a department. The Report Expert runs within the *Visual* dBASE Desktop and creates Crystal compatible reports without opening the Crystal designer.

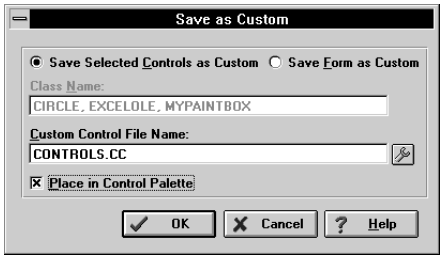
Figure 2.3 Using Quick Address

- **The Label Expert** provides advanced features such as Quick Address and Calculated Fields. The Quick Address feature shown in Figure 2.3, scans your tables for fields such as 'Name', 'Street', 'Zip' and organizes them into a standard address layout. You can accept the default address layout or continue customizing it from the Expert environment. To create calculated fields, enter simple expressions or open the Expression Builder for assistance with complex calculations. The Label Expert provides over 45 Avery label definitions.

Visual dBASE includes extensive refinements that make it easier to leverage the dynamic object model. You will find significant enhancements that allow the Two-Way-Tools to support completely visual subclassing and inheritance. The designers also include many improvements based on customer feedback.

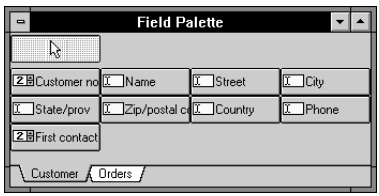
- **The Custom Form Class Designer** is a new Two-Way-Tool for visually creating base form sets. A base form set is a collection of custom form classes. Custom form classes work like style sheets when creating new forms. When you use start with a custom form class, the new form inherits properties, methods, and controls from the custom form class. The dynamic object model insures that any modifications made to a custom form class automatically ripple down to derived forms.
- **Visual Inheritance** lets you use the Form Designer to derive new forms from custom form classes. For instance, you can set the Form Designer to use a custom form class containing a group of speed buttons. When you set a custom form class, it's controls, properties, and events appear in any form you open or create with the Form Designer. If you later change the custom form class to add a new speed button, it appears in any derived forms. The Form Designer also allows you to switch the base form of existing forms to other custom form classes or back to the stock FORM class.

Figure 2.4 Saving a group of custom controls



- **Visual Custom Control Design** is now possible right from the Form Designer. You can save any control as a custom control by simply selecting it and picking File | Save as Custom. You get the option of adding to an existing library or making a new custom control library. As with forms, saving a control creates source code that you can modify with any text editor. Since *Visual* dBASE relies only on source code when working in the designers, any changes you make to the custom control source code automatically transfer to the control palette.
- **The Control Palette** is now fully customizable. You can dynamically dock the palette on the top or bottom of the form designer or keep it in a resizable window. Controls can appear in bitmap only mode with SpeedTips (tool tips), with text or as text only. VBX controls now appear on a separate tab. You can save screen real estate by removing the tabs. When tabs are off, all controls appear on a single page. The Control Registration table (CREG0009.DBF) allows you to create your own custom control groups and specify a bitmap for each custom control.

Figure 2.5 The Field Palette



- **The Field Palette** gives you an easy way to add controls for each field in the active view. When you add controls from the field palette, dBASE automatically sets up the DataLink and DataSource properties. Like the Control Palette, the Field Palette can be docked or used as a floating window.
- **Control and Field Associations** determine what controls are used for what data types. For instance you can associate either an EntryField or a SpinBox with numeric fields. This association is used by both the Form Expert and the Field Palette. You can change the associations at any time.
- **Font and Color Schemes** give you a quick way to try out different font and color combinations. *Visual* dBASE provides a wide selection of schemes that you can apply to existing forms. The colors in a scheme can be specific or relative. Relative colors

correspond to the desktop colors set in the Windows control panel. If you use relative colors, your application will conform to the desktop colors in use at run time. You can also create and save new schemes from the scheme dialog.

- **New Image** support for ICO, EPS, TIF, and WMF files in addition to PCX and BMP files. You can use any of the supported formats with the IMAGE class, Binary fields, the Navigator, and the RESTORE IMAGE command.
- **More Visual Property Builders** for inspecting and setting properties. Throughout *Visual* dBASE, many dialogs including the Inspector have the option of entering values directly or using a Visual Property Builder for guidance. Experienced developers can enter values directly while new users can use Visual Property Builders to learn all the options available for a given property or expression. You can use the new Visual Array Builder to create an array for a ListBox or ComboBox without leaving the Inspector.
- **DesignView** is a new form property for setting up design time data environment. It provides you with the tables and relations you need to design a form that inherits an existing data environment at run time. You can use DesignView in place of View to create forms that share a single data environment.
- **New Utilities** include the SQL Statement Builder and an enhanced Component Builder. The SQL Statement Builder is an interactive tool for creating and learning about SQL commands. The Component Builder now supports conversion of both dBASE III PLUS and dBASE IV files including forms, reports, labels, programs, and menus.

More programming power

The second generation of the *Visual* dBASE object and event model adds a wealth of new stock classes to complement the existing set of user interface, array, DDE and OLE classes. Existing stock classes also get new properties for multiple page forms and more control over event processing. Greater encapsulation is now available through protected properties and member functions.

- **Multiple Page Forms** are a great way to separate large groups of controls and options into logical pages. *Visual* dBASE uses multiple page forms for tools such as the Inspector, the Controls window, and the Desktop properties dialog. The form designer allows you to quickly place controls on different pages and navigate between pages. Any control can appear on any page or on all pages. *Visual* dBASE provides a special page zero for controls you want on all pages. Developers have complete control over page navigation. The most common technique for page navigation is the TabBox control, however developers can also use any other control such as PushButtons, Menus, SpinBoxes, and VBX controls.
- **TabBox** is a user interface class that gives you the same tab controls that *Visual* dBASE uses in its desktop. Although the TabBox is normally associated with multiple page forms, the *Visual* dBASE implementation does not limit it to any specific use.

- **MenuBar** acts as the root object of a completely object-oriented menu tree. It works with the new Menu Designer to automatically create the Edit and Window menus found in applications that conform to the MDI model. The Edit menu items (undo, cut, copy and paste) automatically dim based on the contents of the clipboard and if any text is selected. The Window menu lists all MDI windows and can switch focus to any selected window.
- **Popup** menus appear on-demand. A common use of popup is a menu that appears when you right-click. The SpeedMenus that appear when you right-click in the dBASE Desktop are popup menus. The Popup Designer is a new Two-Way-Tool for visually programming a custom popup class. The Form class has a new PopupMenu property that makes creating SpeedMenus as easy as attaching a MenuBar to a Form.
- **Shape** is a simple user interface class that you can use to draw shapes (circles, ellipses, squares, etc.) on a form. The properties, methods, and behavior are similar to those of the line class.
- **PaintBox** is for advanced developers that want to harness the Windows API to create their own controls. This class provides a generic control that developers can use as a device context for Windows API functions. The developer controls all actions of a PaintBox: what is displayed, how keystrokes are handled, etc. *Visual* dBASE keeps track of where the control fits in the Z-Order, what page it appears on and provides a wide array of events such as focus, mouse, keyboard, and paint messages.
- **OLEAutoClient** complements the OLE field support and the OLE control. You can now use *Visual* dBASE's dynamic object model to control OLE 2 applications that provide server automation. For example, a dBASE program can create an instance of the OLEAutoClient class that points to MS Word. After establishing a connection to MS-Word, the properties of the instance variable are WordBasic commands. The dBASE program can then start controlling Word by issuing WordBasic commands.
- **AssocArray** or Associative Array is a new array class that lets you use character strings as the index to the array. Like the standard array class, AssocArray contains a complete set of methods and properties for navigating through and manipulating the array. The AssocArray class dynamically resizes the associative array as you add and remove elements.
- **Expanded Properties** for existing classes not only provide for multiple page forms, but also give you exacting control over the record buffer, key processing, window handling, colors, and positioning.
- **Strong Encapsulation** is now available for any subclassed object. You can hide or protect any member of a class. Only the methods of the same class can read or write to a protected member.
- **Debugging** is now easier than ever. The dBASE debugger now lets you inspect and modify values contained in local and static variables. You can use this feature to change properties of a form without leaving the debugger.
- **Literal Array Declaration** lets you create and populate arrays with a single statement. The following command creates an array with three character elements:

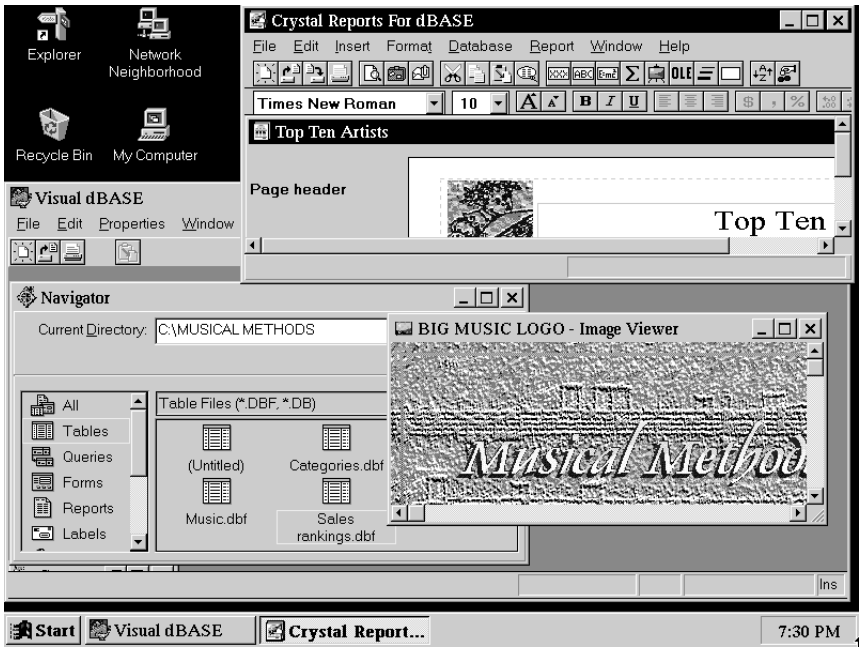
```
ColorArray = {"Red","White","Blue"}
```

Note To accommodate the new array syntax, code blocks must begin with a semicolon or the pipe character. dBASE 5.0 allowed code blocks containing values without an initial delimiter. *Visual* dBASE treats value-only code blocks without initial delimiters as single element literal arrays.

Windows95 support

Visual dBASE supports Windows95 specific features such as long file names, extended attributes, and universal naming conventions.

Figure 2.6 *Visual* dBASE running under Windows95



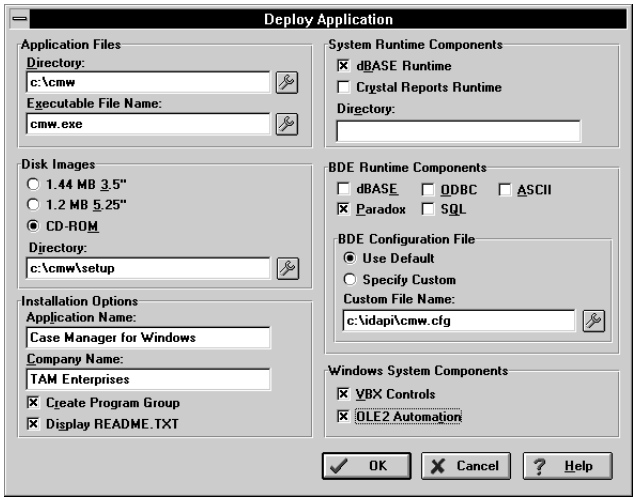
- **Long File Names** are one of the biggest changes in the Windows95 new file system. When running under Windows95, *Visual* dBASE automatically supports the new system's long file names. The Navigator and all other dialogs adjust to fit long file names and maintain case sensitivity.
- **Extended Attributes** are another feature new to the Windows95 file system. The dBASE language has a new set of corresponding methods and functions for working with the extended file attributes.
- **Emerging Interface Standards** appear throughout Windows95. These changes include placing the file name first in the windows caption and support for the application key found on newer keyboards such as Microsoft's Natural keyboard. The application key opens the SpeedMenu, while Alt-Application opens a property sheet for the current object. Shift-F10 substitutes for the Application key on keyboards without the Windows specific keys.

Application distribution

The *Visual* dBASE Compiler includes the compiler technology and utilities you need to distribute your dBASE applications as easily installable royalty free EXE files. The *Visual* dBASE Compiler comes with the compile and build extensions for *Visual* dBASE, a help compiler, and an application deployment system.

- **The dBASE Compiler** gives you an easy way to compile your dBASE applications into EXE files. The compiler extends *Visual* dBASE with new commands and dialogs that can create royalty-free EXE files. The compiler can automatically descend through an application to find and link all relevant files without the need for a make or project file. The dBASE compiler also lets you specify an icon and a splash image. You can dynamically customize your own EXE files using an INI file. In keeping with dBASE tradition, all the options for creating executables are available from both the language and visual tools. Like other *Visual* dBASE dialogs, the compiler dialogs help you learn the new syntax by displaying the new commands in the Command window.
- **The Help Compiler** generates HLP files that can connect your *Visual* dBASE applications to the Windows Help system. You can create context-sensitive help systems for your dBASE applications by setting the HelpID property of any control to a topic in your HLP file. You can use any editor that saves to RTF (Rich Text Format) to create your help system. RTF capable editors include Word, WordPerfect, and the Windows95 WordPad accessory.

Figure 2.7 Creating setup diskettes



- **The Application Deployment System** completes the last step of the development process. For the first time, you can use the same install engine that Borland uses to create your own setup diskettes. The install engine includes a special version of the famous 'freeway' setup program customized for compiled *Visual* dBASE applications. When you deploy an application, the install engine provides data compression, group creation, database drivers, and options to brand the setup

program with an application and company name. The resulting disk set provides a clean distribution system with no changes to your end-user's WINDOWS or WINDOWS\SYSTEM directories other than a program group and optional VBX and OLE2 support files.

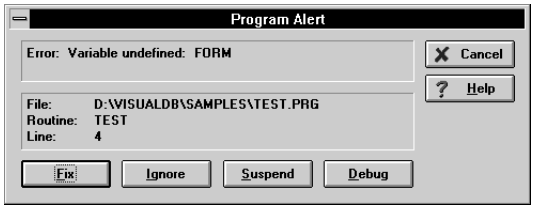
Chapter

3

Upgrading applications

Visual dBASE runs most dBASE 5.0 applications without change. However, a few applications will require some fine tuning to run correctly. Read this chapter to see if you need to modify any of your applications.

Figure 3.1 Error from invalid code block



By far, the most common modification will be in the area of code blocks. dBASE 5.0 allows value only code blocks without initial vertical bars or a semicolon. These will appear as arrays in *Visual* dBASE. Figure 3.1 shows the error that will appear if you run a program or form that refers to FORM in what was a code block under dBASE 5.0 and is an array under *Visual* dBASE.

The following code shows an OnClick that would close a form under dBASE 5.0 and causes an error in *Visual* dBASE. The code blocks for the Valid events also cause errors. The OnClick can be fixed with a single semicolon. The Valid events will need a little more work.

```
* Runs with dBASE 5.0 and has errors in Visual dBASE
DEFINE FORM f1
DEFINE PUSHBUTTON done OF f1 ;
       PROPERTY ;
       OnClick { FORM.CLOSE() }
DEFINE ENTRYFIELD firstname OF f1 ;
       PROPERTY ;
       top    5, ;
       value SPACE(20), ;
       valid  { .NOT. ISBLANK( this.value ) }
DEFINE ENTRYFIELD lastname OF f1 ;
       PROPERTY ;
       top    7, ;
       value SPACE(20), ;
       valid  { .NOT. ISBLANK( this.value ) }
f1.OPEN()
```

The easiest way to update an application to the new syntax is to place a semicolon at the beginning of the code block. Using a semicolon is recommended if you do not need a return value from the code block. Only a few events such as CanClose, Key, Valid, and When require a return value. If you need a return value you can precede the value with a semicolon and a return statement or use, ||, vertical bars. The vertical bars delimit parameters. See the language reference for more information on using parameters with a code block. Here is the same procedure updated to run in *Visual* dBASE.

```
* Updated for Visual dBASE
DEFINE FORM f1
DEFINE PUSHBUTTON done OF f1 ;
       PROPERTY ;
       OnClick { ; FORM.CLOSE() }
DEFINE ENTRYFIELD firstname OF f1 ;
       PROPERTY ;
       top    5, ;
       value SPACE(20), ;
       valid  { ; RETURN .NOT. ISBLANK( this.value ) }
DEFINE ENTRYFIELD lastname OF f1 ;
       PROPERTY ;
       top    5, ;
       value SPACE(20), ;
       valid  { || .NOT. ISBLANK( this.value ) }
f1.OPEN()
```

Note The empty parameter delimiters (i.e. vertical bars) are not compatible with dBASE 5.0. If your application needs to run with both dBASE 5.0 and *Visual* dBASE, begin the code block with "**; RETURN**".

Another unexpected error occurs if you created a custom property with the same name as a new stock property. For instance, if you are already using a custom property called SpeedTip, *Visual* dBASE will overwrite it with the new SpeedTip stock property. You might glance through the new property list to see if anything looks like one of your custom properties. If so, simply rename your property to something unique.

There are new options on the CREATE and MODIFY commands to invoke Experts, prompt for Experts and so on. If you need to create or modify a file that has the same

name as a new option, you need to specify the complete name and place it in quotes. For example to create a new table called Expert, enter the following:

```
CREATE "expert.dbf"
```

The new default setting of CONFIRM is ON. The previous default was OFF. When CONFIRM is OFF, filling up an entry field automatically moves focus to the next control. Although this is handy for data entry applications, it does not conform to the standard Windows interface. If you want CONFIRM to be OFF, simply place the following command at the beginning of your application.

```
SET CONFIRM OFF
```

All color properties now default to relative color strings. These strings correspond to the areas defined in the Windows Control panel. The valid realtive color strings are as follows:

Scrollbar
Background
ActiveCaption
InActiveCaption
Menu
Window
WindowFrame
MenuText
WindowText
CaptionText
ActiveBorder
InActiveBorder
AppWorkSpace
Highlight
HighlightText
BtnFace
BtnShadow
GrayText
BtnText
InactiveCaptionText
BtnHighLight

Chapter

4

Working with the new visual tools

This chapter explains the new and enhanced tools for creating tables, forms, menus, popups, reports, and labels. While many of the areas of the dBASE Desktop have been polished and revamped since version 5.0, you should be aware of the following changes:

- **New Experts** for creating tables, forms, reports, and labels. The Experts provide easy step by step guidance and allow for customization. Development team leaders and departmental administrators can use the customization features to insure adherence to company or departmental standards. Along with the Experts themselves, there are some related language and option changes.
- **The Custom Form Class Designer** is a new Two-Way-Tool to create and modify base form sets. Base form sets are collections of custom form classes from which you can derive new forms.
- **The Form Designer** has undergone significant changes to improve usability and customization. Visual inheritance allows you to derive forms from base form sets by setting a custom form class. You can also save existing forms and controls as reusable components. The new Field Palette provides automatic DataLinking and control selection. You can customize the new Control Palette with your own bitmaps and groups. The new form parameter and the DesignView property make it easy to create multiple form applications.
- **The Popup Menu Designer** is a new Two-Way-Tool for visual programming of on-demand menus. Popup menus that open on a right click are SpeedMenus. The new POPUP class gives you a root object for your own on-demand menus and SpeedMenus. You can quickly create a SpeedMenu using a form's PopupMenu property.
- **The MenuBar** is the new root menu object. You can use it in the menu designer to easily create common clipboard functions such as the Edit | Cut, Edit | Copy, Edit | Paste and Edit | Undo.

- **The SQL Statement Builder** is a new utility for visually building SQL (Structured Query Language) commands. You can use it to learn the differences between SQL and traditional Xbase DML (Data Manipulation Language).

Note This chapter assumes you have a working knowledge of the designers provided with dBASE 5.0. For detailed information on any of the designers, please refer to the dBASE 5.0 User's Guide.

The Experts

Experts provide step-by-step guidance for creating tables, forms, reports, and labels. The Experts are similar to the Coaches found in PerfectOffice and the Wizards found in Microsoft Excel.

Visual dBASE includes Experts for tables, forms, reports, and labels. Unlike traditional Experts or Wizards, the new Form and Table Experts can be customized to fit your organization. System administrators and team leaders can setup the Experts to use the layout schemes and table designs specific to your needs.

Along with the new Experts and enhancements to the Form Expert, there are some new features common to all the Experts. The menu and SpeedBar options for opening Experts differ in *Visual* dBASE. There are also new prompting options in the Experts.

The new menu and SpeedBar streamline the options for opening an Expert. dBASE 5.0 included two menu options and two SpeedBar buttons for opening the Form Expert. Now all Experts are available exclusively from the File | New menu option and the corresponding SpeedBar button.

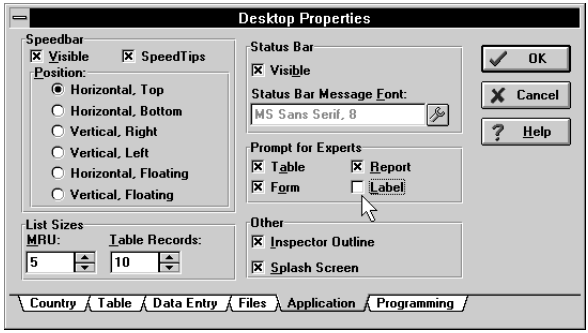
Figure 4.1 Prompting for the label expert

When you create a new file through the Navigator, a Catalog, the Menu, or the SpeedBar dBASE prompts if you want to use the Expert or not. Figure 4.1 shows an example of the prompt for the label expert.

The language provides a corresponding PROMPT option for the CREATE commands. The PROMPT option works with the EXPERT option. Here are a few examples of creating a new label.

```
CREATE LABEL               && opens the label designer
CREATE LABEL EXPERT        && opens the label expert
CREATE LABEL EXPERT PROMPT && gives option of expert and designer
CREATE LABEL PROMPT        && names a new label "PROMPT"
```

Figure 4.2 Selecting when to prompt for Expert



By default, dBASE always includes the EXPERT PROMPT option when creating new files. You can selectively turn off prompting through the Desktop Properties as shown in Figure 4.2. The dialog is available from the menu by selecting Properties | Desktop Properties.

The Table Expert

The Table Expert allows you to mix and match fields from example tables to create your own table. The resulting table can be any type that is currently available to dBASE through BDE.

You can configure BDE to work with Oracle, Sybase, Informix, and Microsoft SQL Server through native SQL-Link drivers. dBASE also works with ODBC drivers through BDE. If you have installed ODBC drivers as BDE alias names, the associated table types become available to the Table Expert.

Using the Table Expert

There are several ways to invoke Experts when prompting is checked in the Desktop Properties. Choose the method that fits your work style.

- From the Menu, select File | New and the desired file type.
- Click on the New file SpeedBar button and pick the desired file type from the popup menu.
- From the Navigator or a Catalog, double click on (Untitled).
- From the Navigator or Catalog, pick (Untitled) and select New file from the SpeedMenu or the Navigator/Catalog menu. You can also use the Shift-F2 accelerator key.

Note If the chosen file type is not appropriate for an Expert, such as Program or Popup, dBASE takes you directly to the design surface or editor.

The Table Expert has two steps. In the first step you select the fields for the table. Use the following buttons to select fields:

> Select the currently highlighted field

>> Select all fields

<< Deselect all fields

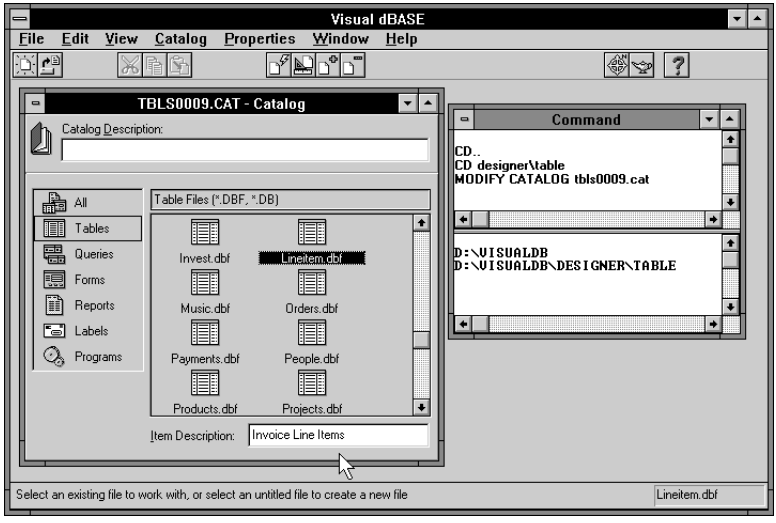
< Deselect the currently highlighted field

The second step lets you pick the table type you want to create. The default type is the dBASE DBF format. Once you reach the last step, you can start adding new records or switch into the table designer to continue refining the new table.

Customizing the Table Expert

The Table Expert works with a Catalog to show example tables and fields. You can add, modify, and remove tables from the catalog. The catalog and example tables reside in the \DESIGNER\TABLE subdirectory.

Figure 4.3 Working with Table Expert Catalog



The US version of the catalog is TBLS0009.CAT. The number 9 refers to the Windows country code and will vary for different localized versions of dBASE. You can use the MODIFY CATALOG command to open and customize the example tables.

```
MODIFY CATALOG tbls0009.cat && open the Table Expert catalog
```

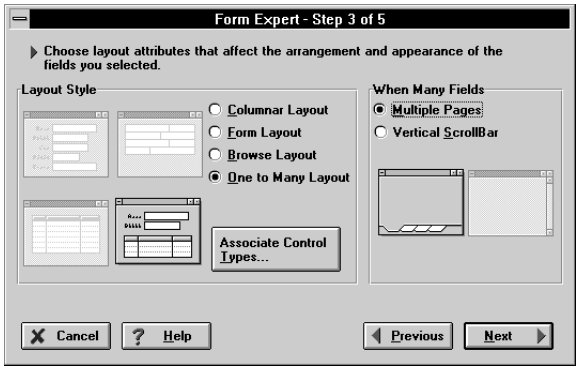
The Form Expert

The new Form Expert contains five steps. In the first step you select a view. In the second, you select fields from the view. The third step determines the form layout and the fourth step controls the font and color selections. The last step of every Expert prompts you if you want to switch to run or design mode.

While the first two steps of the Form Expert are almost identical to those found in dBASE 5.0, the next steps demonstrate significant enhancements. From the third step you can now choose what controls go with what field types. For instance you can select an EntryField or a SpinBox to handle date values.

The third step also gives you the option of creating multiple page forms if there are too many fields to fit in a single form. Not only is this useful for complex views, but it is a great way to learn how multiple page forms work.

Figure 4.4 Creating a multiple page form



The fourth step of the Form Expert lets you select and create form schemes. The schemes determine what colors and fonts apply to different areas of the form. You also create and apply schemes inside the Form Designer.

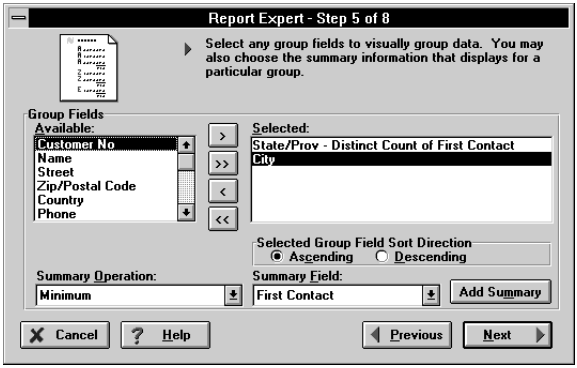
Figure 4.5 Creating schemes in the Form Expert

As with the Table Expert, the Form Expert stores attributes in a dBASE table. The schemes are in SCHS0009.DBF under the \DESIGNER\FORM subdirectory. Once again the number 9 is for US versions and will change to match the country code for the version of dBASE you are using.

The Report Expert

With the Report Expert, you can create a variety of reports from views and add calculated fields without ever opening Crystal Reports. The different report types fall into two main categories: detail and summary.

Figure 4.6 Creating summary groups in the Report Expert



As with the Form Expert, you begin by selecting a view. Once you have the view established you can create a detail or summary report. The steps vary for the different report types. The steps guide you through selecting fields, sort order, summary groups, grand total information, layout styles, and custom headings.

You can create calculated fields as you select fields for printing. Within summary groups, the Report Expert supports a wide variety of group statistics including sum, average, maximum, minimum, count, sample variance, sample standard deviation, population variance, population standard deviation, and distinct count. You can also include grand total calculations for any fields. There are tabular and columnar layouts with a variety of page break options.

The Label Expert

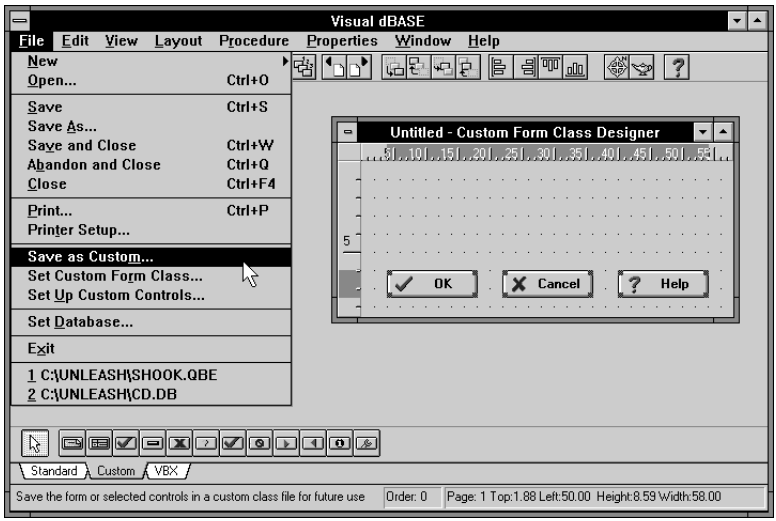
Now you can create labels quicker than ever with the five step Label Expert. In the first step you pick your view or table. In the second step, you can select and arrange the fields. The second step also contains a Quick Address button that automatically scans the view for fields such as "NAME", "STREET", and "ZIP" and arranges them for an address label. Like the Report Designer, you can add calculated fields and select the sort order. The Label Expert supports over 45 of the most common Avery label formats. As with all other Experts, the last step lets you move into run or design mode.

Custom Form Class Designer

Custom form classes serve as templates for new forms. This capability exists in dBASE 5.0 from the language, but not from the Form Designer. In dBASE 5.0 the Form Designer always works with the stock FORM class. In *Visual* dBASE you can substitute any custom form class in place of the stock FORM class. You can save an existing form as a custom form class and use the new Custom Form Class Designer to create and modify base form sets or collections of custom form classes.

As you start designing many forms for an application, it quickly becomes apparent that many of the forms share similar characteristics. For instance, you might create a set of navigation buttons on each data entry form and a set of OK, Cancel and Help buttons on each modal form. The repetitive groups of controls and properties are good candidates for custom form classes. You can create a custom form class for data entry that contains navigation buttons and transaction logic. You can reuse the custom form class for any data entry form. Likewise you can create a custom form class for modal dialogs.

Figure 4.7 Creating a custom form class for modal dialogs



Custom form classes give you a starting point other than a blank form. Unlike ordinary templates, a custom form class is dynamically inherited at run time. This allows changes to the custom form class to ripple down to all derived forms. You can design a custom form class with record navigation buttons and later add buttons for transaction processing. The new transaction buttons automatically appear on all derived forms.

Creating a new custom form class is much like creating any other new form. The main difference is that you will want to place only reusable components into the custom form class. Leave any controls, views, or DataLinks that are form-specific out of the custom form class. Follow the steps below to create a new custom form class for modal dialogs.

1 Open the Custom Form Class Designer. From the menu, select File | New | Custom Form Class.

2 Clear any set custom form class. From the menu, select File | Set Custom Form Class. If you are already using a custom form class, click on the Clear Custom Form Class button.

3 Add the OK, Cancel and Help buttons that are common to modal dialogs. You can use custom controls or add and configure your own PushButtons

4 After you have your custom form class designed, select File | Save as Custom...

5 The Save Custom dialog works with both Custom Controls and Custom Forms. Choose the Save Form as Custom radio button.

6 Name your custom form class and specify the CFM library that will contain the source code definition. You can use CFM files to create base form sets consisting of many custom form classes. CFM files are similar to WFM files in that they work as Two-Way-Tools. The source code in a CFM is fully modifiable and dBASE always uses the latest changes. Unlike WFM files, CFM files do not contain header information or code to create an instance of a form.

7 After naming your new custom form class as shown in Figure 4.8, click OK to save it to the custom form file. If the custom form file does not exist, dBASE creates it. If the custom form class already exists in the custom form file, dBASE asks if you want to overwrite the old definition. This only overwrites the current class and leaves all other classes in the file untouched.

Figure 4.8 Saving a custom form class to a base form library

When you design a custom form class, you do not need to create a standard form or WFM file. The complete source code for the custom form class is in the CFM file.

CFM files appear as auxiliary form files in the Navigator. To see auxilliary form files in the Navigator, select Properties | Navigator | Properties, and check the Show auxiliary form files option. You can also examine and modify the CFM using the program editor invoked with MODIFY COMMAND.

If you work with the source code for a CFM, be sure to leave the CUSTOM option on the class definition. The CUSTOM option insures that properties, controls and events do not stream out in a derived class.

Form Designer enhancements

The Form Designer is now a dual purpose work surface. As before you can use it to create forms. In addition, the Form Designer is a visual class designer for custom form classes and custom controls. Along with this powerful addition there are many other enhancements including new palettes, multiple pages, easier ways to share data between forms, improved visual property builders, a tab order dialog, and sizing options.

In dBASE 5.0, the Form Designer surfaces one aspect of visual component reuse. It allows you to use custom controls. *Visual* dBASE completes the object model with the following four distinct operations of visual inheritance and subclassing.

- Custom Form Class Creation
- Custom Control Creation
- Custom Form Class Inheritance
- Custom Control Inheritance

This section assumes you are familiar with the dBASE 5.0 Form Designer. You might want to review Part III of the dBASE 5.0 User's Guide before continuing. In particular, you should know how to work with the Inspector, Procedure Editor, and Control Windows.

If you are unfamiliar with the term Inspector, it refers to the window that shows properties, events and methods. The window is available in the Form Designer and through the INSPECT( ) function. dBASE 5.0 uses the terms Properties and Object Properties to refer to the Inspector.

Working with custom form classes

New forms can inherit properties, events and controls from a custom form class. For example, you can create a custom form class that colors the form red and contains a CheckBox control. You can set the custom form class and design new forms that inherit both the color and the control. Any later changes to the custom form class ripple down through all derived forms.

Saving forms as custom form classes

The Form Designer can save any form to a base form set. If you discover that one of your forms would work well as a custom form class, you can save it to a CFM file. To save a form as a custom form class, open it with the Form Designer and select File | Save as Custom. When you save a form to a base form set, dBASE leaves the original WFM file unchanged. If you want to modify the custom form class or create a new custom Form class, use the Custom Form Class Designer.

Using a custom form class

If you have created a custom form class, you can use it for new forms. When you work with a custom form class, dBASE generates different code for the class definition. Without a custom form class, new forms are derived from the stock form class. The following code shows what is generated for a form named ASK.WFM when no custom form class is set.

```
CLASS ASKFORM OF FORM
```

If you set the base class of the same form to THREEBUTTON that is stored in FORMLIB.CFM, the Form Designer generates different code. It replaces the stock class FORM with the custom base class THREEBUTTON. The CLASS command also contains a FROM clause that points to the source file where the definition for the custom form class resides. Here is the code for ASK.WFM when it is set to THREEBUTTON.

```
CLASS ASKFORM OF THREEBUTTON From FORMLIB.CFM
```

Controls, properties, and events contained within custom form class do not stream out into derived classes. The DEFINE statements for any control in a custom form class are not included in the source code of a derived form.

Figure 4.9 Setting a custom form class

You can apply a custom form class to new forms and existing forms. All you need to do to apply a custom form class is complete the Set Custom Form Class dialog shown in Figure 4.9. Follow these steps to create a new form that uses the THREEBUTTON form class created in the prior section.

1. Create a new blank form. Open the Form Designer without using the Form Expert.
2. From the menu, select File | Set Custom Form Class...
3. Enter FORMLIB.CFM for the source file name. You can also use the tool button to locate your CFM files.
4. Choose THREEBUTTON from the class name ComboBox. The ComboBox lists all the classes in the CFM file. If you created the CFM in the previous section, there is only one available class name.
5. Click OK to close the dialog and apply the new custom form class.

Now you can add whatever other controls you need to create a specific instance of the three button dialog. To make a login dialog, you could add an entry field and some text controls to explain the dialog.

If you select any of the inherited controls, the nibs appear white. This provides an easy way to distinguish inherited controls from controls defined in the WFM. Normally the selection nibs appear in black.

The Form Expert always uses the stock FORM class. To apply a custom form class to a form created by the Form Expert, simply open it in the form designer and set the custom form class. When you save the WFM, dBASE creates a derived form from the Expert generated form.

Working with dBASE custom controls

dBASE custom controls are controls derived from dBASE stock controls or VBX controls. The definitions for dBASE custom controls reside in CC files. You can use custom controls to create reusable components that encapsulate business rules and common application routines. Common examples of custom controls are push buttons for table navigation or closing a form.

You might be familiar with the BUTTONS.CC file from dBASE 5.0. While you can still use custom controls from BUTTONS.CC, you can now visually create your own custom controls and place them in the Control Palette without leaving the Form Designer.

Creating custom controls

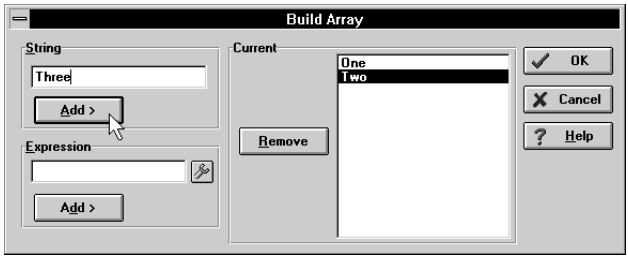
The Form Designer lets you design custom controls the same way you design any control. Simply place a stock or VBX control on the Form Designer and modify the properties. Once you have it complete, you can save it to a CC file.

You can derive custom controls from any stock or VBX control. For example, you might want to create a custom TabBox for three page forms. Follow the steps below to create a new custom control for navigating between pages.

1. Create a new blank form. If you have set a custom form class, clear it using the Set Custom Form Class dialog.
2. From the Control Palette, add a TabBox control. The TabBox automatically anchors to the bottom of the form.
3. Select the TabBox control and open the Inspector. You can quickly get to the Inspector from the Form Designer SpeedMenu whether or not it was previously open.
4. From the Inspector, locate the DataSource property and click on the tool button to open the DataSource Property Builder. This shows the default array containing "TABBOX1".

5 Open the Array Builder by clicking on the tool button. The Array Builder will let you quickly design an array containing "One", "Two" and "Three".

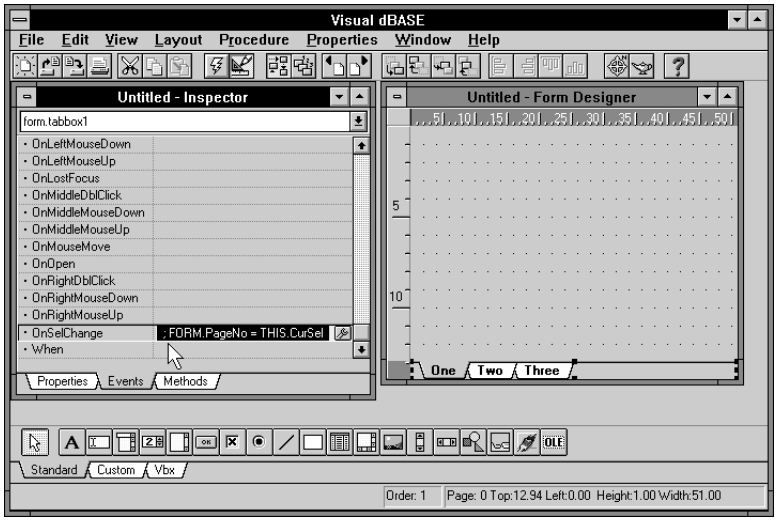
Figure 4.10 Building an array for a custom TabBox



6 From the Visual Array Builder, remove "TABBOX1". The Remove button deletes the currently selected item from the array.

7 Use the String entry field to add "One", "Two" and "Three". You do not need to enter quotes when adding strings. Simply type the number into the String entry field and click on Add as shown in Figure 4.10. The three tabs appear when you close the Visual Property Builders and return to the Inspector.

Figure 4.11 Setting the OnSelChange event for paging



8 Once you have the three numbered tabs you are ready to add an event that navigates between the three pages. Find the OnSelChange event for the TabBox.

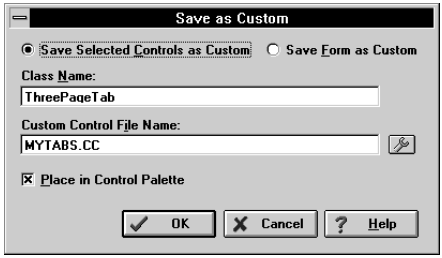
9 Add a code block to set the form's PageNo property to the current tab. The current tab is equal to the CurSel property of the TabBox. Enter the following code block into the Inspector as shown in Figure 4.11. Be sure to include the initial semicolon to distinguish the code block as an assignment statement.

```
; FORM.PageNo = THIS.CurSel
```

10 The last property to change is the Name property. The name of the control becomes the default class name. Locate the Name property in the Identification Properties group and change it from "TABBOX1" to "THREEPAGETAB".

11 After changing the Name property, select File | Save Custom... from the menu. The Save Custom dialog opens as shown in Figure 4.12.

Figure 4.12 Saving custom controls



12 Using the "Save Selected Controls as Custom" option, check that the class name is THREEPAGETAB and enter MYTABS.CC as the custom control file name. Once you have given the control a name and a file to save to, you can close the dialog.

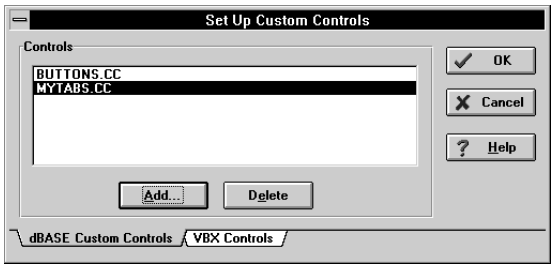
When you create new custom controls you do not need to save the WFM. After dBASE saves the CC file, you can exit the Form Designer without saving any changes.

Now you have a custom control for setting up three page forms. You can examine the code by editing MYTABS.CC in the program editor. Using the Navigator, you will find CC files listed under the Custom view.

Using a custom control

You can place custom controls on the Control Palette and use them as stock controls. By default, your custom controls appear on the custom tab of the control palette. The palette also has on option of creating your own tabs for custom controls.

Figure 4.13 Adding custom controls



dBASE 5.0 included an option to add new CC files to the control palette. *Visual* dBASE expands this feature by allowing to you examine, add and remove CC files and VBX files from the control palette. Follow the steps below to see how to add the custom tab control from the previous section into the control palette.

1 Open the form designer with a blank form. Start with a blank form to insure that it does not already contain a TabBox.

2 From the menu, select File | Setup custom controls.

3 Use the Add button to select your new dBASE custom control file, MYTABS.CC.

4 When you close the dialog, all the controls defined in MYTABS.CC appear on the custom page of the Control Palette.

5 If you are using the Control Palette in bitmap only mode, place the mouse pointer over the TabBox bitmap on the custom page to see the name of your custom control. The name of your control appears as the Text and SpeedTip for the custom control. The default bitmap is the bitmap for the stock class or VBX from which the new custom control is derived.

You can add the ThreePageTab to any form for easy navigation between three pages. When you use a custom control in a form, the Form Designer generates a DEFINE command for your custom class. The definition only includes properties that you have modified. If you are using all the properties and events as they appear in the custom control, the definition will contain no properties. Here is an example using the ThreePageTab custom control:

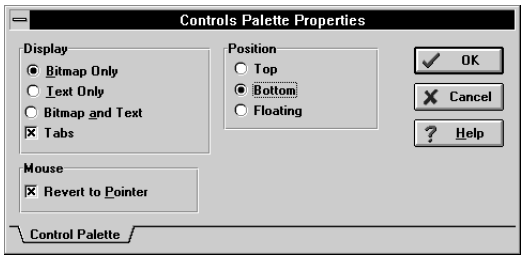
```
DEFINE THREEPAGETAB THREEPAGETAB1 OF THIS
```

Custom controls require that the source CC file be open during control definition. dBASE opens CC files that appear in the [CustomControl] section of the initialization file. You can also open CC files using the SET PROCEDURE statement.

The new palettes

Visual dBASE provides two new palettes that make form design easier and more customizable. One palette contains stock, custom, and VBX controls. The other palette contains a control for each field in the current view.

Figure 4.14 Setting palette properties



You can dock either palette at the top or bottom of the designer or use them as floating MDI windows. There is an option to turn off tabs, conserving screen real estate.

Visual dBASE provides two ways of configuring the palettes. You can use the Properties dialog and select the position as shown in Figure 4.14. The palettes can also be dynamically docked by moving the window to the top or bottom of the frame window.

You can make the palettes float by dragging them away from the dBASE frame window.

The Control Palette

In dBASE 5.0, the control palette consists of a ListBox in a floating window. The control palette in *Visual* dBASE is a series of buttons that can be docked as a SpeedBar or remain a floating window.

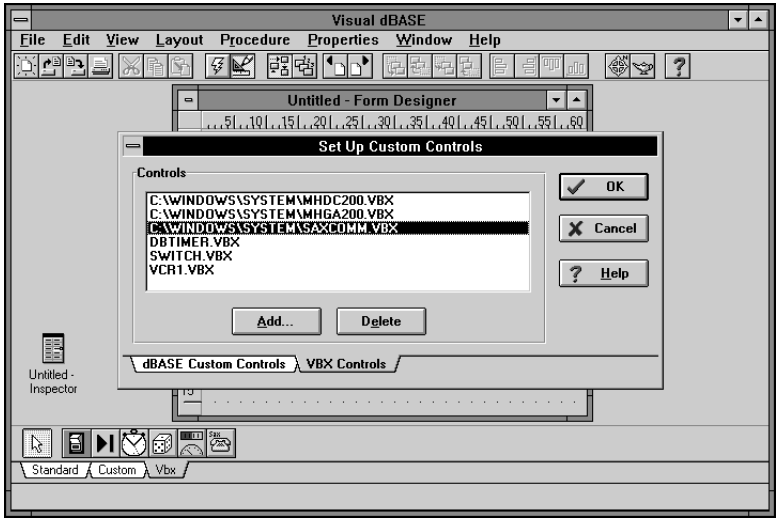
Most users find the control palette useful when it is docked at the bottom with tabs and displaying the bitmap only. Docking the palette on the bottom gives you a larger work area and keeps the top of the work area consistent in both run and design modes.

The control palette separates buttons into categories and groups. Categories can appear on different tabbed pages, while groups can appear on the same page. The default categories are Standard, Custom, and VBX. The default groups for each page include one group for the pointer and one group for all the other controls in the category.

Setting up VBX controls

You can use the same dialog to setup VBX controls that you use to setup dBASE custom controls. When you are in the Form Designer, you can open the Set Up Custom Control dialog by choosing File | Setup Custom Controls...

Figure 4.15 Adding VBX controls from the Visual Solutions Pack

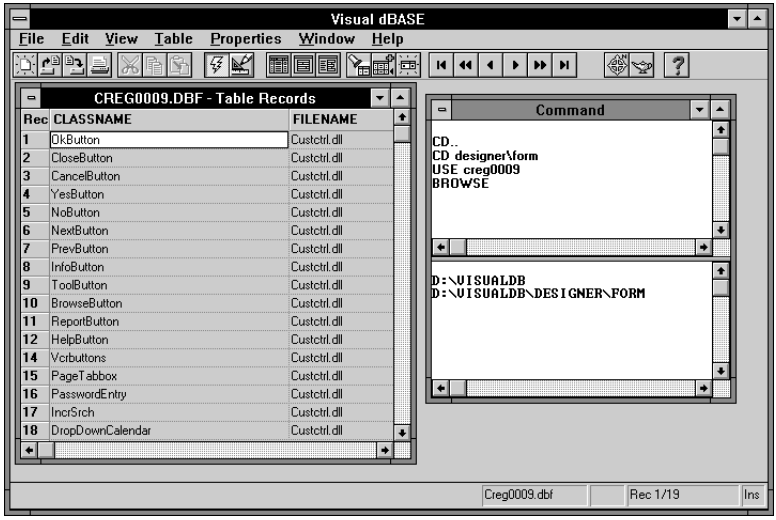


dBASE comes with a few VBX controls, but you can add many more. The Borland Visual Solutions Pack comes with a wide variety of VBX controls including a rich text editor, an Excel 4.0 compatible spreadsheet, charts, and a variety of gauges. The Visual Solutions Pack installs controls into the WINDOWS\SYSTEM directory. Figure 4.15 shows a setup screen that contains sample VBX controls that come with dBASE as well as three controls from the Visual Solutions Pack.

The Custom Control Registry

You can assign bitmaps, categories, and groups to your custom controls using the Custom Control Registry. The registry is a table located in the \DESIGNER\FORM subdirectory. The registry is named CREG0009.DBF where the number 9 is dependent on the localized version of dBASE.

Figure 4.16 Browsing the Custom Control Registry



Each record corresponds to a custom control. The class name field refers to the custom class. The rest of the fields determine how the control appears in the palette. The Filename field refers to a DLL that contains the bitmap for the control. In this case, Filename has no relation to the CC file that contains the control definition. Here is the complete structure for the control registry.

```
Structure for table   CREG0009.DBF
Table type            DBASE
Number of records     25
Last update           06/25/95
---------------------------------------------------------------
Field  Field Name                Type          Length  Dec  Index
    1  CLASSNAME                 CHARACTER         32           N
    2  FILENAME                  CHARACTER         80           N
    3  UPBMPID                   NUMERIC            5           N
    4  DOWNBMPID                 NUMERIC            5           N
    5  UPXOFF                    NUMERIC            5           N
    6  UPYOFF                    NUMERIC            5           N
    7  DOWNXOFF                  NUMERIC            5           N
    8  DOWNYOFF                  NUMERIC            5           N
    9  WIDTH                     NUMERIC            5           N
   10  HEIGHT                    NUMERIC            5           N
   11  BORDER                    LOGICAL            1           N
   12  CATEGORY                  CHARACTER         32           N
   13  GROUP                     NUMERIC            2           N
---------------------------------------------------------------
** Total **                                      188
```

If you want to use custom bitmaps for a control, you should create two bitmaps in a DLL. One to display when the control is not selected or "up" and another for when it is selected or "down". You can use Resource Workshop to create the controls. Resource Workshop is available with Borland C++.

To conserve resources, you can store more than one glyph or image in a bitmap resource. If you combine bitmaps, use the UPXOFF, UPYOFF, DOWNXOFF, and DOWNYOFF to distinguish each glyph. You can ignore these fields if you store each glyph in a separate bitmap resource.

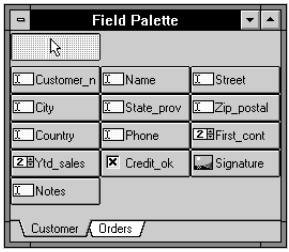
The Category determines which tab the control appears on. If the category is blank, the control displays in the Custom tab. You can use numeric values in the Group field to create separate groups within a category. If you leave the group blank, the controls in the same category appear as a single group.

The Field Palette

You can use the Field Palette to quickly add data aware controls to a form without manually setting the DataLink property. The Field Palette reflects the current view. Any fields in the view appear as controls in the Field Palette. The categories of the Field Palette correspond to tables in the current view. When using page tabs, the fields appear on the page with the associated table name. If the view only contains one table, the Field Palette contains only one page.

You can add fields to a form from the Field Palette the same way you add controls from the Control Palette. Simply drag and drop fields from the palette onto the form. Once the field is on the form, you can use the Inspector to customize it.

Figure 4.17 Two tables in the Field Palette

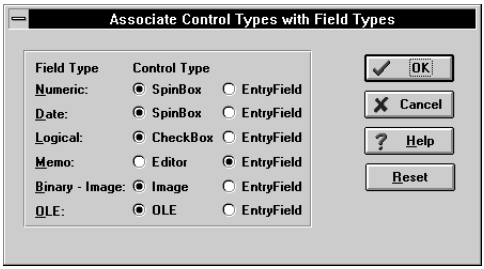


If you open the Field Palette without setting a view, it appears blank except for the pointer button. When you change the view, the Field Palette refreshes to show the new fields.

While you normally want to dock the Control Palette, most developers find that the Field Palette works better in a floating window. The Field Palette is useful when you begin designing a form. Once you have all the data controls on the form, you can clear the work area by closing the Field Palette.

The glyph next to each field shows you what control will appear for that field. For example, the binary Signature field appears in an image control and Credit_OK will appear as a CheckBox. The field type determines the control. In this case the CheckBox appears for logical fields and the Image appears for binary image fields.

Figure 4.18 Customizing the Field Palette



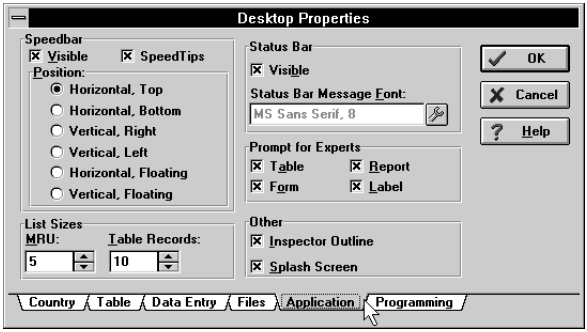
You can customize the Field Palette to use different controls for the various field types. All the field types can work with EntryFields. Many field types can also work with specialized controls such as the SpinBox and the Editor. The Field Palette always associates character fields with EntryFields.

To change the associations, select Layout | Associate Control Types with Field Types. The associations control the Form Expert as well as the Field Palette. You can also open the associations dialog shown in Figure 4.18 from the Form Expert.

Creating multiple page forms

Forms now contain pages for breaking complex forms into a group of logical pages. Pages give forms depth, where each page is a layer. The top layer is page one. By default forms open at page one. For simple forms or dialogs you can simply use page one. For forms with many controls, you can use as many as 255 pages.

Figure 4.19 dBASE dialog with six pages



dBASE contains examples of single and multiple page dialogs that you can use as a guide for designing your own forms. The dialog for Field Palette Properties contains a few controls that exist on a single page, while the dialog for Desktop Properties has many controls separated into six logical pages. dBASE uses the TabBox to navigate between pages.

While it is common to use a TabBox to move between pages, the TabBox is not tied specifically to paging. The page implementation is extremely flexible. Any event on any control can change the current page number.

Forms and controls include a PageNo property. The PageNo of the form determines the current set of controls. The PageNo of a control associates it with a page. For instance, if the current form PageNo is 1, all controls that have a PageNo of 1 appear. If you change the form PageNo property to 2, the controls with PageNo of 1 disappear and controls with a PageNo of 2 appear.

Moving between pages at design time

When you are working in the Form Designer, you navigate between the pages to drop controls. For example, you could drop some fields on page 1, switch to page 2 and drop some more. The designer gives you several ways to quickly move between pages.

- Using the keyboard, press *PgUp* and *PgDn*
- From the menu, select View | Previous Form Page or View | Next Form Page
- Using the mouse, click on the page navigation SpeedBar buttons.

For forms with many pages, you can use the View | Go to Form Page Number. This opens a dialog for navigating to a specific page. All the Navigation options change the Form's PageNo property.

The best way to see how multiple page forms work is to create one. Use the following example to create a multiple page form:

1 Create a new blank form.

2 Set the View property to the sample table CUSTOMER.DBF.

3 From the menu, select View | Field Palette.

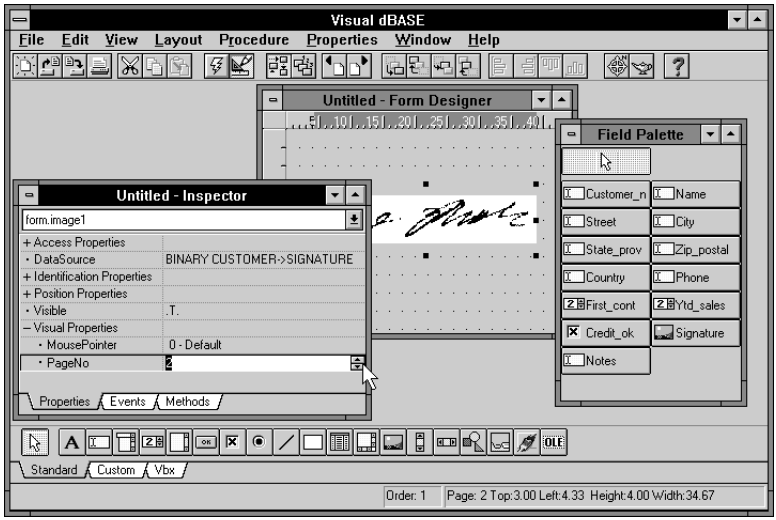
4 Drag and drop the Name, Street and City fields onto the form. Position the Name field near the top of the form. Resize the fields as needed.

5 Press *PgDn* to move to page 2.

6 Drag and drop the Signature field onto the form. Resize the signature image to leave some blank space at the top of the form. After placing the signature field, the form should appear similar to Figure 4.20.

7 Close the Field Palette.

Figure 4.20 Adding the Signature Field to page 2



8 Use the *PgUp* and *PgDn* keys to move between the pages.

The current page appears in the status bar. If you move to page zero you will see controls from both pages.

Page zero

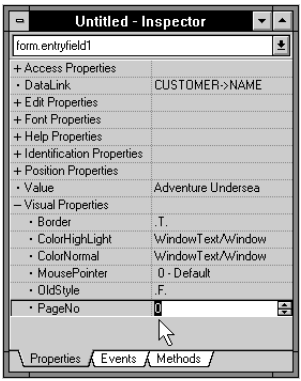
Forms contain a special page for controls that you want to appear on every page. If you examine the Desktop Properties dialog in Figure 4.19, you will find four controls that appear on every page. The TabBox for navigating pages and the OK, Cancel, and Help push buttons appear on every page. You can achieve a similar effect using page zero.

If you press *PgUp* in the Form Designer when your are on page 1 you will end up at page zero. If you switch the Form's PageNo property to zero, all controls appear. This normally causes controls to overlap.

In multiple page data entry forms, you can place key fields on page zero. The key fields help users keep track of the current record. Follow the steps to move the entry field to page zero.

1. From the menu, select View | Go to Form Page Number.
2. Enter "1" and click OK. This changes the Form's PageNo property to 1.
3. Select the top control, EntryField1. It has a DataLink to the Customer Name.
4. Open the Inspector, select View | Inspector.

Figure 4.21 Setting the PageNo to zero



5. For the EntryField1 control, set the PageNo to 0. The PageNo property appears in the Visual Properties group as shown in Figure 4.21.

The control will now appear on all pages. Switch between the pages using *PgUp* and *PgDn* to see how the custom name appears on every page.

At run time the *PgUp* and *PgDn* keys control record navigation. *PgUp* moves to the previous record, while *PgDn* moves to the next record. You need to add some other mechanism to change pages when a form is running. Continue to the next section to see how to add a page navigation TabBox.

Adding a control to move between pages at run time

Switching pages in the Form Designer changes the Form's PageNo property. To switch pages at run time you need a control that does the same thing.

You can accomplish this using any control or set of controls. For instance you could create a series of push buttons labeled one, two, three, etc. Each button could then set the PageNo property to a specific value in the OnClick Event. The most common way to move through pages is with a TabBox control. Use the following steps to complete the multiple page form example with a TabBox.

1 Go to Form Page zero.

2 Add a TabBox control. Be sure to use the standard TabBox control rather than a custom control.

3 Open the Inspector. From the menu, select View | Inspector.

4 Find the DataSource for the TabBox.

5 Set the DataSource property to an array containing "Address" and "Signature". You open Visual Property Builders using the tool button or enter the following directly in the Inspector. In either case be sure to remove the default value "TABBOX1" from the array.

```
ARRAY {"Address","Signature"}
```

6 Locate the OnSelChange event for the TabBox.

Figure 4.22 Adding a page navigation TabBox

7 Enter the following code block. Be sure to include a leading semicolon. This code block sets the form's PageNo property to a number corresponding to the selected tab.

```
;  FORM.PageNo = THIS.CurSel
```

8 If you have the sample custom control library BUTTONS.CC loaded add a custom Close button as shown in Figure 4.22. Place the close button on page zero.

9 Switch back to page one before saving your form. When you save a form, dBASE remembers the current page number and uses it as the active page at run time. If you save the form while on page zero, all the controls will appear when you run the form.

The form is now complete. You can press F2 to save and run your form. If you want to add more pages simply drop the controls on them. For each new page, add one element to the TabBox DataSource array. Any element you add to the array becomes a new tab.

Working with more than one form

Visual dBASE makes it easy to create applications with many forms. When you open more than one form you can provide each form with a separate data session or let them share a view. Any form can also open as a modeless window or a modal dialog. The CREATE SESSION command and the form's View property give forms an independent data environment. The form's new DesignView property helps you design forms that share data. When you save a form, the designer includes an optional parameter to open the form as a dialog.

Using sessions to create independent data environments

The default behavior of the dBASE Navigator opens forms in separate sessions. As one form opens and closes tables, the other forms are not affected. This behavior is desirable for forms that do not use the same tables or views. It lets the user switch between forms as separate applications.

Many applications use a main form or menu to open other forms. You can open additional forms in any event. The two most common events are the menu and push button OnClick. In either case, opening an additional form involves three steps as shown in the following code.

```
Procedure PUSHBUTTON1_OnClick
   CREATE SESSION  && step 1 - make a session
   SET TALK OFF    && step 2 - session specific settings
   DO form2.wfm    && step 3 - open the form
```

The first step is to create a session. The session must exist before creating an instance of the form. A Form is permanently attached to the session which is active when the form is created. If you define a form prior to creating a session, they are not linked.

The second step is to make any session specific settings. Creating a session is similar to starting another copy of dBASE. Each new session starts with a fresh environment. Settings that applied to the previous session, do not apply to the current session. Even though TALK was OFF in the prior session, it will be ON by default in the new session.

The third step is to open the form. To open it as a modeless window, simply DO the WFM file. If you do not include the WFM extension, dBASE will look for a PRG.

If you want a certain form to always open in a separate session, you can add the code for making a session and setting up the environment to the form's header section. Many of the sample forms that come with *Visual* dBASE use this technique.

Sharing the data environment between forms

If you have two forms that work with the same tables or views, you might want to keep them in the same session. This is useful for forms that provide different layouts for the same data. For instance, you might have a locate form that uses a browse control to quickly find records and an update form showing only one record to edit a table. In this case, you can let the two forms share the same view in the same session. This insures that moving the record pointer in the browse control on the first form also moves the record pointer in the second form.

To create view sharing forms it is helpful to select one form that will open first. In the case of the locate and update forms, the first one to open is the locate form. You can design and open the locate form as you would any other form. That is, you can set the View property and open it within a session.

For a second form, such as the update form, do not set the View property. Instead set the DesignView property. This provides a data environment in the Form Designer. DesignView lets you use the Field Palette to quickly add fields and the Visual Property Builders to set DataLink and DataSource properties.

If you set the View property for the second form and ran it in the same session as the first form, it would close and reopen the tables. That repositions the record pointer and is generally not what you want when sharing data between forms. The DesignView property only creates the view when you are in the Form Designer. It is up to you to make sure the appropriate data environment exists at run time.

To open the second form that will share data with the first form, simply run the WFM from an event on the first form without creating a session. Using the locate and update example, you might have an UpdateButton on the Locate form to open the Update form. The OnClick only needs a single line of code that can appear in a code block. Here is an example of calling a form called Update.WFM from the OnClick event of an UpdateButton.

```
Form.UpdateButton.OnClick = { ; DO Update.WFM }
```

Using a form for modal dialog

Forms can appear as modeless windows or modal dialogs. The default behavior for a form is to be a modeless window. The Navigator and Command Window are examples of modeless windows. You can click between them at any time. When a form appears as a dialog, you cannot switch away from it. Dialogs require some type of input and normally invoke some activity such as opening a table. Examples of modal dialogs are the Table Open dialog and the About box. If you select Help | About from the menu, a dialog appears and you cannot select the Navigator until you close the dialog.

Code saved by the Form Designer includes a new parameter for opening a form as a modal dialog. Simply pass a .T. to the WFM and the form will appear as modal dialog. For example:

```
DO myform.wfm WITH .T. && opens the form as a modal dialog.
DO myform.wfm         && opens the form as a modeless window.
```

Dialogs have a fixed height and width. Since a dialog cannot be resized, it cannot appear within an application's MDI space. The code in the WFM sets the MDI property to .F. (false) for all modal dialogs. Here is the header code that controls modal behavior in a WFM.

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

Note If you have forms created in dBASE 5.0, you can open them in the designer and select File | Save and Close. *Visual* dBASE will add the new parameter when it saves the form.

Other changes

Along with the major enhancements to the Form Designer, you will find some revisions that make it easier to arrange and modify objects. There are new options for sizing controls, applying font and color schemes, and a simpler way to rearrange the tab order. There is also a new windowing option.

There is a new Arrange Designer Windows option on the Window menu. Choose this option to quickly reposition the inspector, palettes, and editor so all controls are visible and logically organized.

Size

Four new sizing options let you resize groups of controls. The sizing options are available from the Layout menu. To use the size options, select a group of controls and pick one of the following options.

- Horizontal Grow To Largest
- Horizontal Shrink To Smallest
- Vertical Grow To Largest
- Vertical Shrink To Smallest

You can select a group of controls by clicking and dragging a lasso around all the controls you want in the group. This is the easiest method for controls that are next to each other. To group non-consecutive controls, hold down the shift key while clicking on the different controls.

The horizontal sizing options change the Width property while the vertical options change the Height property. The size options do not alter the top or left properties of any control. You can use the alignment menu options and SpeedBar buttons to adjust the Top and Left properties to relative coordinates.

Schemes

Another new option on the Layout menu is Set Scheme. This option lets you take advantage of the Form Expert's scheme step on existing forms. When you apply a scheme to an existing form, the dBASE categorizes the areas as follows:

- **Title** is a text control with the name "TITLE". The scheme also recognizes custom classes derived from the Text stock class.
- **Non-Editing Controls** include controls using or derived from the following stock classes: CheckBox, RadioButton, Rectangle, TabBox and Text.
- **Editing Controls** include controls using or derived from the following stock classes: Browse, ComboBox, Editor, EntryField, SpinBox and ListBox.
- **PushButtons** includes all PushButtons.
- **Shapes** includes shapes and lines. Neither of these controls can get focus or appear in the tab order.
- **Form** sets the background color of the form, the title and all non-editing controls.

Some controls do not fall into any scheme categories. The OLE and Image controls do not include font or color properties. dBASE does not draw colors or fonts into the area defined by PaintBox. VBX controls use their own default colors and fonts.

ScrollBar always defaults to using the relative ScrollBar color. Relative colors correspond to the current color scheme setup of the Windows environment. Users can change relative colors using the Windows Control Panel.

There are two ways to work with a Scheme. You can use the "Apply Now" push button to set all the controls on your current form to the colors and fonts of the scheme.

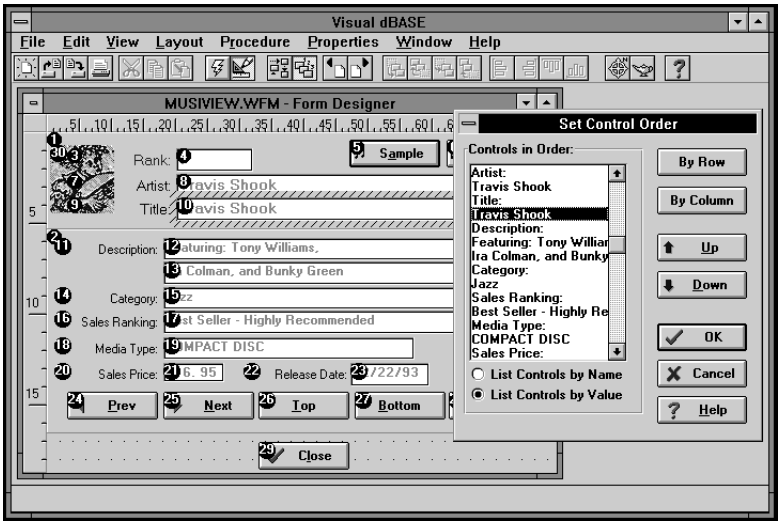
You can also leave your current form colors intact and select a scheme for new controls. In this mode, the controls already on your form keep their colors while field and control palettes take the fonts and colors of the scheme.

When you are working the Scheme designer, do not confuse OK with Apply Now. The OK button establishes the selected scheme for use with new controls only. If you want to change controls already on the form you must select Apply Now.

Set Order

Setting the tab order and Z-Order of controls is now easier than ever. The Z-Order refers to the order of overlapping controls. Tab order is a subset of Z-order, for controls that have a TabStop property. The *Visual* dBASE improves upon the order view of dBASE 5.0 and adds a new Set Order tool.

Figure 4.23 Using the Set Order Tool



The Set Order Tool shown in Figure 4.23 lets you reorder controls listed by name or value. To open the Set Order tool, select Layout | Set Control Order.

The new Order View now shows you the controls as they appear at run time. You can still order controls by clicking on the first control and clicking on all other controls in the desired order. In addition, you can reorder controls from any position. For instance, if you have twenty controls and want to rearrange the last two, simply Shift-Click on the third to the last control. The Shift-click establishes a relative starting point and allows you to reorder all remaining controls.

Popup menus

Visual dBASE includes a new fully object-oriented popup designer to complement the existing menu designer. Popups are generic menu objects that you can open on any event. The common use of a popup is a menu that appears when using the right mouse button. dBASE refers to such menus as SpeedMenus.

You can easily create your own SpeedMenus using the Popup Designer and the form's PopupMenu property. If you use the PopupMenu property, the popup appears whenever the user right clicks on a form.

If you need more precise control, you can position and open the popup on an event. The popup class contains top and left properties for positioning and an OPEN( ) method for activation. You can use top, left and OPEN( ) to create popups that appear at the control level rather than the form level.

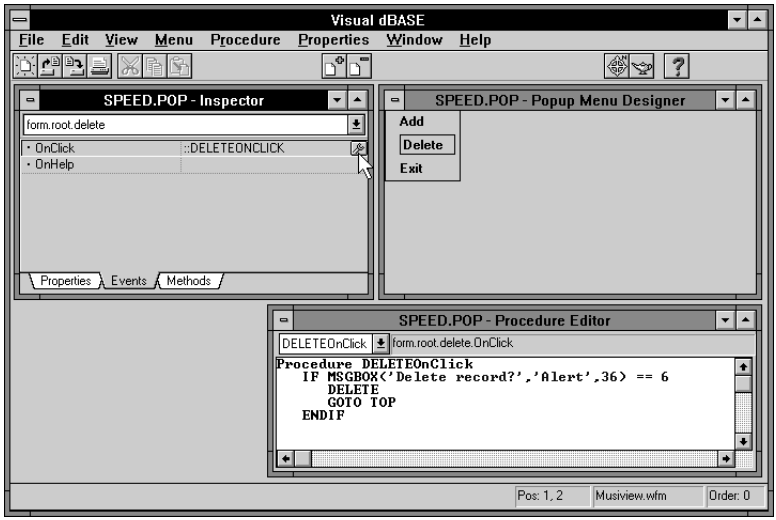
Using the Popup Menu Designer

The popup menu designer works much like the standard menu designer. The major difference is that the popup designer does not let you create a menu bar. Tabbing from the first menu item creates a pull-right. Follow the steps below to create a popup with options for adding records, deleting records, and closing a form.

1 Start creating your popup by invoking the popup designer. You can use the CREATE POPUP command or select File | New | Popup.

2 Create prompts for the SpeedMenu. You can enter the menu prompts directly onto the designer surface. Create three new prompts by entering "Add", "Delete" and "Close". Use the up and down arrow keys to navigate from prompt to prompt.

Figure 4.24 Creating a popup menu



3 Attach a method to the OnClick event of the "Add" menu. The Inspector and Procedure Editor are available in the popup and menu designers. Use the SpeedMenu to open the Inspector. From the ComboBox at the top of the Inspector, select form.root.add. Click on the "Events" tab to see OnClick event. Use the tool button on the OnClick event to open the Procedure Editor as shown in Figure 4.24. dBASE automatically generates the procedure statement and an appropriate name. You can modify the name if you wish, otherwise simply enter the APPEND BLANK command so the method reads:

```
Procedure ADDOnClick
  APPEND BLANK
```

To create an OnClick event for "Delete" that asks for a confirmation. Click on the "Delete" prompt in the menu designer. The Inspector switches to show the events page for form.root.delete. Click on the tool button next to OnClick and enter the method as follows:

```
Procedure DELETEOnClick
  IF MSGBOX('Delete record?','Alert',36) == 6
     DELETE
     GOTO TOP
  ENDIF
```

1. Finally add an OnClick event that will close the form. Click on the "Exit" prompt in the menu designer. The Inspector switches to show the events page for form.root.exit. Since forms serve as containers for popups and menus, events can refer directly to the form and controls within it. The CloseOnClick method should read:

```
Procedure EXITOnClick
  FORM.CLOSE()
```

2. After adding the events save the popup as SPEED.POP. From the menu select File | Save and Close or press Ctrl-W.

Popup menus appears in the custom view of the Navigator. You can right-click on a popup and select Edit as Program to see the source code. The Popup Designer is another Two-Way-Tool. The Popup Menu Designer recognizes modifications you make to the source code.

The PopupMenu property

After creating a popup menu, you can attach it to the form. You can create a SpeedMenu by setting a form's PopupMenu property to a popup. This is easily accomplished in the OnOpen event.

Note The PopupMenu property takes an object reference. Unlike MenuFile, you cannot set it to a file name. This allows you to set PopupMenu to any popup regardless of how you create or store it.

If you created the SPEED.POP file in the previous section, you can follow these steps to make it into a SpeedMenu. Before you can work with popup you need a form.

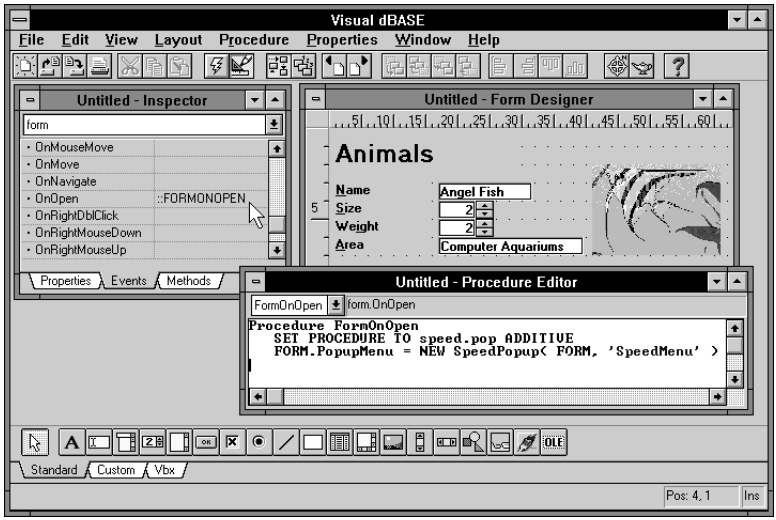
1. Create a new form using the Form Expert. Try using the dBASE language to start the expert. From the menu select Window | Command and enter:

```
CREATE FORM EXPERT
```

2. Choose the ANIMALS.DBF from the samples directory and click on Next.
3. Use >> to select all fields and click on Next.
4. Proceed through the rest of the Expert selecting any layout or scheme. When you get to the last step of the Expert, select Design Form to place your new form in design mode.

5 Right-click on the Form Designer window and select Inspector from the SpeedMenu.

Figure 4.25 Attaching a Popup to a Form



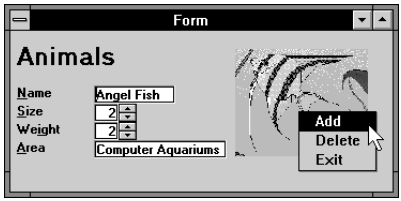
6 Click on the events tab and scroll down to the form's OnOpen event. When you find the OnOpen event, click on the tool button (wrench) to open the procedure editor as shown in Figure 4.25.

7 The Procedure Editor appears with a method heading. Since dBASE already named this method and linked it to the OnOpen event, all you need to do is enter what you want to happen during the event. Enter the following commands to create a SpeedMenu from the SPEED.POP file:

```
Procedure FormOnOpen
   SET PROCEDURE TO speed.pop ADDITIVE
   FORM.PopupMenu = NEW SpeedPopup( FORM, "SpeedMenu" )
```

8 When you finish writing the procedure, test your form. Press *F2* or select View | Form from the menu.

Figure 4.26 Using a Popup



Right-Click anywhere on your form to see the popup shown in Figure 4.26. The PopupMenu property gives dBASE the ability to automatically position the popup at the mouse pointer. If you pick Add or Delete, the appropriate event fires and all controls refresh after the event.

The MenuBar

Most Windows applications include an Edit menu that works with the clipboard and a Window menu for moving between MDI windows. The Menu Designer eases the creation of Edit and Window menus with the MenuBar class. Like Popup, a MenuBar serves as the root object in a menu.

All other objects in the menu tree are from the Menu class. The MenuBar class includes the following properties for setting up standard Edit and Window menus.

- **EditCopyMenu** gives a menu the action of copying selected text to the clipboard. It also enables the menu when something is selected.
- **EditCutMenu** gives a menu the action of moving selected text to the clipboard. Like EditCopyMenu, it only enables the menu if something is selected.
- **EditPasteMenu** gives a menu the action of pasting text from the clipboard into the current control. It enables the menu if something is in the clipboard and focus is on a control that accepts clipboard information.
- **EditUndoMenu** gives a menu the action of undoing the last clipboard operation.
- **WindowMenu** creates a menu item for each MDI window. Each new menu item has the action of setting focus to the selected window.

The MenuBar properties are similar to the form's PopupMenu property. You can set them to menu objects in the current menu tree. PopupMenu is normally setup in a form's OnOpen event. You can also set the MenuBar properties in the OnOpen event or encapsulate the action in the MenuBar OnInit event.

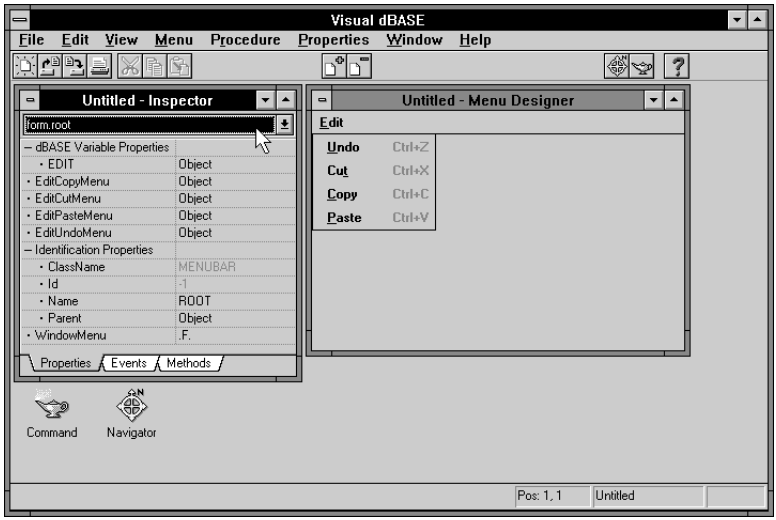
The MenuBar OnInit event fires when focus moves to any part of the menu tree. The OnInit event also provides a central location to the set menu Checked and Enabled properties.

Creating an Edit menu

The Menu Designer has a new option to create all four edit menus. This automates the process of creating the menus and setting the text and short-cut properties. Follow the steps to create a new menu file and add a complete Edit menu.

1. Create a new menu. You can select File | New | Menu, use the New SpeedButton or enter the CREATE MENU command.
2. Add an Edit menu by selecting Menu | Insert "Edit" Menu. This option is available if the cursor is in the menu bar. If you want to add an Edit menu to an existing menu, place the cursor where you want the menu to be. When you select Menu | Insert "Edit" Menu, dBASE will move the current menu and all following menus to the right.

Figure 4.27 Inspecting the MenuBar object



3 Inspect the MenuBar object. All visible menus are menu objects. The MenuBar Object is the parent of each top level menu. The reference to the MenuBar is FORM.ROOT. Open the Inspector with View | Inspector. From the ComboBox at the top of the Inspector, select FORM.ROOT.

4 Examine the properties of the MenuBar. See if the EditCopyMenu, EditCutMenu, EditPasteMenu and EditUndoMenu are set to objects.

5 Switch over to the Events page for the MenuBar. Try using the keyboard instead of the mouse. Press the Tab key to move to the page tabs and use the arrow keys to switch pages. The MenuBar only has one Event, OnInitMenu. You can designate MenuBar properties by placing the following code in the OnInitMenu event.

```
This.EditCopyMenu = This.Edit.Copy
This.EditCutMenu = This.Edit.Cut
This.EditPasteMenu = This.Edit.Paste
This.EditUndoMenu = This.Edit.Undo
```

6 When you finish designing the menu, save it as BAR1.MNU.

To test out the menu, modify any form that has editing controls so the MenuFile property is set to BAR1.MNU. When you run the form, your Edit menu appears and functions with any controls on the form.

Creating a Window menu

Adding a Window menu is similar to adding an Edit menu. The main difference is that the menu items appearing below the Edit menu are static and visible in the designer, while the items under the Window menu are dynamic and not visible in the designer. If

you created the BAR1.MNU file in the previous example, try adding a Window menu as shown below.

1. Open BAR1.MNU in the Menu Designer. You can locate it in the custom section of the navigator or in a file open dialog by selecting File | Open and choosing *.MNU as the File Type. Entering MODIFY MENU BAR1 in the Command window also works.
2. Press Tab to move the cursor to the right of the Edit menu. The standard menu layout always places the Window menu to the right of the Edit menu and that is where you will want to be.
3. From the menu, select Menu | Insert "Window" Menu.
4. Open the Inspector switch to FORM.ROOT to examine the MenuBar object.
5. See that the WindowMenu property is set to the new Window menu object. The following command sets the property for you.

   ```
   FORM.ROOT.WindowMenu = FORM.ROOT.Window
   ```

6. Save and close the menu.

Try using the menu with an MDI form. If you use it with a non-MDI form, the Window menu appears with no items under it. When you are setting up an application with multiple MDI windows and custom menus, it is a good idea to hide the Command and Navigator windows. This gives applications full control of the dBASE environment. To remove the Command and Navigator windows from the MDI space, add SHELL(.F.) to the form's OnOpen event.

Note WindowMenu, EditCopyMenu, EditCutMenu, EditPasteMenu, and EditUndoMenu are properties of the MenuBar class, not the Menu class.

SQL Statement Builder

The SQL Statement Builder is a utility for creating SQL statements and learning the differences between SQL and dBASE. SQL, Structured Query Language is the most common language for working with a relational database system. It only contains a few commands and all the commands involve data manipulation. DML (Data Manipulation Language) commands are restricted to data specific functions such as such relating and updating tables. The SQL Statement Builder lets you create SQL DML commands.

SQL does not include any commands for controlling program flow, the user interface, defining classes, etc. Other languages such as COBOL, C++, Object Pascal and dBASE provide a framework for SQL.

The dBASE Language contains statements such as SET RELATION, REPLACE and APPEND BLANK. *Visual* dBASE lets you mix and match SQL and dBASE DML statements in the same program. The Query Designer is a Two-Way-Tool that works with dBASE DML to create views and relations between tables. You can perform the same operations and update tables using the SQL Statement Builder. You can create .QBE files that contain SQL using the SQL Statement Builder.

Running the SQL Statement Builder

The setup program installs the SQL Statement Builder in the UTILS subdirectory. If you installed *Visual* dBASE to D:\VISUALDB, the SQL Statement Builder will reside in D:\VISUALDB\UTILS.

Note The SQL Statement Builder is only available if you choose a full installation or used the custom setup option to include 'Utilities and Custom Controls'. See chapter one for more information on the setup program.

Follow these steps to run SQL Statement Builder and retrieve data from the sample Animals table.

1 From the Navigator, select the utilities directory. You can also enter the following command in the Command window.

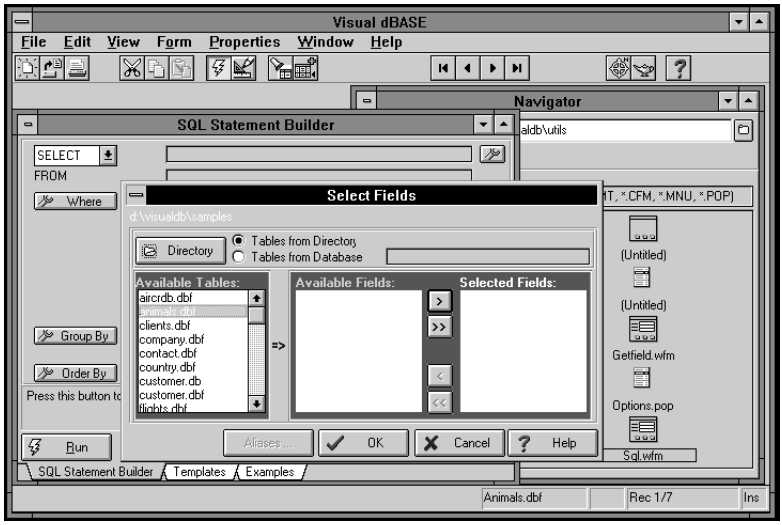
```
CD D:\VISUALDB\UTILS
```

2 From the menu select View | Forms. The SQL Statement Builder is a dBASE form named SQL.WFM. Double click on the form to start it. If your using the Command window, enter:

```
DO SQL.WFM
```

3 You can create four types of SQL statements: Select, Update, Insert and Delete. Regardless of what type you are building, you must begin by selecting a table. For now, try creating a Select statement. Click on the tool button to pick a table.

Figure 4.28 Using the SQL Statement Builder



4 Choose all fields from the Animals table. To pick the Animals table, use the directory button to navigate to SAMPLES directory. Highlight ANIMALS.DBF as shown in Figure 4.28. Click on >> to choose all fields.

5 After picking a table, you can visually create three additional clauses to the Select statement.

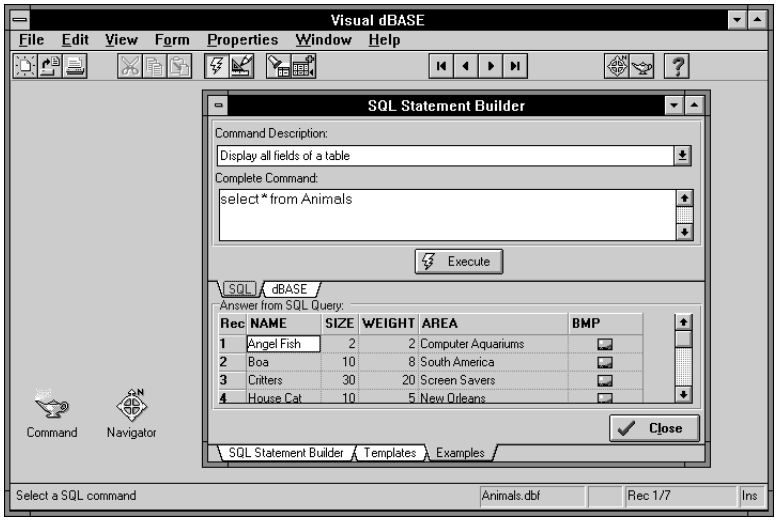
- **Where** lets you place a logical constraint on what rows appear in the table. This is similar to the SET FILTER command. dBASE can create updatable cursors regardless of whether you add a where clause or not. The clause for rows that have a size is greater than ten is:

  ```
  WHERE size > 10
  ```

- **Group By** summarizes your table. You can use group by with aggregate functions to get averages, sums and counts for any column. For instance you could get a count by area. Using the Group By clause is similar to the TOTAL ON command. When you use the Group By clause, dBASE creates an answer table.
- **Order By** places your table in a specific order. This is similar to setting an index or using the SET ORDER command. If no index exists, the behavior is more like the SORT TO command resulting in a read-only answer table.

After adding any additional clauses, you can run the query and browse the result. There is also a Show SQL button to view the SQL statement. If you want to use the statement in your program, select Show SQL and copy the statement to the clipboard. You can then paste it into your program.

Figure 4.29 Working with the SQL examples



SQL templates and examples

Use the Templates page and Examples page to learn more about SQL. Templates show how you can add visual query builders into your own applications. The examples let you compare and contrast SQL statements created by the SQL Statement Builder with the dBASE command created by the Query Designer.

Note See Chapter 8 for a reference guide to embedded SQL.

Chapter

5

Database administration

This chapter covers the data administration tools for local and remote tables. It should be read by database administrators who are responsible for referential integrity and data security.

The first section of this chapter describes how to create and modify referential integrity rules for Paradox tables and remote servers, such as InterBase, ORACLE, and Sybase. Topics include:

- Defining referential integrity
- Update and delete behavior
- Changing or deleting referential integrity rules

The second section shows how to plan out your security scheme. Topics include:

- The various levels of security
- An overview of the various aspects of the Protect feature
- Planning group access for each table
- Planning each user's login and user access level
- Planning user access to tables and fields within tables

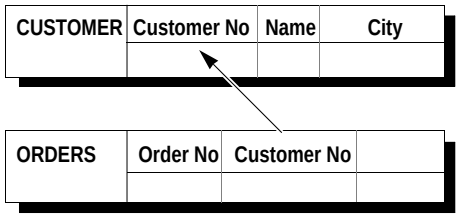
The last section of this chapter shows how to set up your security scheme using the Protect feature in *Visual* dBASE. It shows how to,

- Enter the database administrator's password
- Create user profiles
- Set user privileges for table access
- Set user privileges for fields within tables

Referential integrity

Referential integrity means that a field or group of fields in one table (the "child" table) must refer to the key of another table (the "parent" table). Only values that exist in the parent table's key are valid values for the specified field(s) of the child table.

Figure 5.1 Referential integrity



You can establish referential integrity only between like fields that contain matching values. For example, you can establish referential integrity between the sample *Customer.db* and *Orders.db* tables on their Customer No fields. The field names don't matter as long as the field types and sizes are identical.

Visual dBASE lets you establish referential integrity for any file type that supports it. You cannot establish referential integrity between .DBF files; however, you can use .DB files if you need referential integrity. You can also use some SQL server tables if you need referential integrity. See your server documentation to determine if your table type supports referential integrity.

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,

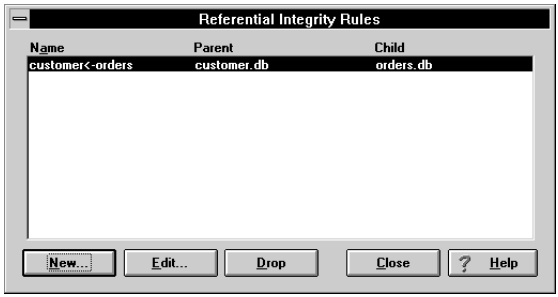
1 Choose File | Database Administration.

  The Database Administration dialog box appears.

2 Specify a Table Type that supports referential integrity, such as Paradox, then choose Referential Integrity.

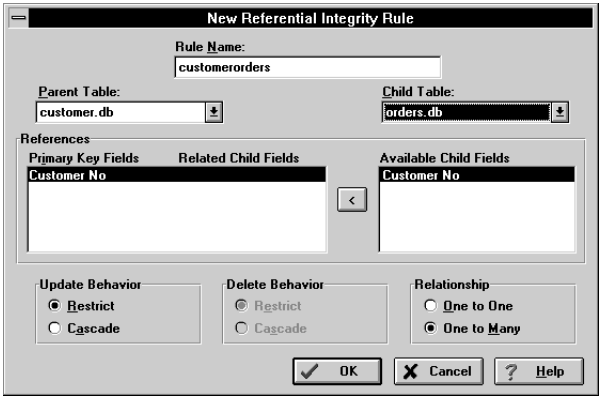
  The Referential Integrity Rules dialog box appears.

Figure 5.2 Referential Integrity Rules dialog box



3 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 lists.



4 Choose the parent table from the Parent Table list. The table's key fields appear in the Primary Key Fields area of the dialog box.

5 Choose the child table from the Child Table list. Fields available for referential integrity appear in the Available Child Fields list.

6 Specify whether the tables are in a one-to-one or one-to-many relationship in the Relationship section. 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.

7 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 referential integrity diagram.

You can establish referential integrity with a composite key. If the parent table has a composite key, add fields from the Fields list to match all of the parent's key fields.

8 Select the update and delete behavior you want. (See "Update and delete behavior" later in this section.)

9 Optionally change the rule name *Visual* dBASE provides.

10 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 to complete 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 records in a child table:

- *Restrict*: You *cannot* change or delete a value in the parent's key if there are records 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 records in *Orders* that contain it). If, however, the value *doesn't* exist in any records 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 records in the child table are also deleted.

The availability of cascading updates and deletes varies according to the data source:

- Paradox: Cascading updates only
- Oracle: Cascading deletes only
- Sybase: No cascading updates or deletes permitted
- InterBase: No cascading updates or deletes permitted
- ODBC: 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.

Visual dBASE security

The Protect system can be used to create and maintain security on a dBASE system. Protect only affects dBASE (DBF) tables; it cannot be used with other file formats. Protect can be used on a single computer or in a network environment.

Protect is optional; you don't have to use it. Once in place, however, the security system will always control access to the affected tables.

Three levels of security

There are three distinct types of database protection, which occur in the following order:

- **Login security** limits access to *Visual* dBASE (or to any protected dBASE table) to authorized personnel only.

  Login security is the first security level. By default, once login security is in place, users can't access *Visual* dBASE until they pass login security. You can set up the security scheme so that users are not forced to login until they try to access a protected table.

- **Data encryption** scrambles dBASE tables so that unauthorized users can't read the information.

  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, *Visual* dBASE automatically encrypts and decrypts tables and their associated index and memo files.

- **Table and field security** lets you define which tables, and which fields within tables, users can access.

  Table and field security is the final security level. It determines what a user can do with both a table and data within the table, and can be used to control processing of program code.

You must implement the security types in the order listed above. For example, you can't establish table and field security without first creating login security. Similarly, you must create login security to have data encryption. *It is not necessary to implement all three levels of security.* Many database administrators implement only login security.

Make sure that users know how to request table protection, if they've created a table that should be protected. Consider developing a form that users can submit to make such requests. Also, keep a hard copy of your security information.

Login security

Protect allows you to create a password-protected system. If password protection is in force,

- Users can only gain access to *Visual* dBASE by entering a valid login statement. The login statement consists of three items: a group name, a login name, and a password.
- The user login screen appears whenever a user tries to start *Visual* dBASE. All paths into the database system initiate the login process.

Decide how your users are to be assigned logins. Will they select their own user login name, password, and group membership, or will you assign them? If you allow users to select values for the login, be sure they know how long user names and passwords can be, and what characters can be used in them.

Password files

When you establish login security, *Visual* dBASE creates and maintains the DBSYSTEM.DB password file, which contains records for each user that you define through the Protect feature. DBSYSTEM.DB stores user profiles, including each user's login name, account name, password, group name, and access level.

When a user starts *Visual* dBASE at a network workstation, dBASE looks for the password file in the dBASE directory. If it's found, the login process is initiated. If it's not found, the user cannot log in.

DBSYSTEM.DB is maintained as an encrypted file that can be decrypted by dBASE. Only a database administrator can view and modify this information.

Groups and user access

Once you have established login security for *Visual* dBASE, you can control access to individual database tables (and to fields within those tables).

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 do not match, the user cannot 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. See Chapter 5 for a description of sessions.

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.

Table privilege schemes

Once you've established each users's access level, you set up a *privilege scheme* for each table. A 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, *Visual* dBASE determines what access the user has to that 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 that:

- 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 can't 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 8 by default (8 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.

Data encryption

A table is not encrypted until you select it, edit the access levels, and save the privilege scheme.

When a table's privilege scheme is saved, *Visual* dBASE encrypts the table, including the production index (.MDX) file and the memo (.DBT) file, if any. dBASE 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.

Using SET ENCRYPTION

Even after a database system has been protected, the database administrator and application programmer maintain control over encryption of copied files.

If a database system has been protected, SET ENCRYPTION is ON by default. If you SET ENCRYPTION OFF, files created with the COPY command won't be encrypted. SET ENCRYPTION can be set from either the Command Window or from the Desktop Properties dialog box in the Properties menu. Refer to online Help for details on this command.

Warning! If you use the COPY TO command and the target file is anything other than a dBASE table (.DBF file), then the target file will be unencrypted even if SET ENCRYPTION is ON.

General procedures

Follow these general steps to set up a protected database system:

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.

Once your planning is done, follow these steps to implement the security scheme:

1. In *Visual* dBASE, start Protect and 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.

The following sections describe these steps in detail.

Planning your security system

This section describes how to plan out your security system. It's a good idea to think through user access and table/field rights before you start entering security profiles into dBASE.

Planning user groups

Take time to think through the various groups you can divide your users into, based on which types of users need access to which tables. For example, an administrative staff might need to access some tables that a sales staff does not, or vice versa.

It helps to develop a worksheet, to map this out in advance. The following table shows one way of organizing this information; use whatever method works best for you.

Table 5.1 Worksheet for defining groups and group members

Table	Group	User name
CUSTOMER	SALES	AMORRIS
		BBISSING
		LJACUS
		FFINE
PRODUCT	ALL	AMORRIS
		BANDERS
		BBISSING
		CDORFFI
		LJACUS

Planning user access levels

Next, think about how much access each user needs to the table.

Although there are 8 access levels, you might standardize on 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 5.2 Worksheet expanded to show 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 table privileges

Next, plan each table's privilege scheme.

For each table operation, you'll 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 can't.

The following worksheet illustrates one way to plan which user access levels grant which table rights.

Table 5.3 Worksheet for defining privileges for table operations

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 5.4 Worksheet for defining field access 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 security system

Once you've planned out your security scheme, you're ready to set it all up. 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 initiating the protect system, make sure any open tables have been closed. Follow these steps to enter the database administrator password:

1. Choose File | Database Administration.
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 tables.
3. Click the Table Security button.

Figure 5.3 Administrator Password dialog box

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.

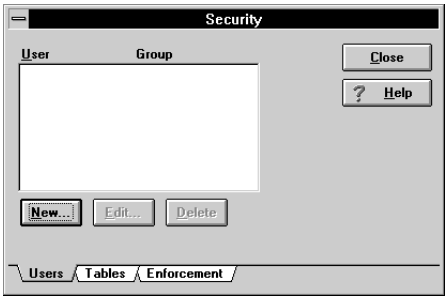
   The first time you use Protect, you're prompted to reenter the password to confirm. (In the future, the system gives you three chances to enter the password correctly before the login terminates.) The Security Administrator dialog box appears, as shown in Figure 5.4.

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.

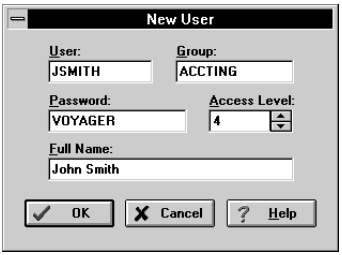
Figure 5.4 Security Administrator dialog box



Follow these steps to add a user profile:

1 In the Security Administrator dialog box, select the Users page and click the New button.

Figure 5.5 New User dialog box



2 Enter a user login name (1–8 alphanumeric characters). The entry is converted to uppercase.

3 Enter a group name (1–8 alphanumeric characters). The entry is converted to uppercase.

4 Enter a password for this user (1–16 alphanumeric characters).

5 Select an access level for this user (from 1 through 8; see page 64).

6 Enter the user's full name (1–24 alphanumeric characters). This entry is optional.

7 Click OK to save the user profile.

You must specify a value for the user login name, group, and password. The full name is the only optional item in a user profile. Since 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 will not be converted to uppercase.

Changing user profiles

To change a user's profile,

1 Open the Users page of the Security Administration dialog box.
2 Select the user name of the user you want to change, and click the Edit button.
3 Make the desired changes, then click OK.

Warning! If you edit the group name, there is no way for the user to access tables associated with the original group. You also shouldn't delete the group. If you delete 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.

Deleting user profiles

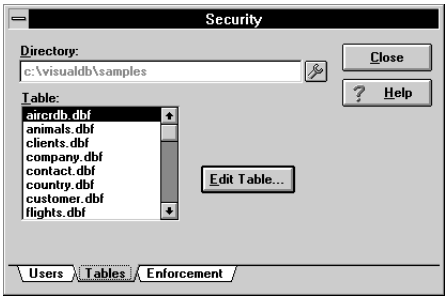
To delete a user profile,

1 Open the Users page of the Security Administration 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 table privileges

You use the Tables page (Figure 5.6) of the Security Administration dialog box to create and modify table privilege schemes. The table privilege schemes are saved in the table structure.

Figure 5.6 Tables page



Use the tables page to:

- Assign a table to a specific group.
- Set table access privileges.
- Set field access privileges for each user access level.

Follow these steps to define table and field privileges for a table:

1 Open the Tables page of the Security Administration 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.

The sections that follow describe these steps in detail.

Selecting a table

To select a table:

1 Open the Tables page of the Security Administration dialog box.

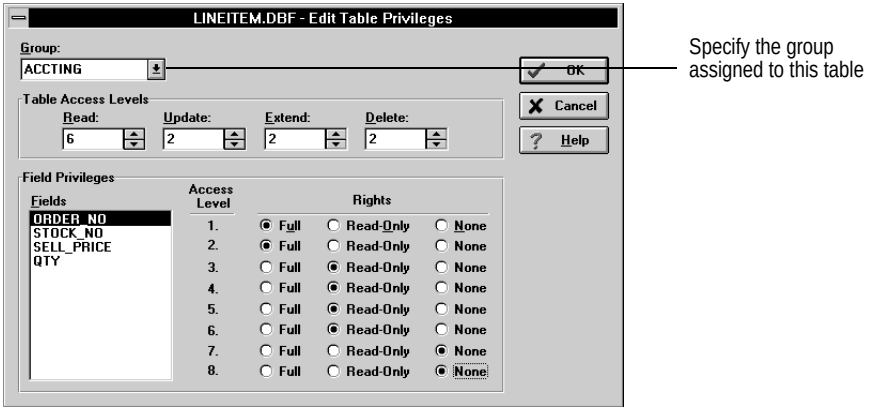
2 In the Table field, type the name of the desired table. (Or click to Tools button and select the table.)

3 Click the Edit Table button. The Edit Table Privileges dialog box opens.

Assigning the table to a group

A table can be assigned to only one group. The group name is matched with a user group name to enable data access.

Figure 5.7 Edit Table Privileges dialog box



Select a group for the table by selecting one of the available groups from the Group list in the dialog box, as shown in Figure 5.7. (These groups were created when you created user profiles.)

Setting table privileges

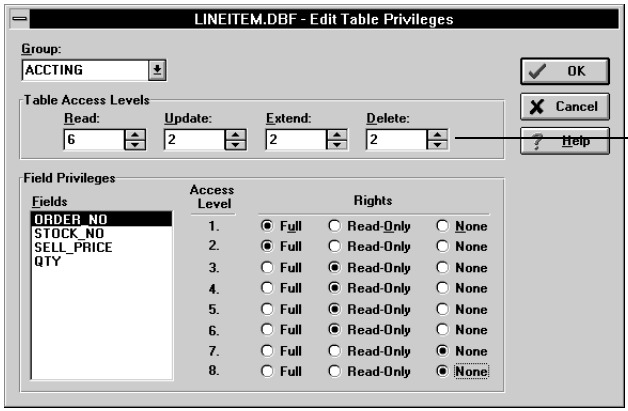
For each type of table operation (see Table 5.5), specify the most restricted access level that can perform that operation.

Table 5.5 Table operations

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.

Figure 5.8 Setting table privileges in the dialog box



Note You can't specify access levels that are logically incompatible. For example, you can't prohibit level 6 from having read access, but also permit level 6 to have update access. To have update access, level 6 also needs read access.

Setting field privileges

You can establish access for each field by user access level. The next table describes the available field privileges.

Table 5.6 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.

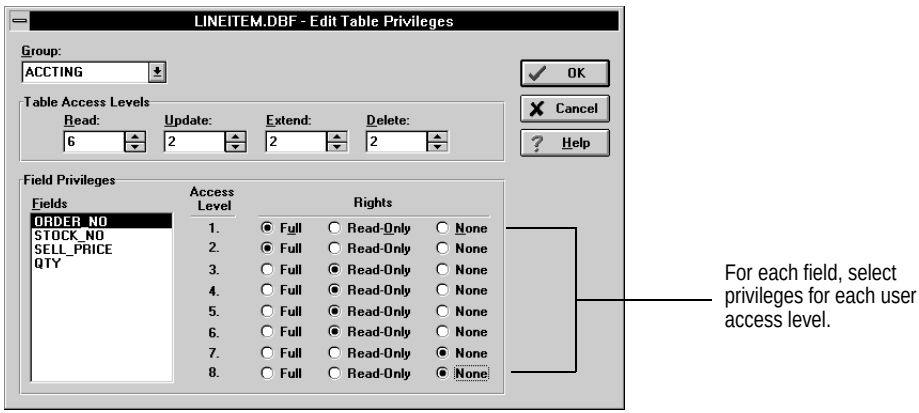
The Fields list in the dialog box lists all of the fields in the table. The Rights buttons display the field privileges for the current field for access levels 1 through 8. Initially, all field privileges are set to Full.

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*.

   For example, to set privileges for the field PAYRATE so that users with access levels 1 and 2 have full access, users with access levels 3 through 6 have read-only access, and users with access level 7 and 8 have no access, define the rights as shown in Figure 5.9.
3. Repeat the process for other fields in the table.
4. Click OK to save the field access privileges.

Figure 5.9 Setting field privileges in the dialog box



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. For details on the _DBASELOCK field see the CONVERT command in online Help.

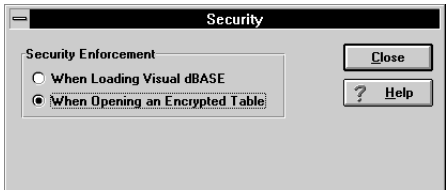
Setting the security enforcement scheme

By default, a protected dBASE system requires a user to login when *Visual* dBASE starts. If you prefer, you can set the security system to require a login only when a user tries to load an encrypted table. This permits anyone to use dBASE and to load unencrypted tables, but prevents unauthorized users from loading protected tables.

To change the security enforcement scheme, follow these steps:

1 Open the Enforcement page of the Security Administration dialog box. The text on the Enforcement page describes the security enforcement scheme currently in effect.

Figure 5.10 Change Security Enforcement dialog box



2 Select the enforcement scheme you want.

3 Click OK.

Adding passwords to Paradox tables

In Paradox, you can assign passwords to Paradox tables. Once a password is assigned, the table can't be opened in either dBASE or Paradox without supplying the password. To assign a password to a Paradox table from within *Visual* dBASE, follow these steps:

1 Make sure to table you want to secure is closed.

2 From the File menu, select Database Administration.

3 Make sure that the Current Database field is set to <None> and the Table Type field is set for Paradox tables.

4 Click the Table Security button to open the Security Administration dialog box.

5 Enter the name of table in the Table field (or click to tool icon to select the table). If the table is not in the current directory, enter the appropriate directory (or use the tool to select it).

6 Click the Edit Table button to open the Set 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 Paradox tables

To remove an existing password from a Paradox 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

6

Enhancements to the dBASE Language

Visual dBASE provides enhancements to three key areas of the dBASE Language: commands and functions, stock classes, and embedded SQL. This chapter includes tips for upgrading applications and a detailed reference for the changes to each area of the dBASE Language.

Note This chapter assumes you have a basic understanding of the dBASE Language, the dynamic object model and event-driven programming. It only covers changes since dBASE 5.0. This guide contains no information on upgrading from dBASE III PLUS or dBASE IV. Please refer to the Programmer's Guide for a tutorial on the dBASE Language and information on migrating dBASE DOS applications.

Most dBASE applications will run without any change in *Visual* dBASE. This section gives an overview of the most significant new language features you can integrate into existing applications and covers the few areas that might require changes. The language changes fall into four categories: database operations, encapsulation, stock classes, Windows95 support and new operators.

Note See Chapter 3 for information on migrating existing dBASE 5.0 for Windows applications.

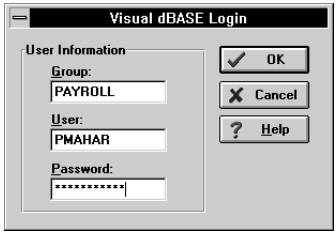
Database operations

Visual dBASE provides robust database support through data administration tools and embedded SQL. There are new commands to work with encrypted tables and ANSI SQL-92 compliant embedded SQL. While both of these areas are new to *Visual* dBASE, they are not new to dBASE. The commands for working with encrypted tables are fully compatible with dBASE for DOS. The SQL implementation differs dramatically from the implementation provided under dBASE for DOS, but the syntax for individual SQL statements remains the same.

Table Security

Database administrators can create table security rules and encrypt tables. The security system lets you define user profiles to restrict a user's ability to view and modify tables and fields within a table. For instance, you can encrypt the payroll table to restrict its use to individuals in the payroll department. You can also restrict field and update rights within the payroll department. For instance, you could limit clerks to "read only" rights on the salary field and give managers all rights.

Figure 6.1 Logging into *Visual* dBASE



Chapter 5 contains complete information on database administration and setting up table security on DBF and DB tables. Here are the dBASE commands and functions for working with table security system:

- **ACCESS( )** returns the dBASE user level for the current user. dBASE user rights and levels are independent of any network profile. The PROTECT command opens up the database administration tools for creating dBASE user profiles.
- **LOGOUT** closes all open tables, logs out the current user and opens the login dialog. Figure 5.1 shows the login dialog that appears when you use the LOGOUT command.
- **PROTECT** opens the database administration tools for setting up security on DBF tables. To setup security on DB tables, select File | Database Administration.
- **SET ENCRYPTION** determines if tables created from encrypted tables inherit the security attributes of the source table. The most common use of SET ENCRYPTION is to create unencrypted copies of tables for backup or archive purposes. Since DBF and DB recovery tools such as dSALVAGE do not work with encrypted tables, it is a good idea to back them up without encryption.
- **USER( )** returns the dBASE user name. As with dBASE user level, the dBASE user name is independent of network login names. Use ID( ) to get the network user name.

Embedded SQL

While Xbase is the most common language for database application development, SQL is the de-facto standard for data manipulation language for almost all client/server development. SQL is a not a complete language. It has a small set of commands that deal only with table operations and must exist within a host system that provides user interface objects, program flow statements and variable manipulation.

Earlier versions of dBASE implemented SQL with varying degrees of success. dBASE for DOS only allows SQL statements in special procedure files called PRS files and restricts the interaction between SQL and standard Xbase DML. dBASE for DOS SQL also translates SQL statements into standard Xbase DML rather than using an SQL database engine. dBASE 5.0 for Windows includes the SQLEXEC( ) function that works with any program or procedure file. Although SQLEXEC( ) works with the native SQL engine in BDE, it does not create live views or cursors that are visible to the dBASE environment.

Visual dBASE provides the only Xbase product with ANSI SQL-92 compliant SQL that is capable of creating live and fully updatable views. You can mix and match SQL statements with standard Xbase DML. You can use the SQL engine in BDE to work with DB and DBF tables. BDE also optimizes performance for remote tables by passing SQL statements directly to back-end servers connected through SQL-Links or ODBC. See the last section in this chapter for detailed information on embedded SQL.

Encapsulation

You can now fully encapsulate properties and methods in the class definition. The new PROTECT clause of the CLASS...ENDCLASS statement hides properties and methods from procedures defined outside the class. The PROTECT clause is similar to the LOCAL and STATIC scope specifiers. When a variable is LOCAL or STATIC, it is not visible to any routines outside the routine that declared it. The encapsulation PROTECT clause is not related to the table security PROTECT command

The following procedure shows how full encapsulation works. The class uses the PROTECT clause to hide the *top* property and the *funk1* method. When you create an instance of a class, the object reference can refer to any property or method that is not hidden. In this case, *left* and *funk2* are not hidden. The TYPE( ) function reveals that *top* and *funk1* are undefined. Notice that the *funk2* procedure can refer to hidden members. This is allowed since *funk2* is defined within the class. You can use this technique to create visible or public methods that operate on hidden members.

```
f1 = NEW MyForm()
? TYPE('f1.left')  && numeric
? TYPE('f1.top')   && undefined
? TYPE('f1.funk1') && undefined
? f1.funk2()       && can call funk1
CLASS myForm OF FORM
      PROTECT top, funk1
      this.top = 5
      FUNCTION funk1
         this.top = 7
      RETURN 0
      FUNCTION funk2
         CLASS::funk1()
      RETURN this.top
ENDCLASS
```

Using the new classes, properties, events, and methods

Visual dBASE contains seven new stock classes and a wealth of new properties, events and methods for existing classes. There are five new classes for creating UIObjects: MenuBar, PaintBox, Popup, Shape and TabBox. You will find all of these on the Form Designer's Control Palette. Two new classes, AssocArray and OLEAutoClient, do not create user interface controls: . New properties, events, and methods allow you to control the record buffer, Windows clipboard, SpeedTips (tool tips), and multiple page forms.

Working with the new stock classes

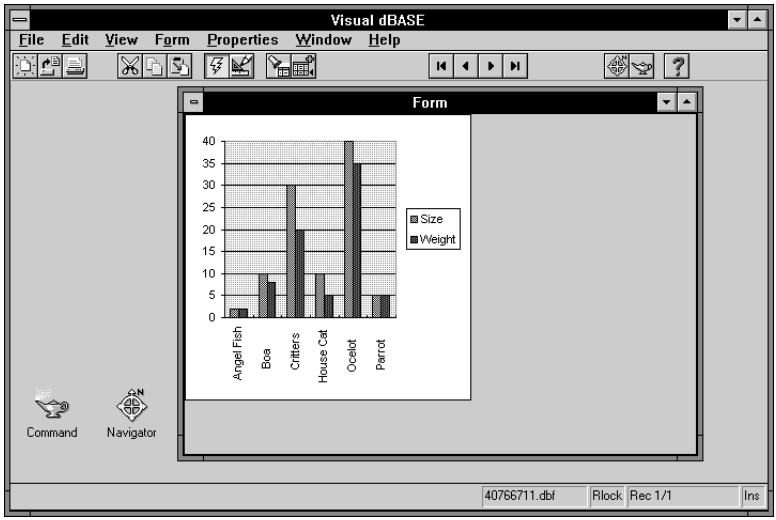
Here is a brief overview of each of the new stock classes. See Chapter 7 for reference information.

- **AssocArray** creates an associative array that uses character strings as the index. Standard arrays use a numeric subscript. The following procedure shows how you can create, examine and resize an associative array.

```
USE animals
aa = NEW AssocArray()
SCAN                            && fill from character fields
   aa[ RTRIM(animals->name) ] = animals->area
ENDSCAN
CLOSE TABLE
CLEAR
IF aa.IsIndex( "Angel Fish" )     && look for name
   ? aa[ "Angel Fish" ]           && retrieve area
   ? aa.Count(), 'before'         && check size
   ? aa.RemoveKey( "Angel Fish" ) && remove item
   ? aa.Count(), 'after'          && check new size
ENDIF
```

- **MenuBar** replaces the Menu object at the root of a menu tree. If you modify existing MNU files with the Menu Designer, dBASE replaces the MENU object at the root with a MenuBar object. All other menu objects remain the same. This is required to take advantage of the new cut, copy, paste, undo and window menu properties.

Figure 6.2 Chart created with OLE2 automation



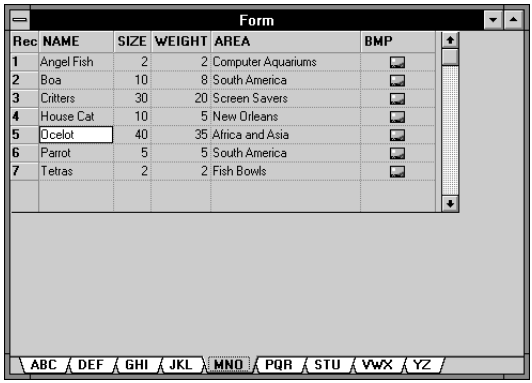
- **OLEAutoClient** gives you full control of OLE2 servers such as the latest releases of Excel, Word, Quattro Pro and WordPerfect. The properties, events and methods of an OLEAutoClient object are determined by the server application. OLE2 applications vary greatly in their implementations and you should refer to the server applications documentation for specifics. The following program demonstrates how to create a chart and place it in a form using Excel.

```
* open objects
aSheet = NEW OLEAutoClient("Excel.Sheet")
aSheet.Application.Visible = .T.
USE animals
* make grid
cell = aSheet.Cells( 2, 1 )
cell.Value = "Size"
cell = aSheet.Cells( 3, 1 )
cell.value = "Weight"
SCAN
    cell = aSheet.cells( 1, RECNO() + 1)
    cell.value = animals->name
    cell = aSheet.cells( 2, RECNO() + 1)
    cell.value = animals->size
    cell = aSheet.cells( 3, RECNO() + 1)
    cell.value = animals->weight
ENDSCAN
* make chart
aChart = aSheet.ChartObjects.Add(100, 100, 200, 200)
series = aChart.chart.SeriesCollection
series.Add("Sheet1!R1C1:R2C7",.T.)
series.Add("Sheet1!R3C1:R3C7",.T.)
* copy chart to dBASE
```

```
aChart.Copy()
aSheet.application.quit()
* create temporary table to contain chart
temp1DBF = FUNIQUE()
CREATE (temp1DBF) STRUCTURE EXTENDED
APPEND BLANK
REPLACE field_name WITH "OLE", field_type WITH "G"
temp2DBF = FUNIQUE()
CREATE (temp2DBF) FROM (temp1DBF)
CLOSE TABLES
DELETE TABLE (temp1DBF)
 * create form with OLE control
USE (temp2DBF) NOSAVE
APPEND BLANK
f1           = new form()
ole          = new ole(f1)
ole.width    = f1.width
ole.height   = f1.height
ole.datalink = "OLE"
f1.OnOpen =  { ; KEYBOARD "{Ctrl+V}" CLEAR }
f1.OnClose = { ; CLOSE TABLES }
f1.open()
* end of Excel example
```

- **PaintBox** gives advanced and add-on product developers a direct link between the Windows API and a dBASE control object. The PaintBox contains an OnPaint event in addition to the standard event handlers. dBASE gives you full control of repainting the PaintBox region.
- **Popup** is the root object of a popup menu. See Chapter 3 for more information on creating popups and attaching them to forms.
- **Shape** allows you to draw simple shapes on forms. Like the line object, shape is not in the control tab order.

Figure 6.3 Using a TabBox to locate records



- **TabBox** is normally used to change pages in a multiple page form. See chapter 3 for information about selecting pages through the OnSelChange event. You can also use TabBox for record navigation in a single page form. The following procedure creates

a TabBox that serves as an alphabetical index into the Animals table. Figure 5.3 shows the program in action. In this case the TabBox acts much like the tab controls in popular address book programs such as Sidekick.

```
SET NEAR ON
USE animals ORDER TAG name
f1 = NEW TabForm()
f1.OPEN()
CLASS TabForm OF FORM
      DEFINE TABBOX TabFind OF this ;
            PROPERTY ;
            DataSource "ARRAY {'ABC','DEF','GHI','JKL','MNO', ;
                        'PQR','STU','VWX','YZ'}", ;
            OnSelChange { ;SEEK( CHR( 62 + (this.CurSel * 3 )))}
      DEFINE BROWSE BrowseAnimals OF this ;
            PROPERTY ;
            width  70, ;
            height 10
ENDCLASS
```

Working with the new properties, events, and methods

There are many new properties, events, and methods. Here are a few groups to outline some uses for the new properties, events, and methods.

- **Record Buffer** methods include AbandonRecord( ), BeginAppend( ), IsRecordChanged( ) and SaveRecord( ). While dBASE has complete local and server transaction support using BeginTrans( ), Commit( ) and RollBack( ), the new record buffer methods let you control the dBASE record buffer for optimum performance.
- **Clipboard** methods let you cut, copy, and paste information to and from the Windows clipboard. There is also an Undo( ) method for reversing the last clipboard action. These methods are available on all controls that allow keyboard editing. See chapter 3 for information on the MenuBar object and how to setup menus that work with the clipboard.
- **SpeedTips** are helpful text messages that appear near a control when the mouse pointer is positioned over the control. Many of the UIObjects now have a SpeedTip property. The form's new ShowSpeedTips property controls the display of all SpeedTips on the form.
- **Page** properties let you create and navigate through multiple page forms. Each control has a PageNo property that ties it to a specific page. The PageNo property of a form determines what page is currently active. Forms also have a PageCount( ) method that returns the highest page number that contains a control. The TabBox OnSelChange event is the most common event for switching pages. See chapter 3 for information on setting up multiple page forms.
- **MarkCustom**is a new method for creating custom controls that inherit from multiple classes. This lets you encapsulate the properties of a control this is created in the constructor of a custom control class. See the sample custom controls for examples of using MarkCustom.

Windows95 Support

Visual dBASE fully supports Windows95 through both the Desktop and the language. The Desktop, Visual Property Builders, and designers automatically adjust to work with long file names. Similarly, your applications will run without change in Windows95. The language has some new extensions that help you take advantage of the new operating system.

One of the biggest changes in Windows95 is the new file system that supports long file names and an extended set of attributes. Long file names can contain spaces and special characters that are invalid under earlier versions of Windows. To refer to a long file name in a dBASE application, place the name in quotes. Similarly you can create and work with long directory names by placing them in quotes. Here are some examples of working with Windows95 long names.

```
CREATE FORM "Accounts Receivable.WFM"
MODIFY REPORT "Top Ten Artists.RPT"
CD "C:\Musical Methods"
DO "Sales Rankings.PRG"
USE "Categories.DBF"
```

To maintain compatibility with Delphi and Paradox, BDE restricts names to DOS limits during table creation. If you want to work with long table names, you need to rename the tables using the COPY FILE command. The following function, *Copy2Big*, provides a quick way to make long file names for existing tables.

```
* Program:      Copy2Big
* Description:  Copies an existing table
*               to new table with long
*               table name under Windows95
* Syntax:
* Copy2Big( <source>, <destination>)
*
* source:      expC short DBF name - no extension
* destination: expC long DBF name - no extension
*
* returns .T. if file is created.
*
* example:
* Copy2Big("Animals","Animals Of The World")
*
FUNCTION Copy2Big( lcFrom, lcTo )
  CLOSE TABLES
  SET SAFETY OFF

  ** copy without index tags
  LOCAL lcTemp
  lcTemp = FUNIQUE()
  USE (lcFrom)
  COPY TO (lcTemp)
  COPY FILE lcTemp + ".DBF" TO lcTo + ".DBF"
  IF  FILE( lcTemp + ".DBT" )  && check for memo
      COPY FILE lcTemp + ".DBT" TO lcTo + ".DBT"
  ENDIF
```

```
    DELETE TABLE (lcTemp)

    ** create index tags
    LOCAL lKey, lName, lFor, lDescend, lUnique
    USE (lcTo) EXCLUSIVE IN SELECT()
    SELECT (lcTo)
    FOR i = 1 TO TAGCOUNT( lcFrom, lcFrom )
        lKey     = KEY( i, lcFrom )
        lName    = TAG( i, lcFrom )
        lFor     = FOR( i, lcFrom )
        lDescend = IIF(DESCENDING( i, lcFrom ), ;
                   "DESCENDING", SPACE(0) )
        lUnique  = IIF(UNIQUE( i, lcFrom ), ;
                   "UNIQUE", SPACE(0) )
        PRIVATE macro
        macro = lKey + SPACE(1) + "TAG" + SPACE(1) + lName + SPACE(1);
        + IIF( ISBLANK(lFor), SPACE(1), "FOR"  + SPACE(1) + lFor ) ;
        + SPACE(1) + lDescend + SPACE(1) + lUnique
        INDEX ON &macro.
        RELEASE macro
    NEXT

    ** clean up and check if file was created
    CLOSE TABLES
    SET SAFETY ON
  RETURN FILE( lcTo + '.DBF')
```

The Windows95 file system provides an expanded set of attributes including a short file name or alias for older applications. dBASE contains the following new functions for examining the new attributes: FACCESSDATE( ), FCREATEDATE( ), FCREATETIME( ), and FSHORTNAME( ). The Array class contains a new DirExt( ) method that fills the array with standard and extended attributes for all the files in a directory. To determine if a drive supports long file names you can use the new FNAMEMAX( ) function.

New operators

Visual dBASE contains new operators, one that simplifies array creation and another for working with the alias of a long table name. Braces, { }, now serve as an array declaration and assignment operator in addition to delimiting dates and code blocks. The colon, : , delimits the alias of a long table name and continues to be the delimiter for long field names.

Using braces, { }, to create arrays

Along with using the DECLARE command and the NEW operator to create arrays, you can now create them using braces. The advantage of using braces is that you can assign elements to the array during creation. Here are three examples of creating an array.

```
* Using the DECLARE statement
DECLARE colors[3]
colors[1] = "Red"
colors[2] = "White"
colors[3] = "Blue"

* Using the NEW oprator
colors = NEW ARRAY()
colors.Add("Red")
colors.Add("White")
colors.Add("Blue")

* Using braces
colors = {"Red","White","Blue"}
```

Each technique creates the exact same array. The first technique provides compatibility with the older Xbase programs. The NEW operator is the most flexible, but still requires four lines of code. Using braces produces the same result with only one line of code.

Keep in mind that braces still serve as code block and date delimiters. dBASE examines the values within braces to determine if it is an array, code block or date. A single value or comma delimited list of values specifies an array. Three numbers arranged as a literal date are a date. If the character after the opening brace is a semicolon or a vertical bar, the braces contain a code block.

```
* single element array
SmallArray = { "One" }

* date literal
Birthday = {10/30/67}

*  code block
EndProgram = { ; QUIT }

* code block that creates an array of date literals
MakeArray = { ;  SpecialDays = { { 11/02/85}, {03/12/88} } }
```

Note Code blocks in dBASE 5.0 applications may appear as literal arrays in *Visual* dBASE. See Chapter 3 for tips on upgrading dBASE 5.0 applications.

Using the colon, :, to delimit a table alias

If you open a table using a long file name under Windows95, dBASE creates a long alias name. Since the name can contain spaces and special characters, you need to delimit it with the colon. This is the same delimiter that works with long field names. Here are some examples that use colon to delimit the alias and fields:

```
USE "C:\Musical\Methods\Top Ten Artists.DB" IN SELECT()

REPLACE :Top Ten Artists:->:First Name: WITH "Travis"

SELECT :Top Ten Artists:

REPLACE :Last Name: WITH "Shook"
```

Chapter

7

New commands, functions, classes and properties

This chapter is a subset of the *Language Reference* covering items changed since dBASE 5.0 for Windows. The complete *Language Reference* is available as online help. The chapter is organized as follows:

- **NULL data type** describes how to use NULL in expressions. Read this section if you work with NULL data in .DB tables or servers such as ORACLE, Sybase, and InterBase.
- **Commands and Functions** includes all of the syntax elements such as looping structures and mathematical functions. The section is a language reference for all new and changed commands and functions.
- **Classes** provide common user interface, DDE, OLE and array objects. The classes section is a reference for the seven new stock classes.
- **Properties** as listed in the language reference refers to properties, events and methods for stock classes. The properties section covers properties of new classes and new properties for existing classes.

NULL data type

Visual dBASE supports the concept of a null value. NULL is a constant representing the absence of a value, as opposed to a value of 0 for a numeric or an empty string for a character variable. NULL is not supported in fields of .DBF files, but may be supported by other database file formats which *Visual* dBASE can access.

When *Visual* dBASE is used to access tables which support null values, NULL can be used to set a field to a null value or to retrieve a null value from a field. Therefore, each of the following would be legal statements:

```
REPLACE TABLE->FIELD WITH NULL
SET FILTER TO TABLE->FIELD <> NULL
SEEK NULL
```

Null can be used as a value in an expression. In general, if the expression needs to manipulate the value passed to it, and that value is null, the expression will return NULL. For example:

```
? SIN(0)       && Returns 0.00
? SIN(NULL)    && Returns NULL
? UPPER("")    && Returns an empty character string
? UPPER(NULL)  && Returns NULL
? 4 + 0        && Returns 4
? 4 + NULL     && Returns NULL
```

If a function does not need to manipulate the value passed, the NULL will be converted to its default value for the type of argument the function expects. For example:

```
? TIME()       && Returns the time to whole second
? TIME(0)      && Returns the time to hundredths of a second
? TIME(NULL)   && Returns the time to hundredths of a second, NULL treated as 0
```

Table 7.1 shows the internal default values for NULL that dBASE supports.

Table 7.1 Default values for NULL

Type	Default value
Bookmark	"" (empty character string)
Character	blank character(s)
Date	{//} (empty date)
Logical	.F.
Numeric	0
Unknown	0, "", .F., or {//}

The default data type of NULL is unknown. NULL can be assigned a data type by using it in an expression expecting a certain type, although it retains the value NULL:

```
x = NULL       && x assigned NULL of unknown type
? TYPE("x")    && Returns "U"
? UPPER(x)     && Returns NULL of character type, x still unknown type
x = SIN(NULL)  && Returns NULL of numeric type and assigns to x
? TYPE("x")    && Returns "N", x is numeric but still has NULL as value
? UPPER(x)     && Data type mismatch error, UPPER() expects a character type parameter
```

Commands and functions

ACCESS( ) — Security

Returns the access level of the current user, as assigned with the PROTECT command.

Syntax
ACCESS( )

Description
Use ACCESS( ) to build security into an application. The access level returned can be used to test privileges assigned with PROTECT. If a user is not logged in to the application, ACCESS( ) returns 0 (zero).

If you write programs that use encrypted files, check the user's access level early in the program. If ACCESS( ) returns zero, your program might prompt the user to log in, or to contact the system administrator for assistance.

For more information, see PROTECT.

See Also
LOGOUT, PROTECT, SET ENCRYPTION, USER()

BEGINTRANS( ) — Shared data

Starts a transaction and returns .T. if the transaction started successfully.

Syntax
BEGINTRANS([<*database name expC*>] [<*isolation level expN*>])

<database name expC> The name of the SQL database in which to begin the transaction.

- If <*database name expC*> is omitted but a SET DATABASE statement has been issued, BEGINTRANS( ) refers to the database in the SET DATABASE statement.
- If <*database name expC*> is omitted and no SET DATABASE statement has been issued, BEGINTRANS( ) refers to the database opened after issuing BEGINTRANS( ).

<isolation level expN> Specifies a pre-defined server-level transaction isolation scheme.

- Valid values for <isolation level> are:

expN	Description
0	Server's default isolation level
1	Uncommitted changes read (dirty read)

expN	Description
2	Committed changes read (read committed)
3	Full read repeatability (repeatable read)

- <isolation level> is not supported for local tables.
- If an invalid value is given for <isolation level>, a dBASE "Value out of range" error is generated.
- The <isolation level> is server-specific; a "Not supported" error will result from the database engine if an unsupported level is specified.

Note If you include *<database name expC>* when you issue BEGINTRANS( ), you must also include it in subsequent COMMIT( ) or ROLLBACK( ) statements within that transaction. If you don't, dBASE ignores the COMMIT( ) or ROLLBACK( ) statement.

Description

Use BEGINTRANS( ) to initiate a *transaction* during which the user might make changes to a table or tables in an SQL database that supports transaction processing. Within a transaction initiated with BEGINTRANS( ), you can work in only one database. Also, you can't nest transactions.

If you issue BEGINTRANS( ) against an SQL database that does not support transactions, or if a server error occurs, BEGINTRANS( ) returns .F. Otherwise, it returns .T. If BEGINTRANS( ) returns .F., use SQLERROR( ) or SQLMESSAGE( ) to determine the nature of the server error that might have occurred.

To close a transaction, use COMMIT( ) or ROLLBACK( ).COMMIT( ) saves changes made during the transaction, and ROLLBACK( ) discards changes and returns tables to the state they were in before BEGINTRANS( ) was issued.

BEGINTRANS( ) applies to the following commands:

@...SAY...GET	DELETE	REPLACE
APPEND	EDIT	REPLACE MEMO/BINARY/OLE
APPEND BLANK	FLOCK( )	RLOCK( )
APPEND MEMO	INSERT	
BROWSE	RECALL	

The following commands are not allowed in transactions. dBASE returns an error if you try to issue them from within a transaction.

BEGINTRANS( ) (nested transactions are not allowed)	DELETE TAG
CLEAR ALL	INDEX
CLOSE ALL/DATABASE/INDEX (any command that closes open tables or indexes)	MODIFY STRUCTURE

CONVERT	PACK
CREATE FROM	a USE that would close an open table, or open a table in another database
ZAP	

Example

The following example begins a transaction with BEGINTRANS( ). It opens a multi-user version of Company.dbf and attempts to make all YTD_SALES 0. ON ERROR detects any error which might occur. In particular, it will detect if another user has locked any record in Company.dbf. If an error occurs, ROLLBACK( ) resets all values. Otherwise COMMIT( ) writes the changes to disk:

```
CLOSE ALL
SET PROCEDURE TO PROGRAM(1) ADDITIVE
SET EXCLUSIVE OFF

BEGINTRANS()

TransErr=.f.
ON ERROR DO TransErr    && Activates ON ERROR trap

USE L:\MultiUse\Company
REPLACE ALL Ytd_Sales WITH 0
ON ERROR                && Disables ON ERROR

IF TransErr
  ? "Rollback"
  ROLLBACK()            && restore data
ELSE
  ? "Commit"
  COMMIT()              && save changes
ENDIF

PROC TransErr
WAIT "Warning: Transaction Fails"
TransErr=.t.
```

Portability

Not supported in dBASE IV or dBASE III PLUS. BEGINTRANS( ) replaces the BEGIN TRANSACTION and END TRANSACTION commands in dBASE IV.

See Also

COMMIT( ), FLOCK( ), RLOCK( ), ROLLBACK( ), SET EXCLUSIVE, SQLERROR( ), SQLMESSAGE( )

BUILD

Links object code files (.PRO, .WFO) and resources into a Windows executable file (.EXE) if the optional *Visual* dBASE Compiler is installed.

Syntax

BUILD <*filename list*> | FROM <*reponse filename*>
[ICON <*icon filename*>]
[SPLASH <*bitmap format filename*>]
[TO <*executable filename*>]
<*filename list*>

List of filenames, separated by commas, to be linked into the executable. Unless otherwise specified, the filename extensions are assumed to be .PRO.

FROM <*response filename*> Build the executable from the list of files listed in <response filename>. Unless otherwise specified, the extension of the response file is assumed to be .RSP. See the online help for details on the format of the response file.

ICON <*icon filename*> The optional icon (.ICO) file which is displayed when the executable is minimized. The icon file is also the default icon when the executable is represented in Program Manager.

```
SPLASH <bmp format filename>
```

The optional graphics (.BMP) file that displays as the executable is loading.

TO <*executable filename*> The name of the Windows executable file (.EXE) produced by BUILD. If not specified, the executable filename defaults to the name of the first file listed in the <filename list> or the <response file>.

Description

Use the BUILD command to link previously compiled dBASE program (.PRO, .WFO) files, Windows resource files (such as .BMP and .ICO files), and other files needed to support an application into a Windows executable (.EXE) file. See the online help for the Visual dBASE compiler for details on creating executable files from dBASE programs.

Portability

Not supported in dBASE IV or dBASE III PLUS. Very similar in functionality to the BDL.EXE linker utility in the dBASE Compiler for DOS.

See Also

COMPILE, DO, SET FORMAT, SET PROCEDURE

CLASS...ENDCLASS

Objects

Declares a custom class and specifies the member variables and functions for that class.

Syntax

CLASS <*class name*> [(<*parameters*>)] [CUSTOM]
[PARAMETERS <*parameters*>]
[FROM <*filename*>]
[OF <*superclass name*> [(<*parameters*>)]]
[PROTECT <*propertyList*>]
[<*constructor code*>]
[<*member functions*>]
ENDCLASS

CUSTOM Specifies that the new object is a custom control. For information on custom controls, see Chapter 15 of the *Programmer's Guide*.

<*class name*> The name you give to the new class.

OF <*superclass name*> Specifies that the class you create inherits the properties and methods of <*superclass name*>. For example, you can give your new class all the properties and methods of the listbox class or another class you create with CLASS...ENDCLASS.

FROM <*filename*> <*filename*> specifies the file containing the definition code for the <*superclass*>, if the <*superclass*> is not defined in the same file as the class.

PROTECT <*propertyList*> <*propertyList*> is a list of properties and/or methods of the class which are to be accessible only by other members of the class, and by classes derived from the class.

<*constructor code*> Code that is executed when you create an object of class <*class name*>. Constructor code includes all commands between the CLASS and ENDCLASS keywords except code in <*member functions*>.

<*member functions*> Procedures and functions that you declare between the CLASS and ENDCLASS keywords. These subroutines make up the methods of the new class.

Description

Use CLASS...ENDCLASS to create a new class.

A class is a specification, or template, for a type of object. *Visual* dBASE provides many standard classes, such as Form and Entryfield; for example, when you create a form, you are creating a new form object that has the standard properties and methods from the Form class. However, when you declare a custom class with CLASS...ENDCLASS, *you* specify the properties and methods that objects derived from the new class will have.

You create properties for the new class with <*constructor code*>. Constructor code executes when you create an object of the class. Although constructor code can contain any dBASE commands, it usually contains only property and method assignment statements.

Properties and methods can be protected to prevent the user of the class from reading or changing the protected property values, or calling the protected methods from outside of the class.

When you create a new property in a class declaration, preface the property name with the *This* keyword. *This* references the object you create. For example, the following code sample includes a class declaration. The declaration uses This to specify that TagName, a new property, is a member of the new class TableFile.

```
xFile = NEW TableFile()
? xFile.TagName
? xFile.FileNameId()

CLASS TableFile
   This.TagName = "XORDER"
   FUNCTION FileNameId&& Custom method.
   RETURN DBF()
ENDCLASS
```

You create custom methods for the class with *<member functions>*, which can consist of procedure declarations or user-defined function declarations. FUNCTION FileNameId is an example of a custom method.

Example

The following example uses CLASS...ENDCLASS to define a class of objects within a form that displays pictures from the Pictures table in the SAMPLES directory. This example is an extract from PICTURES.WFM in the SAMPLES directory:

```
** END HEADER -- do not remove this line*
* Generated on 06/27/94
*
LOCAL f
f = NEW PFORM()
f.Open()

CLASS PFORM OF FORM
PROTECT HelpFile, HelpId
   this.HelpFile = ""
   this.Width =         97.00
   this.Maximize = .F.
   this.Minimize = .F.
   this.Height =         24.82
   this.Left =         19.40
   this.Text = "Pictures Form"
   this.Top =          0.53
   this.ColorNormal = "BG/B"
   this.OnOpen = {;create session}
   this.View = "PICTURES.QBE"
   this.HelpId = ""

   DEFINE PUSHBUTTON SOUND OF THIS;
       PROPERTY;
         Width         18.00,;
         Default .T.,;
         OnClick {;play sound binary pictures->sound},;
         Height          3.00,;
```

```
        Left          4.00,;
        Text "Sound",;
        Top           8.00,;
        ColorNormal "N/W",;
        FontName "Courier",;
        FontSize         16.00

   DEFINE LISTBOX THINGS OF THIS;
       PROPERTY;
         Width         18.40,;
         DataSource "FIELD NAME",;
         ColorHighLight "W+/B",;
         Height          5.47,;
         Left          3.60,;
         Top         13.41,;
         ColorNormal "bg+/b",;
         FontName "Fixedsys",;
         FontSize         11.25,;
         ID        800

   DEFINE IMAGE PICTURE OF THIS;
       PROPERTY;
         Width         62.00,;
         DataSource "BINARY PICTURES->BITMAPOLE",;
         Height         18.00,;
         Left         25.00,;
         Top          5.00,;
         ID         88

   DEFINE TEXT TITLE OF THIS;
       PROPERTY;
         Width         70.00,;
         Height          4.30,;
         Left         20.00,;
         Text "Sights and Sounds",;
         Top          0.00,;
         ColorNormal "gr+/b",;
         FontName "Serif",;
         Border .F.,;
         FontSize         32.00
* Provide methods to get and set the HelpFile
   * and HelpID properties, since the user can't
   * access them directly
   FUNCTION GetHelpFile
   RETURN This.HelpFile
   FUNCTION GetHelpID
   RETURN This.HelpID
   FUNCTION SetHelpFile(cHelpFile)
      IF TYPE("cHelpFile") = "C"
...      This.HelpFile = cHelpFile
      ENDIF
   RETURN This.HelpFile
   FUNCTION SetHelpID(cHelpID)
      IF TYPE("cHelpID") = "C"
         This.HelpID = cHelpID
      ENDIF
```

```
   RETURN This.HelpID

ENDCLASS

PROCEDURE Sound_OnClick
PLAY SOUND Binary Pictures->Sound

PROCEDURE ClosePictures
USE IN Pictures
FORM.CLOSE()
```

Portability
Not supported in dBASE IV or dBASE III PLUS.

See Also
DEFINE, REDEFINE

COMMIT( ) — Shared data

Ends a transaction initiated by BEGINTRANS( ) and writes to the open files any changes made during the transaction. Returns .T. if the data was committed successfully.

Syntax
COMMIT([<*database name expC*>])

<database name expC> The name of the database in which to complete the transaction.

- If you began the transaction with BEGINTRANS(<*database name expC*>), you must issue COMMIT(<*database name expC*>). If instead you issue COMMIT( ), dBASE ignores the COMMIT( ) statement.
- If you began the transaction with BEGINTRANS( ), <*database name expC*> is an optional COMMIT( ) argument. If you include it, it must refer to the same database as the SET DATABASE TO statement that preceded BEGINTRANS( ).

Description
Use COMMIT( ) to end the open transaction and write changes to any open files. To end a transaction without writing changes to the file, use ROLLBACK( ). For more information on transactions, see BEGINTRANS( ).

Example
The following example begins a transaction with BEGINTRANS( ). It opens a multi-user version of Company.dbf and attempts to set all values in the Ytd_Sales field to 0. ON ERROR detects any error which might occur. In particular, it will detect if another user has locked any record in Company.dbf. If an error occurs, ROLLBACK( ) resets all values. Otherwise COMMIT( ) writes the changes to disk:

```
CLOSE ALL
SET PROCEDURE TO PROGRAM(1) ADDITIVE
SET EXCLUSIVE OFF

BEGINTRANS()

TransErr=.f.
ON ERROR DO TransErr          && Activates ON ERROR trap

USE L:\MultiUse\Company
REPLACE ALL Ytd_Sales WITH 0
ON ERROR                      && Disables ON ERROR

IF TransErr
  ? "Rollback"
  ROLLBACK()                  && Restore data
ELSE
  ? "Commit"
  COMMIT()                    && Save changes
ENDIF

PROC TransErr
WAIT "Warning: Transaction Fails"
TransErr=.t.
```

Portability
Not supported in dBASE IV or dBASE III PLUS. COMMIT( ) replaces the COMMIT command in dBASE IV.

See Also
BEGINTRANS( ), ROLLBACK( ), SET EXCLUSIVE

COMPILE — Programs

Compiles program files (.PRG, .WFM), creating object code files (.PRO, .WFO).

Syntax
COMPILE <*filename*> <*filename skeleton*>
[AUTO]
[LOG <*filename*>]
[TO <*response filename*>]

<filename> <filename skeleton> The file to compile. The <*filename skeleton*> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory. If you specify a file without including its extension, dBASE assumes .PRG.

AUTO The optional AUTO clause causes the compiler to detect automatically which files are
called by your program, and to compile those that have changed since the last compile. All .PRG and associated files must be in the current directory to use this option.

LOG <filename> LOG causes the compiler to write any error or warning messages to <filename>.

TO <response filename> The TO clause causes the compiler to create a response file containing the names of the
files output by the compiler (the object code files). If a specified source file cannot be successfully compiled, its name is included in the response file marked with an asterisk. The response file can be used by the BUILD command in the Visual dBASE compiler to link the object code files into an executable file.

Description

Use COMPILE to create compiled program files without executing or opening the files, or to compile only certain files. You can't run a program until it's been compiled. Because a compiled file can't be read or modified, compiling a program protects your source code from modification by users of the program. By default, dBASE creates compiled object files in the same directory as the source code files.

COMPILE has several advantages over compiling files with DO, SET PROCEDURE, or SET FORMAT:

- COMPILE doesn't execute or open the specified files.
- If you write an application that contains many program files, you can use COMPILE to compile only those program files you change rather than all the program files of the application. To specify a date and time range for the programs to be compiled, use FDATE( ) and FTIME( ).
- COMPILE *<filename skeleton>* lets you compile groups of unrelated or related files.

When you compile a program, dBASE detects any syntax errors in the source file and displays an error message corresponding to the error in a dialog box that contains three buttons:

- *Cancel* cancels compilation (equivalent to pressing *Esc*).
- *Ignore* cancels compilation of the program containing the syntax error but continues compilation of the rest of the files that match *<filename skeleton>* if you specified a skeleton.
- *Fix* lets you fix the error by opening the source code in an editing window, positioning the insertion point at the point where the error occurred.

See the on-line help for information about compiling dBASE programs into standalone executable files.

Portability

By default, both dBASE IV and dBASE III PLUS create compiled object files in the current directory rather than in the same directory as the source code files.

See Also

CLEAR PROGRAM, DO, FDATE( ), FTIME( ), SET COVERAGE, SET DEVELOPMENT, SET FORMAT, SET PROCEDURE

CREATE

Table basics

Opens the Table Designer to create or modify a table interactively.

Syntax

CREATE
[<*filename*> | ? | <*filename skeleton*>
[[TYPE] PARADOX | DBASE]
[EXPERT [PROMPT]]

<filename> | ? | <filename skeleton> The name of the table you want to create. CREATE ? and CREATE <*filename skeleton*> display a dialog box, in which you can specify the name of a new table. If you specify a table name without including its path, *Visual* dBASE saves the table to the current drive and directory. If you specify a table name without including an extension, *Visual* dBASE assigns a .DBF extension or the file extension of the table type you specified with SET DBTYPE. If you don't specify a name, the table remains untitled until you save the file. If you specify an existing table name, *Visual* dBASE asks whether you want to modify the existing table or overwrite it.

You can also create a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, :*database name*:*table name*. If the database is not already open, *Visual* dBASE displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

[TYPE] PARADOX | DBASE Specifies the type of table to create. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Specifying PARADOX creates a Paradox table with a .DB extension.

Specifying DBASE creates a dBASE table (the default). If you don't include an extension for <*filename*>, dBASE assigns a .DBF extension.

Description

CREATE opens the Table Designer, an interactive environment in which you can create or modify the structure of a table, or the Table Expert, a tool which guides you through the process of creating tables. The type of table you create or modify depends on your selection of the table type specified with the CREATE command, or with SET DBTYPE.

Create a table by defining the name, type, and size of each field. For more information on using the Table Designer, see the *User's Guide*.

Example

The following examples show several ways to use CREATE to design a table from the Command window:

```
CREATE MailList                && Opens table designer -.DBF table
CREATE MailList TYPE PARADOX
                               && Opens table designer - .DB table
CREATE ?                       && Opens dialog box for naming file
```

See Also
APPEND, APPEND MEMO, COPY STRUCTURE, DISPLAY STRUCTURE, LIST STRUCTURE,

CREATE FORM — Forms

Opens the Form Designer to create or modify a form.

Syntax
CREATE FORM
[<*filename*> | ? | <*filename skeleton*>]
[EXPERT [PROMPT]]

<filename> | ? | <filename skeleton> The form to create or modify. CREATE FORM ? and CREATE FORM <*filename skeleton*> display a dialog box, from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .WFM.

[EXPERT [PROMPT]] If the PROMPT clause is used, a dialog appears asking if you want to use the Form Designer or the Form Expert. You can then invoke either the designer or the expert. The EXPERT clause without PROMPT causes the Form Expert to be invoked.

Description
Use CREATE FORM to open the Form Designer or Form Expert and create or modify a form interactively. The Form Designer automatically generates dBASE program code that defines the contents and format of a form, and stores this code in an editable text file (.WFM).

CREATE SCREEN, CREATE APPLICATION and CREATE FORM are identical. For all these commands, the presence of a form file determines whether a create or modify operation occurs. If the .WFM file exists, the commands let you modify it in the Form Designer. If the file doesn't exist, the commands create a new file.

Since a .WFM file is a program file, you can edit it with MODIFY COMMAND.

See the Forms chapters in the *User's Guide* for instructions on using the Form Designer.

Note The Forms Designer is a Two-Way-Tool. You can open a form in the Form Designer even if you've edited the code in the .WFM file.

Example
The following examples open the Save File dialog box with the cursor positioned at the File Name block to name a new form. The picklist of .WFM files on the current directory is available if you desire to use an existing form name to create a new form. By contrast, MODIFY FORM ? would edit an existing form:

```
CREATE FORM ?
CREATE FORM *.WFM
```

CREATE FORM issued alone in the Command window will open an unnamed form design surface:

```
CREATE FORM
```

Portability
Not supported in dBASE IV or dBASE III PLUS.

See Also
CREATE APPLICATION, CREATE SCREEN, MODIFY APPLICATION, MODIFY FORM, MODIFY SCREEN, OPEN FORM

CREATE LABEL — Input/Output

Opens the Report Designer to create or modify a label file.

Syntax
CREATE LABEL
[<*filename*> | ? | <*filename skeleton*>]
[EXPERT [PROMPT]]

<filename> | ? | <filename skeleton> The label file to create or modify. The ? and <*filename skeleton*> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .RPL.

If dBASE can't find <*filename*>, it creates the file. By default, dBASE assigns an .RPL extension to <*filename*> and saves the file in the current directory.

CREATE LABEL without an option opens the empty Report Designer to create a new label file.

[EXPERT [PROMPT]] If the PROMPT clause is used, a dialog appears asking if you want to use the Report Designer or the Report Expert. You can then invoke either the designer or the expert. The EXPERT clause without PROMPT causes the Report Expert to be invoked.

Description
Use CREATE LABEL to open the Report Designer and create or modify a label file. A label file contains the information that formats labels. For information about using the Report Designer, see the Crystal Reports documentation. CREATE LABEL and MODIFY LABEL are identical commands.

Before issuing CREATE LABEL, you must have a default printer selected. After you create or modify the label file, use LABEL FORM to print the labels.

Example
This example opens a database table and then issues CREATE LABEL to construct a label form:

```
CLOSE DATABASE
USE Company
CREATE LABEL COMPLBL1
* If COMPLBL1.LBL already exists then it will be modified.
* Otherwise, a new label form will be created.
```

Portability
The *<filename skeleton>* option is not supported in dBASE IV or dBASE III PLUS. In dBASE IV and dBASE III PLUS, the default label file extension is .LBL.

See Also
CREATE REPORT, LABEL FORM

CREATE MENU — Forms

Opens the Menu Designer to create or modify a menu file.

Syntax
CREATE MENU
[<*filename*> | ? | <*filename skeleton*>]

<filename> | ? | <filename skeleton> The menu file to create or modify. CREATE MENU? and CREATE MENU *<filename skeleton>* display a dialog box, from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .MNU.

Description
Use CREATE MENU to design a menu for a form.

The menu you design is stored in a menu definition file (.MNU), which contains dBASE program code. You attach this program to a form via the form's Menu property.

For information on using the Menu Designer, see the *User's Guide*.

See Also
CLASS MENU

CREATE POPUP

Forms

Opens the Menu Designer to create or modify a popup menu file.

Syntax

CREATE POPUP
[<*filename*> | ? | <*filename skeleton*>]

<filename> | ? | <filename skeleton> The popup menu file to create or modify. CREATE POPUP ? and CREATE POPUP <filename skeleton> display a dialog box, from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .POP.

Description

Use CREATE POPUP to design a popup menu for a form. The menu you design is stored in a popup definition file (.POP), which contains dBASE program code. For information on using the Menu Designer, see UG_MENU.

See Also

CLASS MENU, CLASS POPUP

CREATE REPORT

Input/Output

Opens the Report Designer to create or modify a report file.

Syntax

CREATE REPORT
[CROSSTAB]
[<*filename*> | ? | <*filename skeleton*>]
[EXPERT [PROMPT]]

CROSSTAB Opens the Report Designer with the Cross-Tab dialog box displayed.

<filename> | ? | <filename skeleton> The report file to create or modify. The ? and <*filename skeleton*> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH.

If you specify a file without including its extension, dBASE assumes .RPT if you haven't specified CROSSTAB, and .RPC if you have. If dBASE can't find <*filename*>, it creates a file with the appropriate extension and saves it in the current directory.

CREATE REPORT without any options opens the empty Report Designer to create a new report.

[EXPERT [PROMPT]] If the PROMPT clause is used, a dialog appears asking if you want to use the Report Designer or the Report Expert. You can then invoke either the

designer or the expert. The EXPERT clause without PROMPT causes the Report Expert to be invoked.

Description

Use CREATE REPORT to open the Report Designer and create or modify a report file. A report file contains the information that formats a report. For information about using the Report Designer, see the Crystal Reports documentation. CREATE REPORT and MODIFY REPORT are identical commands.

Before issuing CREATE REPORT, you must have a default printer selected. After you create or modify the report file, use REPORT FORM to print the report.

Example

This example opens a database table and then issues CREATE REPORT to construct a report form:

```
CLOSE DATABASE
USE COMPANY
CREATE REPORT Comprep1
* If Comprep1 already exists then it will be modified.
* Otherwise, a new report form will be created.
```

Portability

The <*filename skeleton*> option is not supported in dBASE IV or dBASE III PLUS. In dBASE IV and dBASE III PLUS, the default report file extension is .FRM.

See Also

REPORT FORM

DBMESSAGE( ) — Error handling and debugging

Returns the error message of the last BDE error.

Syntax

DBMESSAGE( )

Description

DBMESSAGE( ) returns the error message of the most recent IDAPI error.

See the table in the description of ERROR( ) that compares ERROR( ), MESSAGE( ), DBERROR( ), DBMESSAGE( ), CERROR( ), SQLERROR( ), and SQLMESSAGE( ).

See online Help for a listing of all error messages.

Example

See DBERROR( ) for an example of using DBMESSAGE( ).

Portability
Not supported in dBASE IV or dBASE III PLUS.

See Also
CERROR( ), DBERROR( ), ERROR( ), MESSAGE( ), SQLERROR( ), SQLMESSAGE( )

EXTERN — Windows programming

Declares a prototype for a non-dBASE function contained in a DLL file.

Syntax
EXTERN [CDECL] <*return type*> <*function name*>
([<*parameter type list*>])
[<*path*>] <*filename*>

or

EXTERN [CDECL] <*return type*> <*user-defined function name*>
([<*parameter type list*>])
[<*path*>] <*filename*>
FROM <*export function name*> | <*ordinal number*>

Since you create a function prototype with EXTERN, parentheses are required as with other functions. Parentheses affect the way data types are promoted and converted.

CDECL Makes EXTERN use the C calling convention. If you omit CDECL, dBASE uses the Pascal calling convention. (See the following table.)

<function name> The export name of the function. The export name of an external function is contained in the .REF file associated with the DLL file that holds the function.

<return type>* and *<parameter type> A keyword representing the data type of the value returned by the function, and the data type of each argument you send to the function, respectively. The following table lists the keywords you can use.

Keyword	dBASE data type	C data type	Pascal data type	ASM data type
Parameters or return values				
CDOUBLE	Numeric	long double (80 bit)	Double	N/A
CHANDLE	Numeric	Handles, such as HANDLE, HWND, HFONT, HDC	Handles, such as Hwnd, HFont, HDC	dw
CINT	Numeric	int	Integer	dw (16 bit)
CLOGICAL	Logical	short Int	Integer	dw (16 bit)
CLONG	Numeric	long int (32 bit)	Long Int	dd (32 bit)
CSTRING	Character	char far * (zero terminated)	PChar	dw (16 bit)
CVOID	N/A	void	Procedure	N/A
CWORD	Numeric	short int (16 bit)	WORD	dw (16 bit)

Keyword	dBASE data type	C data type	Pascal data type	ASM data type
Parameters only				
CPTR	N/A	void *	Pointer	dd (32 bit)

For example, a C function may expect a 32-bit unsigned long value as a parameter, and return a char * string. In your EXTERN command, you specify CLONG as the parameter in *<parameter type list>* and CSTRING as the *<return type>*. When you call the function, you pass a dBASE numeric variable and store the returned value to a character variable.

<user-defined function name> The name you give to the external function instead of the export name. When you specify *<user-defined function name>* (instead of *<function name>*), you must use the FROM *<expC>* | *<expN>* clause to identify the function in the DLL file.

FROM *<export function name>* | *<ordinal number>* Identifies the function in the DLL file specified by *<filename>*. *<export function name>* identifies the function by its name, which is stored in the .DEF file that is associated with the DLL file. *<ordinal number>* identifies the function with a number, which is also stored in the .DEF file.

When the function you call does not return a value, specify CVOID for *<return type>*.

<filename> The name of the DLL file in which the external function is stored. This name must include the extension if the DLL file is not already in memory. The file name of any DLL that you load in memory must be unique; for example, you can't load SCRIPT.DLL and SCRIPT.FON into memory concurrently, even though they have different file-name extensions.

If the DLL file is not already loaded into memory, EXTERN loads it automatically. If the DLL file is already in memory, EXTERN increments the reference counter once. Therefore, it isn't necessary to execute LOAD DLL before using EXTERN.

The reference counter is incremented only the first time, regardless of how many times you execute the LOAD DLL and EXTERN commands.

<path> The directory path to the DLL file in which the external function is stored. When you omit *<path>*, dBASE looks in the following directories for the DLL by default:

1. The current directory.
2. The Windows directory (for example, C:\WINDOWS).
3. The Windows SYSTEM subdirectory (for example, C:\WINDOWS\SYSTEM).
4. The directory containing DBASEWIN.EXE, or the directory in which the .EXE file of your compiled program is located.
5. The directories in the current DOS path.
6. The directories mapped for search in a network.

The *<path>* specification is necessary only when the DLL file is not in one of these directories.

Description

Use EXTERN to declare a prototype for an external function written in a language other than dBASE. A prototype tells dBASE to convert its arguments to data types the external

function can use, and to convert the value returned by the external function into a data type dBASE can use.

To call an external DLL function, first prototype it with EXTERN. Then, using the name of the function you specified with EXTERN, call the function as you would any dBASE function. You must prototype an external function before you can call that function in your dBASE program.

The external function is held in a C library such as Windows API or a customized DLL file you create in C, Pascal, or ASM. (For more information on using EXTERN and DLL files, see Chapter 25 in the *Programmer's Guide*.) Although most library code is contained in files with extensions of .DLL, such code can be held in .EXE files, or even in .DRV or .FON files.

Example

The following example is a dBASE program that uses EXTERN to call a C program (cSample1.PRG). Explanations of included functions are as follows:

```
* Reverse():
*   Uses the FROM option to specify a new name for
*   the function (StrRevC) in the .DLL that has the
*   ordinal number 2.  In this example the var
*   myString is passed by reference, it is then
*   modified by the function StrRevC() in the dll
*   and the modified string is then displayed in
*   dBASE.
* GetDOSEnv():
*   This example shows returning a address of a
*   String to display in dBASE from a .dll, the
*   String is one of the DOS Environment strings.
* GetDOSEnvNum():
*   This example shows returning a number to dBASE
*   from a .dll, the number is the number of null
*   terminated strings in the DOS Environment. Use
*   this first to find how many strings to get from
*   GetDOSEnv().
* GetDOSEnvLen():
*   This example shows returning a number to dBASE
*   from a .dll, the number is the length of all the
*   strings in the DOS Environment.
************************************************
CLEAR
EXTERN CVOID   Reverse(CSTRING) cSample1.dll FROM 2
* 2 is the ORDINAL number in the DLL cSample.dll
EXTERN CSTRING GetDOSEnv(CWORD) cSample1.dll
EXTERN CWORD   GetDOSEnvLen()   cSample1.dll
EXTERN CWORD   GetDOSEnvNum()   cSample1.dll
myString = "AbCdE"
? "Original Str: ",myString
Reverse(myString)
? "From StrRevC: ",myString
nSize=GetDOSEnvLen()
? "The Size of the DOS Environment Sting is: ",nSize
?
```

```
? "The DOS Environment is:"
FOR i = 1 to GetDOSEnvNum()   && for 1 to Number of;
                                 Strings in DOS Env
   ? GetDOSEnv(i)             && print each String;
                                 from the DOS Env
NEXT
```

The C source code for creating the .DLL file is as follows: (cSample1.C)

```
#include <windows.h>
#include <string.h>
#pragma argsused
int FAR PASCAL LibMain(HINSTANCE hInstance,
  WORD wDataSeg, WORD wHeapSize, LPSTR lpszCmdline){
         if (wHeapSize > 0) UnlockData (0) ;
         return 1;}
#pragma argsused
int FAR PASCAL WEP(int wParameter) {
         return 1;}
/* ######################################
   ##  Function StrRevC()              ##
   ######################################*/
#pragma argsused
void FAR PASCAL StrRevC(LPSTR lpstrString) {
          strrev(lpstrString);}
/* ######################################
   ##  Functions related to  GetDOSEnv() ##
   ######################################*/
#pragma argsused
LPSTR FAR PASCAL GetDOSEnv(UINT index) {
        LPSTR p;
        UINT  i;
          for(i=1,p=GetDOSEnvironment();
            i<index && *p != '\0';
            p += (lstrlen(p)+1),++i);
          return p;}
#pragma argsused
UINT FAR PASCAL GetDOSEnvLen(void) {
        return (UINT)
        (GetDOSEnv((UINT)-1) - GetDOSEnv(1)) + 1;}
#pragma argsused
UINT FAR PASCAL GetDOSEnvNum(void) {
        LPSTR p;
        UINT  i;
        for(i=0,p=GetDOSEnvironment(); *p != '\0';
          p += (lstrlen(p)+1),++i);
        return i;}
```

The C source code for creating the .DEF file is as follows(cSample1.DEF):

```
LIBRARY        CSAMPLE1
DESCRIPTION    'A Sample DLL for dBASE for Windows'
EXETYPE        WINDOWS
CODE           PRELOAD MOVEABLE DISCARDABLE
DATA           PRELOAD SINGLE
HEAPSIZE       1400
```

```
EXPORTS
               WEP              @1
               STRREVC          @2
               GETDOSENV        @3
               GETDOSENVLEN     @4
               GETDOSENVNUM     @5
```

The following section demonstrates the FROM option of EXTERN (using the name of the function in the .DLL): This code segment displays a messagebox with an 'i' icon, title of 'MyBox', message of 'Hello World', and 2 buttons, 'OK' and 'Cancel'.

```
EXTERN CWORD MyBox(CHANDLE,CSTRING,CSTRING,CWORD);
  user.exe FROM "MessageBox"
? MyBox(0,"Hello World","MyBox",65)
```

Example 2: Calling a PASCAL dll. In this example the variable myString is passed by reference. It is then modified by the function StrRevP() in the dll and the modified string is returned.

```
* dBASE Program (pStrRev.PRG)
EXTERN CVOID StrRevP(CSTRING) pStrRev.dll
myString = "ABCD"
StrRevP(myString)
? myString
```

DLL Source Code (pStrRev.PAS)

```
Library pStrRev;
Uses
  Strings;
Function StrRevP(str : PChar) : PChar; Export;
Var
  endstr : PChar;
  ch     : Char;
Begin
  StrRevP := str;
  If Assigned(str) And (str^ <> #0) Then
    Begin
      endstr := StrEnd(str);
      Dec(endstr);
      While endstr > str Do
        Begin
          ch := str^;
          str^ := endstr^;
          endstr^ := ch;
          Inc(str);
          Dec(endstr)
        End
    End
End;
Exports
  StrRevP Index 1;
Begin
End.
```

Portability
Not supported in dBASE IV or dBASE III PLUS.

See Also
LOAD DLL, RELEASE DLL

FACCESSDATE( ) Windows95

Returns the last date a file was opened under Windows95.

Syntax

FACCESSDATE(<*filename expC*>)
<*filename expC*> The name of the file to check. If the file is not in the current directory, you must provide the path. If the file does not exist, dBASE returns an error message.

Description
FACCESSDATE( ) checks the file specified by <filename> and returns the date that the file was last opened, provided the file was last opened under the Windows95 operating system. This function is only useful on a system running the Windows95 operating system. Under Windows 3.1, FACCESSDATE( ) returns a blank date.

Example
The following example uses FACCESSDATE( ) to check the last opened date of a table:

```
? FACCESSDATE("C:\VISUALDB\SAMPLES\ANIMALS.DBF")
06/01/95
```

Portability
Not supported in dBASE IV or dBASE III PLUS.

See Also
FCREATEDATE( ), FCREATETIME( ), FDATE( ), FTIME( )

FCREATEDATE( ) Windows95

Returns the date a file was created under Windows95.

Syntax

FCREATEDATE(<*filename expC*>)
<*filename expC*> The name of the file to check. If the file is not in the current directory, you must provide the path. If the file does not exist, dBASE returns an error message.

Description
FCREATEDATE( ) checks the file specified by <filename> and returns the date that the file was created, provided the file was created under the Windows95 operating system. This function is only useful on a system running the Windows95 operating system. Under Windows 3.1, FCREATEDATE( ) returns a blank date.

Example
The following example uses FCREATEDATE( ) to check the creation date of a table:

```
? FCREATEDATE("C:\VISUALDB\SAMPLES\ANIMALS.DBF")
06/01/95
```

Portability
Not supported in dBASE IV or dBASE III PLUS.

See Also
FACCESSDATE( ), FCREATETIME( ), FDATE( ), FTIME( )

FCREATETIME( ) Windows95

Returns the time a file was created under Windows95.

Syntax
FCREATETIME(<*filename expC*>)

<filename expC> The name of the file to check. If the file is not in the current directory, you must provide the path. If the file does not exist, dBASE returns an error message.

Description
FCREATETIME( ) checks the file specified by <filename> and returns the time, as a character string, that the file was created, provided the file was created under the Windows95 operating system. This function is only useful on a system running the Windows95 operating system. Under Windows 3.1, FCREATETIME( ) returns an empty string.

Example
The following example uses FCREATEDATE( ) to check the creation time of a table:

```
? FCREATETIME("C:\VISUALDB\SAMPLES\ANIMALS.DBF")
12:20:10
```

Portability
Not supported in dBASE IV or dBASE III PLUS.

See Also
FACCESSDATE( ), FCREATEDATE( ), FDATE( ), FTIME( )

FNAMEMAX( ) Windows 95

Returns the maximum allowable file-name length on a given drive or volume.

Syntax
FNAMEMAX([<*expC*>]
<*expC*>
The drive letter, or name of the volume, to check. If <expC> is not provided, the current drive/volume is assumed. If the drive/volume does not exist, dBASE returns an error message.

Description
FNAMEMAX( ) checks the drive or volume specified by <expC> and returns the maximum file-name length allowed for files on that drive/volume. This function is only useful on a system running the Windows 95 operating system. Under Windows 3.1, FNAMEMAX( ) always returns 12.

Example
The following example uses FNAMEMAX( ) to determine the maximum allowable file-name

```
length on drive C:
? FNAMEMAX("C")
255
```

Portability
Not supported in dBASE IV or dBASE III PLUS.

See Also
FSHORTNAME( )

FSHORTNAME( ) Windows95

Returns the short name (i.e. the DOS compatible name) of a file created under Windows95.

Syntax
FSHORTNAME(<*filename expC*>)

<filename expC> The name of the file to check. If the file is not in the current directory, you must provide the path. If the file does not exist, dBASE returns an error message.

Description
FSHORTNAME( ) checks the file specified by <filename> and returns a name for the file following the DOS file naming convention (eight character file name, three character extension). If SET FULLPATH is ON, the path is also returned. This function is only

useful on a system running the Windows95 operating system. Under Windows 3.1, FSHORTNAME( ) returns the file-name.

Example

The following example uses FSHORTNAME( ) to check the short name of a table:

```
? FSHORTNAME("ANIMAL_LISTINGS_TABLE.DBF")
ANIMAL~1.DBF
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

FNAMEMAX( )

LABEL FORM

Input/Output

Generates and displays or prints a label report, using the label format stored in a specified label file and information derived from records in the current table.

Syntax

LABEL FORM <*filename 1*> | ? | <*filename skeleton 1*>
[<*scope*>] [FOR <*condition 1*>] [WHILE <*condition 2*>]
[SAMPLE]
[TO FILE <*filename 2*> | ? | <*filename skeleton 2*>] | [TO PRINTER]

<*filename* 1> | ? | <*filename skeleton*> The file to get label formats from. The ? and <*filename skeleton*> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE looks for .RPL, .LBG, or .LBL, in that order.

<*scope*> The number of records in the current table from which to derive labels. RECORD <*n*> identifies a single record by its record number. NEXT <*n*> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <*condition 1*>
WHILE <*condition 2*> Determines which records are affected by LABEL FORM. FOR restricts LABEL FORM to records that meet <*condition 1*>. WHILE starts processing with the current record and continues with each subsequent record as long as <*condition 2*> is true.

SAMPLE Displays or prints a label containing asterisks for text and then prompts you for more samples.

TO FILE <*filename 2*> | ? | <*filename skeleton 2*> Directs output to the text file <*filename*>. By default, dBASE assigns a .TXT extension to <*filename*> and saves the file in the current directory. The ? and <*filename skeleton*> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER Directs output to the printer.

Description

Use LABEL FORM to print or display labels in a format that you've defined in the Report Designer using CREATE LABEL or MODIFY LABEL. For information about using the Report Designer, see the Crystal Reports documentation. If you don't specify a <*scope*>, WHILE <*condition 1*>, or FOR <*condition 2*> option, LABEL FORM prints the label specifications for each record in record number or index order.

When printing or displaying a label report that includes groups of data or group subtotals, either the current table must be in sorted order or its master index must be in use. The sorted file or index must be arranged according to the value of the field on which the data is grouped.

LABEL FORM without the TO FILE or TO PRINTER options displays the labels in the results pane of the Command window or current user-defined window.

Example

This example opens the Company database and then generates labels using the Complbl1 label form:

```
CLOSE DATABASE
USE Company
LABEL FORM Complbl1 TO PRINT
* This label format produces one across labels,
* 6 lines per label, with Company, Street,
* City, State and Zip code:
* General Consolidated
* 35 Libra Plaza
* Nashua  NH              09242
*
*
*
* Consolidated Brands, Inc.
* 3 Independence Parkway
* Rivendell  CA           93456
```

Portability

The <*filename skeleton*> option is not supported in dBASE IV or dBASE III PLUS.

See Also

CREATE LABEL

LOGOUT — Security

LOGOUT logs out the current user and sets up a new log-in dialog.

Syntax

LOGOUT

Description

LOGOUT logs out the current user from the current session and sets up a new log-in dialog when used with PROTECT. The LOGOUT command enables you to control user sign-in and sign-out procedures. The command forces a logout and prompts for a login.

When the command is processed, a log-in dialog appears. The user can enter a group name, log-in name, and password. The PROTECT command establishes log-in verification functions and sets the user access level.

LOGOUT closes all open tables, their associated files, and program files.

If PROTECT has not been used, and no DBSYSTEM.DB file exists, the LOGOUT command is ignored.

See also

PROTECT, QUIT

OPEN DATABASE

Table basics

Establishes a connection to a database server or a database defined for a specific directory location.

Syntax

OPEN DATABASE <*database name*>
[AUTOEXTERN]
[LOGIN <*username*>/<*password*>]
[WITH <*option string*>]

<database name> The name of the database you want to open. Databases are created using the BDE Configuration Utility (see *Getting Started* for more information).

<user name>/<password> Character string specifying the user name and password combination required to access the database.

WITH *<option string>* Character string specifying server-specific information required to establish a database server connection. For information about establishing database server connections, refer to your Borland SQL Link documentation, and contact your network or database administrator for specific connection information.

Description

The OPEN DATABASE command is used to establish a connection with a database defined with the BDE Configuration Utility. When opening a database, you need to specify whatever login parameters and database-specific information that connection requires. Typically, your network or system administrator can provide you with the information necessary to establish connections to established databases and database servers at your site.

Example

The following example uses OPEN DATABASE to establish a connection with a database server, opens a database previously created using the BDE Configuration Utility with SET DATABASE TO, and opens a server dBASE table and appends it to the client/server database:

```
CLEAR
SET DBTYPE TO DBASE
OPEN DATABASE CAClients      && Establish connection with database server
USE CAClients IN 1           && Opens server table
SELECT 1                     && Work area 1 active
APPEND FROM CLIENTS.DBF      && Append from local dBASE table
CLOSE ALL                    && Closes Clients server table
CLOSE DATABASE CAClients     && Disconnect from database server
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CLOSE..., DATABASE( ), SET DATABASE, SET DBTYPE

PROTECT — Security

Creates and maintains security on a dBASE system.

Syntax

PROTECT

Description

This command is issued within dBASE by the database administrator, who is responsible for data security. PROTECT works in a single user or multiuser environment.

PROTECT is optional. If you use it, however, the security system always controls dBASE table access.

This command displays a multi-page dialog. The first time you use protect, the system prompts you to enter and confirm an administrator password.

Warning Remembering the administrator password is essential. You can access the security system only if you can supply the password. Once established, the security system can be changed only if you enter the administrator password when you call PROTECT. Keep a hard copy of the database administrator password in a secured area. There is no way to retrieve a password from the system.

PROTECT includes three distinct types of database protection:

- Log-in security, which prevents access to dBASE, or all protected tables (at the discretion of the database administrator), by unauthorized personnel.

- File and field access security, which allows you to define what dBASE tables, and fields within tables, each user can access.
- Data encryption, which encrypts dBASE tables so that unauthorized users cannot read them

The following table summarizes the database security types, how to implement each security type, and the results of security implementation.

Security Type	You Define:	You Get:
Log-in	User name and password	Control over access to dBASE or all protected tables
File and Field Access	Access levels	Control over access to dBASE tables, fields in tables, and application code
Data Encryption	User and file group	Automatic encryption and decryption of data

It is not necessary to implement all three levels of security; you can stop at the log-in level if you wish. You must implement the security types in the order shown in the previous table.

Log-in security is the first security level. Once a security system is in place, you can set up log-in security in one of two ways:

- Users cannot access dBASE until they pass log-in security.
- Users can access dBASE, but cannot access any protected table until they pass log-in security.

Access control is the next security level. Access control determines what a user can do both with the table and the data in the table, and can be used to control processing of application code. *User access levels* are numbered 1 through 8, where 1 has the greatest and 8 has the lowest access privileges. You establish an access level for each user in the user's profile, and additional access levels for table and field privileges in the *table privilege scheme*.

You establish privileges for a table by assigning access levels, in any combination, for read, update, extend, and delete operations.

Data encryption scrambles the table so that unauthorized users cannot read the data.

The DBSYSTEM.DB file PROTECT builds and maintains a password system file called DBSYSTEM.DB, which contains a record for each user who accesses a PROTECTed system. Each record, called a user profile, contains the user's log-in name, account name, password, group name, and access level. When a user attempts to start dBASE (if dBASE is configured to require a log-in to start the program), or attempts to access a protected table (if dBASE is configured to require a log-in when a protected table is accessed), dBASE looks for a DBSYSTEM.DB file. You can specify a location for this file in the [CommandSettings] section of DBASEWIN.INI:

```
DBSYSTEM=C:\VISUALDB\BIN
```

If there is no DBSYSTEM entry in DBASEWIN.INI, dBASE looks for the file in the same directory in which DBASEWIN.EXE is located. If it finds the file, it initiates the log-in process. If it does not find the file, there is no log-in process.

DBSYSTEM.DB is maintained as an encrypted file. Keep a record of the information contained in DBSYSTEM.DB, as well as a current backup copy of the file. If the DBSYSTEM.DB file is deleted or damaged and no backup is available, the database administrator will need to reinitialize PROTECT using the same administrator password and group names as before, or the data will be unrecoverable.

See the "Restricting access to confidential tables" chapter in the *User's Guide* for more information about PROTECT.

See Also
ACCESS( ), LOGOUT, SET ENCRYPTION, USER( )

REPORT FORM

Input/Output

Generates and displays or prints a report, using the report format stored in a specified report file and information derived from records in the current table.

Syntax
REPORT FORM <*filename 1*> | ? | <*filename skeleton 1*>
[<*scope*>] [FOR <*condition 1*>] [WHILE <*condition 2*>]
[CROSSTAB]
[HEADING <*expC*>]
[NOEJECT]
[PLAIN]
[SUMMARY]
[TO FILE <*filename 2*> | ? | <*filename skeleton 2*>] | [TO PRINTER]

<filename 1> | ? | <filename skeleton> The report format file to use. The ? and <*filename skeleton*> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE looks for .RPT, .FRG, or .FRM, in that order. If you specify CROSSTAB, dBASE looks for .RPC, .FRG, or .FRM, in that order.

<scope> The number of records to derive the report from. RECORD <*n*> identifies a single record by its record number. NEXT <*n*> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <*condition 1*>
WHILE <*condition 2*> Determines which records are affected by REPORT FORM. FOR restricts REPORT FORM to records that meet <*condition 1*>. WHILE starts processing with the current record and continues with each subsequent record as long as <*condition 2*> is true.

CROSSTAB Specifies that the report was created with the Cross-Tab dialog box.

HEADING <*expC*> Adds a character expression, <*expC*>, as the heading of each page. HEADING has no effect if used with PLAIN.

NOEJECT Prevents a page feed before printing begins.

PLAIN Suppresses page numbers and dates on the pages of the report. Any title appears only on the first page.

SUMMARY Includes only the totals of groups and subtotals of subgroups, excluding the individual contents of records within each group and subgroup. The report format defines groups and subgroups.

TO FILE *<filename 2>* | ? | *<filename skeleton 2>* Directs output to the text file *<filename 2>*. By default, dBASE assigns a .TXT extension to *<filename 2>* and saves the file in the current directory. The ? and *<filename skeleton>* options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER Directs output to the printer.

Description

Use REPORT FORM to print or display reports in a format that you've defined in the Report Designer using CREATE REPORT or MODIFY REPORT. For information about using the Report Designer, see the Crystal Reports documentation. If you don't specify a *<scope>*, WHILE *<condition 1>*, or FOR *<condition 2>* option, REPORT FORM prints the report specifications for each record in record number or index order.

When printing or displaying a report that includes groups of data or group subtotals, the current table must either be in sorted order or its master index must be in use. The sorted file or index must be arranged according to the value of the field on which the data is grouped.

REPORT FORM without the TO FILE or TO PRINTER options displays the report in the Command window or current user-defined window.

Example

This example opens the Company database and then generates a report using the Comprep1 report definition:

```
CLOSE DATABASE
USE Company
REPORT FORM Comprep1 TO PRINT
```

Portability

The *<filename skeleton>* option is not supported in dBASE IV or dBASE III PLUS.

See Also

CREATE REPORT

RESTORE IMAGE

Displays an image stored in a file or a binary field.

Syntax

RESTORE IMAGE FROM
<*filename*> | ? | <*filename skeleton*> | BINARY <*binary field*>
[TIMEOUT <*expN*>]
[TO PRINTER]
[[TYPE] PCX | TIF[F] | ICO | WMF | EPS]

FROM <*filename*> | ? | <*filename skeleton*> | BINARY <*binary field*> Identifies the file or binary field to restore the image from. RESTORE IMAGE FROM ? and RESTORE IMAGE FROM <*filename skeleton*> display the Open Source File dialog box, which lets the user select a file. <*filename*> is the name of an image file; RESTORE IMAGE assumes a .BMP extension and file type unless you specify otherwise. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. RESTORE IMAGE FROM BINARY <*binary field*> displays the image stored in a binary field. You store an image in a binary field with the REPLACE BINARY command.

TIMEOUT <*expN*> Specifies the number of seconds the image is displayed onscreen.

TO PRINTER Sends the image to the printer as well as to the screen.

[TYPE] PCX Specifies an image stored in PCX format, and assumes a .PCX file-name extension if none is given. The word TYPE is optional.

Description

Use RESTORE IMAGE to display a graphic image that was generated and saved in bitmap or PCX format. The image is displayed in a window.

Notes Your computer must have a graphics adapter to display an image.

To print an image with the TO PRINTER option, you must have a printer that can process and print graphic data.

Example

The following example defines a form and list box for selection of an aircraft model and uses RESTORE IMAGE to display a graphic from the memo field Image of the selected record:

```
CLOSE ALL
SET TALK OFF
USE Aircrdb ORDER Aircraft IN SELECT()
SELECT Aircrdb
DEFINE FORM AC              ;
   PROPERTY                 ;
      Top 5,                ;
      Left 2,               ;
      Height 13,            ;
      Width 30,             ;
      Text "Aircraft",      ;
```

```
      Sizeable .T.,          ;
      OnSelection Photo
DEFINE LISTBOX Model OF AC             ;
   PROPERTY                            ;
      Top 2,                           ;
      Left 10,                         ;
      Height 7,                        ;
      Width 18,                        ;
      DataSource "FIELD Aircrdb->Aircraft"
OPEN FORM AC

FUNCTION Photo
RESTORE IMAGE FROM BINARY Image
RETURN .T.
```

Portability
Not supported in dBASE IV or dBASE III PLUS.

ROLLBACK( ) — Shared data

Ends a transaction initiated by BEGINTRANS( ) without saving any changes to the open files. Returns .T. if the data was rolled back successfully.

Syntax
ROLLBACK([<*database name expC*>])

<database name expC> The name of the database in which to cancel the transaction.

- If you began the transaction with BEGINTRANS(<*database name expC*>), you must issue ROLLBACK(<*database name expC*>). If instead you issue ROLLBACK( ), dBASE ignores the ROLLBACK( ) statement.
- If you began the transaction with BEGINTRANS( ), <*database name expC*> is an optional ROLLBACK( ) argument. If you include it, it must refer to the same database as the SET DATABASE TO statement that preceded BEGINTRANS( ).

Description
Use ROLLBACK( ) to end the open transaction and restore any open files to the state they were in when BEGINTRANS( ) was issued. To end a transaction and write changes to the files, use COMMIT( ).For more information on transactions, see BEGINTRANS( ).

Example
The following example begins a transaction with BEGINTRANS( ). It opens a multi-user version of Company.dbf and attempts to set values in the Ytd_Sales field to 0. ON ERROR detects any error which might occur. In particular, it will detect if another user has locked any record in Company.dbf. If an error occurs, ROLLBACK( ) resets all values. Otherwise COMMIT( ) writes the changes to disk:

```
CLOSE ALL
SET PROCEDURE TO PROGRAM(1) ADDITIVE
SET EXCLUSIVE OFF

BEGINTRANS()

TransErr=.f.
ON ERROR DO TransErr    && Activates ON ERROR trap

USE L:\MultiUse\Company
REPLACE ALL Ytd_Sales WITH 0
ON ERROR                && Disables ON ERROR

IF TransErr
  ? "Rollback"
  ROLLBACK()            && restore data
ELSE
  ? "Commit"
  COMMIT()              && save changes
ENDIF

PROC TransErr
WAIT "Warning: Transaction Fails"
TransErr=.t.
```

Portability
Not supported in dBASE IV or dBASE III PLUS. ROLLBACK( ) replaces the ROLLBACK command in dBASE IV.

See Also
BEGINTRANS( ), COMMIT( ), SET EXCLUSIVE

SET ENCRYPTION — Security

Establishes whether a newly created dBASE table is encrypted if PROTECT is used.

Syntax
SET ENCRYPTION ON | off

Default
The default for SET ENCRYPTION is ON.

Description
This command determines whether copied dBASE tables (that is, tables created through the COPY, JOIN, and TOTAL commands) are created as encrypted tables. An encrypted table contains data encrypted into another form to hide the contents of the original table. An encrypted table can only be read after the encryption has been deciphered or copied to another table in decrypted form.

To access an encrypted table, you must enter a valid user name, group name, and password after the login screen prompts. Your authorization and access level determine whether you can or cannot copy an encrypted table. After you access the table, SET ENCRYPTION OFF to copy the table to a decrypted form. You need to do this if you wish to use EXPORT, COPY STRUCTURE EXTENDED, MODIFY STRUCTURE, or options of the COPY TO command.

Note Encryption works only with dBASE (.DBF) tables. Encryption works only with PROTECT. If you do not enter dBASE or access the table through the log-in screen, you will not be able to use encrypted tables.

All encrypted tables used concurrently in an application must have the same group name.

Encrypted tables cannot be JOINed with unencrypted tables. Make both tables either encrypted or unencrypted before JOINing them.

You can encrypt any newly created table by assigning the table an access level through PROTECT.

See also
COPY TO, PROTECT, SET( )

SET LDCONVERT — Environment

Determines whether data read from and written to character and memo fields is transliterated when the table character set does not match the global language driver.

Syntax
SET LDCONVERT ON | off

Default
The default for SET LDCONVERT is ON. To change the default, set the LDCONVERT parameter in DBASEWIN.INI.

Description
Use SET LDCONVERT to determine whether the contents of character and memo fields in tables created with a given language driver in effect, are converted to match the language driver in effect at the time the fields are read or written to.

Language drivers determine the character set and sorting rules that dBASE uses, so if you create a dBASE table with one language driver and then use that file with a different language driver, some of the characters will appear incorrectly and you may get incorrect results when querying data.

In general, SET LDCONVERT should be ON to insure that dBASE behaves as expected when using data created under different language drivers.

For more information about working with language drivers, see PG_CHARLANG.

Portability
Not supported in dBASE IV or dBASE III PLUS.

See Also
CHARSET( ), LDRIVER( ), SET LDCHECK

USER( ) Security

Returns the login name of the user currently logged in to a protected system.

Syntax
USER( )

Description
The USER( ) function returns the log-in name used by an operator currently logged in to a system that uses PROTECT to encrypt files. On a system that does not use PROTECT, USER( ) returns a null string.

See Also
ACCESS( ), PROTECT

Classes

CLASS ASSOCARRAY

An array object class that takes character strings as subscripts.

Properties
The following table lists the properties of the Assocarray class. For more information on each property, see the Properties section that follows.

Property	Default	Description
ClassName	ASSOCARRAY	Identifies the assocarray class
Count( )	N/A	Returns the number of elements in the associated array
FirstKey	N/A	Returns the subscript character string for an element of an associated array
IsKey( )	N/A	Returns .T. if the specified character expression is a subscript of the associated array
NextKey( )	N/A	Returns the subscript of the next element in the associated array
RemoveAll( )	N/A	Deletes all elements of the associated array.
RemoveKey( )	N/A	Deletes a specified element from the associated array

Description

Use the ASSOCARRAY class to create an array that has character strings as subscripts. This lets you assign meaningful information to both the subscript and the array element it references. For more information on using array objects, see ARRAYXREF.

Use the standard array operator [ ] to add and reference items in the array. An empty string "" can be used as a subscript.

Example

The following example creates an associated array and displays its subscripts and contents. It then deletes a specified element from the array.

```
aa = NEW ASSOCARRAY()
aa["San Francisco"] = "49ers"
aa["Los Angeles"] = "Rams"
x = aa.FirstKey
DO WHILE .NOT. EMPTY(x)
   ? x, aa[x]               && display element subscript and contents
   x = aa.NextKey(x)      && 'increments' index pointer
ENDDO
? aa.Count()   && Returns 2
aa.RemoveKey("San Francisco")&& Removes element from the array
? aa.Count()   && Returns 1
```

See Also

CLASS ARRAY

CLASS MENUBAR

A MenuBar object specifies a top-level menu for a form. Using the MENUBAR class lets you add the standard Windows Edit and Windows pull-down menus to a form.

Properties

The following table lists the properties of the Menubar class. For more information on each property, see the Properties section that follows.

Property	Default	Description
ClassName	MENUBAR	Identifies the menubar object's class
EditCopyMenu	.F.	Specifies a menu item that copies selected text from a control to the Windows clipboard
EditCutMenu	.F.	Specifies a menu item that deletes selected text from a control and copies it to the Windows clipboard
EditPasteMenu	.F.	Specifies a menu item that pastes text from the Windows clipboard to the edit control with focus
EditUndoMenu	.F.	Specifies a menu item that restores the form to the state before the last edit operation was performed
ID	1	Identifies the menubar object with a numeric value
Name	MENUBAR1	Specifies the menubar object's name

Property	Default	Description
OnInitMenu	N/A	Specifies code that executes when the menubar is accessed
Parent	N/A	An object reference that points to the parent form
Release( )	N/A	Removes the MenuBar definition from memory
WindowMenu	.F.	Specifies a top-level menu that displays the Window List of all open MDI windows

Description

A Menubar object specifies a top-level menu for a form. A form's top-level menu doesn't contain any menu prompts itself; it is only the container for child menu objects. The child menu objects of the top-level menu contain the form's actual menu items.

Menu objects that have a Menubar as a Parent appear on the top line of a form. By default, the Menu Designer creates a MenuBar subclass when creating a .MNU file.

You can design and implement menus without using Menubars, as in earlier versions of dBASE. The advantage of using the MENUBAR class is that you can implement an Edit pulldown that uses the Windows clipboard for Cut, Copy, Paste and Undo operations, and you can implement a Window pulldown that offers the standard MDI window list. The properties that enable these menu choices (EditCutMenu, EditCopyMenu, etc.) all take an object reference to a Menu object as their value.

To quickly add the Edit and Windows menus and their dropdown options (Cut, Copy, etc.), use the Menu Designer and add these options using the Menu pulldown menu.

Note The command CREATE MENU creates a Menubar subclass by. You can also use DEFINE MENUBAR m OF FormX to create a menu bar for the form named FormX.

Example

```
** END HEADER -- do not remove this line*
* Generated on 03/31/95
*
Parameter FormObj
NEW FOOMENU(FormObj,"Root")
CLASS FOOMENU(FormObj,Name) OF MENUBAR(FormObj,Name)
   DEFINE MENU FILE OF THIS;
       PROPERTY;
         Text "&File"
         DEFINE MENU EXIT OF THIS.FILE;
             PROPERTY;
               Text "E&xit"
   DEFINE MENU EDIT OF THIS;
       PROPERTY;
         Text "&Edit"
         DEFINE MENU UNDO OF THIS.EDIT;
             PROPERTY;
               Text "&Undo"
         DEFINE MENU CUT OF THIS.EDIT;
             PROPERTY;
               Text "Cu&t"
         DEFINE MENU COPY OF THIS.EDIT;
             PROPERTY;
```

```
            Text "&Copy"
        DEFINE MENU PASTE OF THIS.EDIT;
            PROPERTY;
              Text "&Paste"
    DEFINE MENU WINDOW OF THIS;
        PROPERTY;
          Text "&Window"
          DEFINE MENU ARRANGE OF THIS.WINDOW;
              PROPERTY;
                Text "&Arrange"
    DEFINE MENU HELP OF THIS;
        PROPERTY;
          Text "&Help"
          DEFINE MENU ABOUT OF THIS.HELP;
              PROPERTY;
                Text "&About"
    This.EditUndoMenu  = This.Edit.Undo
    This.EditCutMenu   = This.Edit.Cut
    This.EditCopyMenu  = This.Edit.Copy
    This.EditPasteMenu = This.Edit.Paste
    This.WindowMenu    = This.Window
ENDCLASS
```

See Also
CLASS MENU, CLASS POPUP, DEFINE

CLASS OLEAUTOCLIENT

Creates an OLE2 controller which attaches to an OLE2 server.

Properties
The properties of this class are determined by the server.

Description
Use CLASS OLEAUTOCLIENT to attach to a server program. The syntax is:

```
<ClientClassName> = NEW OLEAUTOCLIENT<exp>
```

where <exp> is the server program ID. There is no equivalent DEFINE OLEAUTOCLIENT statement.

After you have created the class, you can use the Property Inspector to see its properties. You can change properties by using the Inspector or by using standard ClientClassName.property statements.

Example

```
*
*  OLEWORD.PRG
*  Sample program to illustrate OLE2 Automation
*  with Microsoft Word as the server.
*

*
*  Create OLE2 Automation object. The parameter
*  is the ProgID
*
ww = new oleautoclient("word.basic")
*
*  All properties and methods of the OLE2
*  Automation object are documented by the
*  server.
*
ww.FileNew("Normal", 0)
ww.Insert("This is my configuration file")
ww.InsertBreak(6)
ww.InsertBreak(6)
ww.InsertFile("c:\config.sys")
ww.StartOfDocument()
ww.EndOfLine(1)
ww.EditCut()
ww.EditPaste()
ww.EditPaste()
? "current font size is", ww.FontSize()
ww.EditSelectAll()
ww.GrowFont()
ww.GrowFont()
ww.GrowFont()
? "font size is now ", ww.FontSize()
ww.EditCopy()      && Can paste into dBASE later
ww.FilePrint()
*
* Uncomment the following line to
* close Word
*ww.AppClose()
```

See Also

CLASS DDELINK, CLASS DDETOPIC, CLASS OLE

CLASS PAINTBOX

A generic control that can be placed on a form.

Properties

The following table lists the properties of the Paintbox class. For more information on each property, see the Properties section of this chapter.

Property	Default	Description
Before	N/A	Specifies which object the paintbox object precedes in the tabbing order of the parent form
ClassName	PAINTBOX	Identifies the paintbox object's class
ColorNormal	WindowText/ Window	Sets the color of the paintbox object when it isn't highlighted
Enabled	.T.	Determines if the paintbox object can be selected
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
hWnd	N/A	Returns the paintbox object handle
ID	-1	Identifies the paintbox object with a numeric value
Left	N/A	Sets the position of the left border
Move( )	N/A	Moves or sizes the paintbox object
Name	PAINTBOX1	Specifies the paintbox object's name
OnChar	N/A	Executes a subroutine when a "printable" key or key combination is pressed
OnClose	N/A	Executes a subroutine when a form is closed
OnFormSize	N/A	Executes a subroutine whenever the parent form is resized, restored, or maximized
OnGotFocus	N/A	Executes a subroutine when the paintbox object receives focus
OnKeyDown	N/A	Executes a subroutine when any key is pressed
OnKeyUp	N/A	Executes a subroutine when any key is released
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the paintbox object
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the paintbox object with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the paintbox object
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDblClick	N/A	Executes a subroutine when the user double-clicks the paintbox object with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the paintbox object with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the paintbox object

Property	Default	Description
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse over the paintbox object
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnPaint	N/A	Executes a subroutine whenever the object needs to be redrawn
OnRightDblClick	N/A	Executes a subroutine when the user double-clicks the paintbox object with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the paintbox object with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the paintbox object
PageNo	N/A	Specifies on which page of a multi-page form the paintbox object appears
Parent	N/A	An object reference that points to the parent form
Release( )	N/A	Removes the paintbox object definition from memory
SetFocus( )	N/A	Gives focus to the paintbox object
TabStop	.T.	Determines if the user can give object focus to the paintbox object by pressing Tab or Shift+Tab
Top	N/A	Sets the position of the top border
Visible	.T.	Determines whether the paintbox object is visible or hidden
Width	N/A	Sets the width

Description

The PaintBox object is a generic control you can use to create a variety of objects. It is designed for advanced developers who want to create their own custom controls using the Windows API. It is simply a rectangular region of a form which has all the standard control properties such as Height, Width, and Before, as well as all the standard mouse events.

In addition to the standard events or properties, the PaintBox object has three events that let you detect keystrokes entered when it has focus: OnChar, OnKeyDown, and OnKeyUp. These let you create customized editing controls. The OnPaint and OnFormSize properties let you modify the appearance of the object based on user interaction.

Example

```
local f
f = new PAINTEXFORM()
f.Open()
CLASS PAINTEXFORM OF FORM
   this.OnLeftMouseUp = CLASS::FORM_ONLEFTMOUSEUP
   this.Text = "Form"
   this.Left = 54.5
   this.Top = 2
   this.PageNo = 1
   this.ColorNormal = "N/BTNFACE"
   this.Height = 20.6465
   this.TopMost = .F.
```

```
   this.Width = 67.666
   DEFINE PAINTBOX PAINTBOX1 OF THIS;
       PROPERTY;
         OnPaint CLASS::PAINTBOX1_ONPAINT,;
         OnLeftMouseDown CLASS::PAINTBOX1_ONLEFTMOUSEDOWN,;
         OnLeftMouseUp CLASS::PAINTBOX1_ONLEFTMOUSEUP,;
         Left 9.333,;
         Top 2,;
         ColorNormal "B+/0xffff80",;
         PageNo 1,;
         Height 6.8818,;
         OnOpen CLASS::PAINTBOX1_ONOPEN,;
         Width 21.333
   Procedure PAINTBOX1_OnPaint
       hfact = (256/43)
       vfact = (256/15.5)
       lwidth = form.paintbox1.width * hfact
       lheight = form.paintbox1.height * vfact
       form.pointarray = makepoint(lwidth/2,0)
       form.pointarray = form.pointarray + makepoint(0,lheight)
       form.pointarray = form.pointarray + makepoint(lwidth,lheight)
       local hDC
       hDC = GetDc(this.hwnd)
       hBrush = CreateSolidBrush(hdc,RGB(255,0,0))
       SelectObject(hDC,hbrush)
       SetTextColor(hDC, RGB(0,0,255))
       SetPolyFillMode(hDC,2)
       Polygon(hDC,form.pointarray,3)
       ReleaseDc(this.hWnd, hDC)
   return

   Procedure PAINTBOX1_OnOpen
      set proc to program(1) ADDITIVE

      EXTERN CHANDLE  GetDc(CHANDLE) USER.EXE
      EXTERN CINT     ReleaseDc(CHANDLE,CHANDLE) USER.EXE
      EXTERN CLOGICAL Polygon(CHANDLE,CPTR,CINT) GDI.EXE
      EXTERN CINT     SetTextColor(CHANDLE, CLONG) GDI.EXE
      EXTERN CLOGICAL Ellipse(CHANDLE,CINT,CINT,CINT,CINT) GDI.EXE
      EXTERN CLOGICAL FloodFill(chandle,cint,cint,clong) GDI.EXE
      EXTERN CINT     SetPolyFillMode(chandle,cint) GDI.EXE
      EXTERN Chandle  CreateSolidBrush(chandle,clong) GDI.EXE
      EXTERN Chandle  SelectObject(chandle,chandle) GDI.EXE
      this.moving = .f.
   return

   Procedure PAINTBOX1_OnLeftMouseDown(flags, col, row)
       this.moving = .t.
   return

   Procedure form_OnLeftMouseUp(flags, col, row)
       if form.paintbox1.moving

form.paintbox1.move(col,row,form.paintbox1.width,form.paintbox1.height)
          form.paintbox1.moving = .f.
```

```
        endif
    return

    Procedure PAINTBOX1_OnLeftMouseUp(flags, col, row)
        this.moving = .f.
    return
ENDCLASS
function RGB(r, g, b)
return b*65536+g*256+r
function MakePoint(x,y)
return(MakeInt(x) + MakeInt(y))
function MakeRect(left, top, width, height)
return
MakeInt(left)+MakeInt(top)+MakeInt(width+left)+MakeInt(top+height)
function MakeInt(int)
return chr(bitand(int,255))+chr(bitr(int,8))
```

See Also

CLASS IMAGE, CLASS OBJECT

CLASS POPUP

A Windows-style popup menu assigned to a form.

Properties

The following table lists the properties of the Popup class. For more information on each property, see the Properties section of this chapter.

Property	Default	Description
ClassName	POPUP	Identifies the Popup class
ID	1	Identifies the popup with a numeric value
Left	N/A	Sets the position of the left border
Name	POPUP1	Specifies the name of the popup menu
OnInitMenu	N/A	Specifies code that executes when the popup menu is opened
Open( )	N/A	Opens the popup menu
Parent	N/A	An object reference that points to the parent form
Release( )	N/A	Removes the popup definition from memory
Top	N/A	Sets the position of the top border
TrackRight	.T.	Determines whether the popup menu responds to a right mouse click for selection of a menu item

Description

Use CLASS POPUP to add a popup menu to a form. Popup menus give users a "shortcut" way to perform actions without pulling down items from the menu bar. A Popup's parent is always a Form. Individual menu items are attached to a Popup by defining Menu objects with the Popup as parent.

A .POP file is similar to a .MNU in format with the name of the Popup passed as a parameter instead of the explicit "Root" used in .MNU files.

A popup menu consists of two elements:

- The object reference variable that identifies the entire popup menu. You must create this variable before you can create the menu. (The variable name is not displayed anywhere.)
- Menu items, the prompts offered by the popup menu.

When you create a popup menu with the NEW operator, you can specify two parameters:

- *<parent form reference>*-An object reference pointing to the parent form.
- *<object name expC>*-A character string assigned to the Name property of the new popup menu. This value is optional.

For example, the following commands create a form and a popup to display in it:

```
MyForm = NEW FORM()
MyForm.MyPop = NEW POPUP(MyForm, "OurPopup")
MyForm.MyPop.Item1 = NEW MENU(MyForm.MyPop, "Close")
MyForm.MyPop.Item2 = NEW MENU(MyForm.MyPop, "Close and Save")
MyForm.Open()
MyForm.MyPop.Open()
```

The Name property of the new popup object contains "OurPopup".

Assigning actions to popup menu items You assign an action to a popup menu item with the OnClick subroutine. For example, the following command assigns a subroutine named ClsSave to the Close and Save menu item of the example above:

```
MyForm.MyPop.Item2.OnClick = ClsSave
```

Note You can design a popup menu with the Popup Designer, a tool that a popup menu file (.POP). A popup menu file contains dBASE code that generates the popup menu you design. To access the Popup Designer, click the File menu and select New | Popup.

Example

```
f = NEW Form()
DEFINE POPUP p OF f
DEFINE MENU Inspect OF f.p;
  PROPERTY;
    Text "Inspector"
```

See Also

CLASS MENU, CREATE POPUP

CLASS SHAPE

A region of color within a form.

Properties

The following table lists the properties of the Shape class. For more information on each property, see the Properties section that follows.

Property	Default	Description
ClassName	SHAPE	Identifies the shape class
ColorNormal	BtnText/BtnFace	Sets the border and interior colors of the shape object
Height	N/A	Sets the height
Left	N/A	Sets the position of the left border
Move( )	N/A	Moves or sizes the shape object
Name	SHAPE1	Specifies the name of the shape object
OnOpen	N/A	Executes a subroutine when the parent form is opened
PageNo	N/A	Specifies on which page of a multi-page form the shape object appears
Parent	N/A	An object reference that points to the parent form
PenStyle	0	Specifies one of a series of line styles to be used for the border of the shape object
PenWidth	1	Specifies the width of the border line of a shape object
Release( )	N/A	Releases the shape object definition from memory
ShapeStyle	3 (Circle)	Specifies which of several styles are applied to a shape object
Top	N/A	Sets the position of the top border
Visible	.T.	Determines whether the shape object is visible or hidden
Width	N/A	Sets the width

Description

Use a Shape object to create a region of color within a form. The ShapeStyle property determines the shape of the region you create. Like a Line object, a Shape does not have an ID or an hWnd property.

While a Shape does not have a Border property, you can simulate a border by designating a pair of colors (<foreground color>/<background color>) for the ColorNormal property. The Shape object will display with a single-line border which is <foreground color> while the interior of the Shape object is <background color>. Setting ColorNormal to a single color value makes the entire shape that color.

Example

The following example creates a form and places an elliptical blue object with a bright white border inside the form.

```
MyForm = NEW FORM("Shape Display")
MyShape = NEW SHAPE(MyForm, "OURSHAPE")   &&Name property = "OURSHAPE"
MyShape.ShapeStyle = 2&& Elliptical shape
```

```
MyShape.ColorNormal = "W+/B"&& Bright white border, blue interior
MyForm.Open()
```

The Name property of the new Shape object contains "OURSHAPE".

See Also
CLASS IMAGE, CLASS RECTANGLE

CLASS TABBOX

A class that makes available the type of tab-based control that is used in the SET dialog, the Property Inspector and other places in dBASE.

Properties
The following table lists the properties of the TabBox class. For more information on each property, see the Properties section of this chapter.

Property	Default	Description
Anchor	1	Specifies whether the tab box stays in the same relative position when the form is resized
Before	N/A	Specifies which object the tab box precedes in the tabbing order of the parent form
ClassName	TABBOX	Identifies the tab box class
ColorHighlight	BtnText/BtnFace	Sets the color of the tab box when it is selected
ColorNormal	BtnText/BtnFace	Sets the color of the tab box when it isn't selected
CurSel	1	Specifies the currently-selected prompt in the tab box
DataSource	ARRAY {"TABBOX1"}	Specifies the prompts to display in the tab box
Enabled	.T.	Determines if the tab box can be selected
FontBold	.T.	Determines if characters in the tab box prompt are displayed in bold type
FontItalic	.F.	Determines if characters are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to displayed characters
FontSize	N/A	Specifies the size of the font in points
FontStrikeOut	.F.	Determines if characters are displayed in strikeout type
FontUnderline	.F.	Determines if characters are displayed in underlined type
Height	1	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
hWnd	N/A	Specifies the tab box handle
ID	100	Identifies the tab box with a numeric value
Left	0	Sets the position of the left border
Move( )	N/A	Moves or sizes the tab box
Name	TABBOX1	Specifies the tab box name

Property	Default	Description
OnGotFocus	N/A	Executes a subroutine when the tab box receives focus
OnHelp	N/A	Executes a subroutine when the user presses *F1*
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the tab box
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the tab box with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the tab box
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDblClick	N/A	Executes a subroutine when the user double-clicks the tab box with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the tab box with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the tab box
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse pointer over the tab box
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDblClick	N/A	Executes a subroutine when the user double-clicks the tab box with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the tab box with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the tab box
OnSelChange	N/A	Executes a subroutine when the highlight is moved from one prompt to another
PageNo	0	Specifies on which page of a multi-page form the tab box object appears; a value of 0 means it appears on all pages
Parent	N/A	An object reference that points to the parent form
Release( )	N/A	Removes the tab box definition from memory
SetFocus( )	N/A	Gives focus to the tab box
TabStop	.T.	Determines if the user can give focus to the tab box by pressing *Tab* or *Shift+Tab*
Top	N/A	Sets the position of the top border
Visible	.T.	Determines whether the tab box is visible or hidden
When	N/A	Specifies a condition that must evaluate to true before the user can give focus to the tab box
Width	N/A	Sets the width

Description

A TabBox contains a number of tabs that users can select. The TabBox control behaves like a list box. As with the list box, you can use the CurSel and OnSelChange( ) properties to specify actions to perform.

By setting the PageNo property of a TabBox control to 0 (the default), you can implement a tabbed multi-page form where the user can easily switch pages by selecting tabs. Use the PageNo property of a control to determine on which page the

control appears, and use the CurSel and OnSelChange( ) properties of the TabBox to switch between pages.

You can use the DataSource property with a literal array to specify the text prompts that appear on the tabs. For example, to create three tabs that say January, February, and March, use the following statement as the DataSource property:

```
Array {"January","February","March"}
```

Note There is a space between the word Array and the opening brace ({).

Tabs within a TabBox control are always situated horizontally.

Example

```
f = NEW Form()
DEFINE TABBOX t OF f;
  PROPERTY;
    DataSource 'ARRAY {"Page 1", "Page 2", "Page 3"}'
f.Open()
```

See Also
CLASS ARRAY, CLASS LISTBOX

Properties

AbandonRecord( ) — Method

Releases a newly-created record from memory.

Property of class
FORM

Description
Use AbandonRecord( ) to cancel the creation of a new record stored in a temporary memory buffer you created with BeginAppend( ).
For more information, see BeginAppend( ).

Example
See BeginAppend( ) for an example.

See Also
BeginAppend( ), IsRecordChanged( ), SaveRecord( )

Anchor — Property

Specifies whether an object stays in the same relative position when the form is resized.

Property of class
TABBOX

Data type
Numeric

Default
The default for Anchor is 1 (Bottom).

Description
Use Anchor to specify whether a tab box should maintain its size and location even if the parent form is resized. Acceptable values for Anchor are 1 (Bottom) and 0 (None). Generally, you'll want tabs on a form to automatically resize and reposition themselves as their parent form is resized, so you'll want to set Anchor to 1. For example, if you are using a tab box to move between different pages in a form, one tab is equivalent to one page, so the size of the tab and the size of the page (or form) should be the same.

However, if you want the tabs to retain a particular placement and size configuration despite the size of the parent form, set Anchor to 0.

Example

```
f = NEW Form()
DEFINE TABBOX TABBOX1 OF f PROPERTY Anchor 1
```

See Also
PageCount( ), PageNo

BeginAppend( ) — Method

Creates a temporary buffer in memory for a record that is based on the structure of the current table, letting the user input data to the record without automatically adding the record to the table.

Property of class
FORM

Description
BeginAppend( ) creates a single record buffer in the current table, without actually adding the record to the table until SaveRecord( ) is issued. While this buffer exists, the user can input data to the record with controls such as an entry field or a check box. Use SaveRecord( ) to append the record to the currently active table, and use

AbandonRecord( ) to discard the record. Use IsRecordChanged( ) to determine if the record has been changed since the BeginAppend( ) was issued.

For example, a form might contain two pushbuttons, one labeled Save and the other labeled Abandon. The OnClick subroutine of the Save pushbutton might execute SaveRecord( ) and the OnClick subroutine of the Abandon pushbutton might execute AbandonRecord( ).

You might also attach a procedure to the Abandon pushbutton to check the status of IsRecordChanged( ). If it is true, you could ask the user to confirm that they want to cancel the append operation.

Using BeginAppend( ) has different results than using either BEGINTRANS( ) and APPEND BLANK or APPEND AUTOMEM. With these commands, if you cancel the append operation, you have a record marked for deletion added to the table. If you use AbandonRecord( ) to cancel the BeginAppend( ) operation, a new record is never added to the table.

Example

```
local f
f = new ANIFORM()
f.Open()
CLASS ANIFORM OF FORM
   this.Top = 3.5879
   this.PageNo = 1
   this.Width = 57.666
   this.ColorNormal = "N/BTNFACE"
   this.View = "animals.dbf"
   this.Text = "Form"
   this.TopMost = .F.
   this.ScrollBar = 2
   this.Height = 10.9404
   this.Left = 24
   this.OnOpen = CLASS::FORM_ONOPEN
   DEFINE TEXT TITLE OF THIS;
       PROPERTY;
         Top 0.5,;
         PageNo 1,;
         Width 28,;
         FontSize 18,;
         ColorNormal "HIGHLIGHT/BTNFACE",;
         Text "Animals",;
         Border .F.,;
         Height 2.0293,;
         Left 1
   DEFINE TEXT TEXT1 OF THIS;
       PROPERTY;
         Top 3,;
         PageNo 1,;
         Width 14,;
         ColorNormal "BTNTEXT/BTNFACE",;
         Text "&Name",;
         Border .T.,;
         OldStyle .T.,;
```

```
      Height 2,;
      Left 1
DEFINE ENTRYFIELD ENTRYFIELD1 OF THIS;
    PROPERTY;
      ColorHighLight "WindowText/Window",;
      Top 4,;
      PageNo 1,;
      Width 13.3672,;
      ColorNormal "WINDOWTEXT/WINDOW",;
      Border .F.,;
      Height 0.8232,;
      Left 1.2988,;
      DataLink "ANIMALS->NAME"
DEFINE TEXT TEXT2 OF THIS;
    PROPERTY;
      Top 3,;
      PageNo 1,;
      Width 14,;
      ColorNormal "BTNTEXT/BTNFACE",;
      Text "&Size",;
      Border .T.,;
      OldStyle .T.,;
      Height 2,;
      Left 15
DEFINE SPINBOX SPINBOX1 OF THIS;
    PROPERTY;
      ColorHighLight "WindowText/Window",;
      Top 4,;
      PageNo 1,;
      Width 13.3672,;
      ColorNormal "WINDOWTEXT/WINDOW",;
      Border .F.,;
      Rangemax 100,;
      Rangemin 1,;
      Height 0.8232,;
      Left 15.2988,;
      DataLink "ANIMALS->SIZE"
DEFINE TEXT TEXT3 OF THIS;
    PROPERTY;
      Top 3,;
      PageNo 1,;
      Width 28,;
      ColorNormal "BTNTEXT/BTNFACE",;
      Text "We&ight",;
      Border .T.,;
      OldStyle .T.,;
      Height 2,;
      Left 29
DEFINE SPINBOX SPINBOX2 OF THIS;
    PROPERTY;
      ColorHighLight "WindowText/Window",;
      Top 4,;
      PageNo 1,;
      Width 13.3672,;
      ColorNormal "WINDOWTEXT/WINDOW",;
```

```
      Border .F.,;
      Rangemax 100,;
      Rangemin 1,;
      Height 0.8232,;
      Left 29.2988,;
      DataLink "ANIMALS->WEIGHT"
DEFINE TEXT TEXT4 OF THIS;
    PROPERTY;
      Top 5,;
      PageNo 1,;
      Width 56,;
      ColorNormal "BTNTEXT/BTNFACE",;
      Text "&Area",;
      Border .T.,;
      OldStyle .T.,;
      Height 2,;
      Left 1
DEFINE ENTRYFIELD ENTRYFIELD2 OF THIS;
    PROPERTY;
      ColorHighLight "WindowText/Window",;
      Top 6,;
      PageNo 1,;
      Width 20.0342,;
      ColorNormal "WINDOWTEXT/WINDOW",;
      Border .F.,;
      Height 0.8232,;
      Left 1.2988,;
      DataLink "ANIMALS->AREA"
DEFINE PUSHBUTTON PUSHBUTTON1 OF THIS;
    PROPERTY;
      Top 8,;
      PageNo 1,;
      Width 14.167,;
      ColorNormal "BtnText/BtnFace",;
      Text "&Save",;
      Default .T.,;
      OnClick CLASS::PUSHBUTTON1_ONCLICK,;
      UpBitmap "RESOURCE #20",;
      DownBitmap "RESOURCE #20",;
      DisabledBitmap "RESOURCE #21",;
      FocusBitmap "RESOURCE #20",;
      Group .T.,;
      Height 1.8818,;
      Left 9.166
DEFINE PUSHBUTTON PUSHBUTTON2 OF THIS;
    PROPERTY;
      Top 8,;
      PageNo 1,;
      Width 14.167,;
      ColorNormal "BtnText/BtnFace",;
      Text "&Abandon",;
      OnClick CLASS::PUSHBUTTON2_ONCLICK,;
      UpBitmap "RESOURCE #24",;
      DownBitmap "RESOURCE #24",;
      DisabledBitmap "RESOURCE #25",;
```

```
            FocusBitmap "RESOURCE #24",;
            Group .T.,;
            Height 1.8818,;
            Left 31.166
        Procedure PUSHBUTTON1_OnClick
            IF Form.IsRecordChanged()
                Form.SaveRecord()
            ENDIF
            GOTO BOTTOM
            SKIP
            Form.BeginAppend()
        RETURN

        Procedure PUSHBUTTON2_OnClick
            Form.AbandonRecord()
            Form.Close()
        RETURN

        Procedure Form_OnOpen
            GOTO BOTTOM
            SKIP
            This.BeginAppend()
        RETURN

    ENDCLASS
```

See Also

AbandonRecord( ), APPEND AUTOMEM, BEGINTRANS( ), IsRecordChanged( ), SaveRecord( )

CanClose — Event

Executes a subroutine that determines if a form can be closed when an attempt is made to close the form.

Property of class

FORM

Description

Use CanClose to prevent a form from closing until certain conditions are met. CanClose can also be used to perform tasks when the form is about to close, but the form and its objects are still in scope. This is different from OnClose which does not execute its subroutine until the form is closed and the form's object are out of scope. The subroutine you assign to CanClose returns a value of true (.T.) which allows the form to close, or false (.F.) which prevents the form from closing.

For example, when a form is based on a QBE that joins two tables on a key field, you might want to prevent the key field of the parent table from containing blank or duplicated values. The subroutine you assign to CanClose might search the parent table

for blank or duplicated values and return true (.T.) if no such values exist (allowing the form to close) or return false (.F.) if such values do exist (preventing the form from closing).

Example

```
f = new SAMPLEFORM()
f.Open()
CLASS SAMPLEFORM OF FORM
  * Form can only be closed if entryfield is not blank
   this.CanClose = CLASS::FORM_CANCLOSE
   DEFINE ENTRYFIELD ENTRYFIELD1 OF THIS;
       PROPERTY;
         Value "          ",;
         Top 2,;
         Width 10,;
         Height 1,;
         Left 4
  * Returns .T. only if TestField is not blank
   Procedure Form_CanClose
   RETURN IIF(ISBLANK(TRIM(THIS.ENTRYFIELD1.VALUE)), .F., .T.)
ENDCLASS
```

See Also
Close( ), OnClose

Copy( ) — Method

Copies selected text to the Windows clipboard.

Property of class
BROWSE, COMBOBOX, EDITOR, ENTRYFIELD, SPINBOX

Description
Use Copy( ) when the user has selected text and wants to copy it to the Windows clipboard. The action of Copy( ) is identical to the Copy menu item on the standard Windows Edit menu.

If you have assigned a menubar to the form, you can use the menubar's EditCopyMenu property instead of using the Copy( ) property of individual objects on the form. For more information, see EditCopyMenu.

Example

```
local f
f = new SAMPLEFORM()
f.Open()
CLASS SAMPLEFORM OF FORM
   this.Width = 51.833
   this.Text = "Sample Form"
```

```
this.Height = 12.6465
this.OnOpen = CLASS::FORM_ONOPEN
DEFINE ENTRYFIELD ENTRYFIELD1 OF THIS;
    PROPERTY;
      OnGotFocus CLASS::ENTRYFIELD1_ONGOTFOCUS,;
      Value "Sample Text",;
      Border .T.,;
      Top 3,;
      PageNo 1,;
      Width 25,;
      ColorNormal "WINDOWTEXT/WINDOW",;
      Height 1.5,;
      ColorHighLight "WINDOWTEXT/WINDOW",;
      Left 3
DEFINE PUSHBUTTON PUSHBUTTON1 OF THIS;
    PROPERTY;
      Group .T.,;
      Top 1,;
      PageNo 1,;
      Width 14,;
      ColorNormal "BtnText/BtnFace",;
      Text "&UnDo",;
      OnClick CLASS::PUSHBUTTON1_ONCLICK,;
      Height 1.5,;
      Enabled .F.,;
      Left 33.5
DEFINE PUSHBUTTON PUSHBUTTON2 OF THIS;
    PROPERTY;
      Group .T.,;
      Top 4,;
      PageNo 1,;
      Width 14,;
      ColorNormal "BtnText/BtnFace",;
      Text "Cu&t",;
      OnClick CLASS::PUSHBUTTON2_ONCLICK,;
      Height 1.5,;
      Left 33.5
DEFINE PUSHBUTTON PUSHBUTTON3 OF THIS;
    PROPERTY;
      Group .T.,;
      Top 7,;
      PageNo 1,;
      Width 14,;
      ColorNormal "BtnText/BtnFace",;
      Text "&Copy",;
      OnClick CLASS::PUSHBUTTON3_ONCLICK,;
      Height 1.5,;
      Left 33.5
DEFINE PUSHBUTTON PUSHBUTTON4 OF THIS;
    PROPERTY;
      Group .T.,;
      Top 10,;
      PageNo 1,;
      Width 14,;
      ColorNormal "BtnText/BtnFace",;
```

```
      Text "&Paste",;
      OnClick CLASS::PUSHBUTTON4_ONCLICK,;
      Height 1.5,;
      Enabled .F.,;
      Left 33.5
DEFINE ENTRYFIELD ENTRYFIELD2 OF THIS;
    PROPERTY;
      OnGotFocus CLASS::ENTRYFIELD2_ONGOTFOCUS,;
      Value "",;
      Border .T.,;
      Top 6,;
      PageNo 1,;
      Width 25,;
      ColorNormal "WINDOWTEXT/WINDOW",;
      Height 1.5,;
      ColorHighLight "WINDOWTEXT/WINDOW",;
      Left 3
DEFINE TEXT TEXT1 OF THIS;
    PROPERTY;
      Border .F.,;
      Top 2,;
      PageNo 1,;
      Width 20.166,;
      ColorNormal "BTNTEXT/BTNFACE",;
      Text "C&opy/Cut from here:",;
      Height 0.7646,;
      Left 3.5
DEFINE TEXT TEXT2 OF THIS;
    PROPERTY;
      Border .F.,;
      Top 5,;
      PageNo 1,;
      Width 18.166,;
      ColorNormal "BTNTEXT/BTNFACE",;
      Text "P&aste to here:",;
      Height 0.7646,;
      Left 3.5
Procedure PUSHBUTTON1_OnClick
    Form.EntryLast.Undo()
    This.Enabled = .F.
Return
Procedure PUSHBUTTON2_OnClick
    IF .NOT. ISBLANK(TRIM(Form.EntryLast.Value))
        Form.EntryLast.Cut()
        Form.Pushbutton1.Enabled = .T.
    ENDIF
RETURN
Procedure PUSHBUTTON3_OnClick
    IF .NOT. ISBLANK(TRIM(Form.EntryLast.Value))
        Form.EntryLast.Copy()
    ENDIF
RETURN
Procedure PUSHBUTTON4_OnClick
    Form.EntryLast.Paste()
    Form.Pushbutton1.Enabled = .T.
```

```
    RETURN
    Procedure ENTRYFIELD1_OnGotFocus
        Form.EntryLast = This
        Form.Pushbutton1.Enabled = .F.
        Form.Pushbutton2.Enabled = .T.
        Form.Pushbutton3.Enabled = .T.
        Form.Pushbutton4.Enabled = .F.
    Return
    Procedure ENTRYFIELD2_OnGotFocus
        Form.EntryLast = This
        Form.Pushbutton1.Enabled = .F.
        Form.Pushbutton2.Enabled = .F.
        Form.Pushbutton3.Enabled = .F.
        Form.Pushbutton4.Enabled = .T.
    RETURN
    Procedure Form_OnOpen
        This.EntryField1.SetFocus()
    Return
ENDCLASS
```

See Also
Cut( ), EditCopyMenu, Paste( ), Undo( )

Count( ) Method

Returns the number of prompts in a list box or the number of elements in an associated array.

Property of class
ASSOCARRAY, LISTBOX

Description
Use Count( ) when you can't anticipate the number of prompts a list box might have at run time. For example, when you specify FILE *.* for the DataSource property, the number of prompts varies when files are added to or deleted from the default directory.

You can use Count( ) to control loops that evaluate user choices in a multiple-choice list box. For example, you can see which prompts were chosen by evaluating each prompt with the Selected( ) method in a DO...WHILE loop.

Make a list box multiple-choice by setting the Multiple property to true (.T.).

You can also use Count( ) to determine the number of elements in an associated array.

Example
The following example defines a form that contains a listbox that displays names from the Animals.DBF table. Property Multiple .T. lets the user select more than one listbox prompt. The OnRightMouseDown property calls procedure Checked, which uses the methods Count( ) and Selected( ) to send the selected prompts to the Command window results pane with each OnRightMouseDown:

```
LOCAL f
f = NEW XFORM()
f.Open()

CLASS XFORM OF FORM
   this.Left =         52.80
   this.Width =         40.60
   this.Text = "Animals of the World"
   this.HelpId = ""
   this.OnRightMouseDown = CHECKED
   this.HelpFile = ""
   this.Top =          3.12
   this.Height =         20.00

   DEFINE LISTBOX LB1 OF THIS;
       PROPERTY;
         Left         10.00,;
         ColorNormal "N/W*",;
         Width         20.00,;
         DataSource "FIELD ANIMALS->NAME",;
         Multiple .T.,;
         ColorHighLight "W+/B",;
         Top          4.00,;
         Height         12.00
ENDCLASS

PROCEDURE Checked
FOR i=1 TO Form.LB1.Count()
  ? Form.LB1.Selected(i)
NEXT i
RETURN
```

See Also
FOR...NEXT, LISTCOUNT( ), LISTSELECTED( ), Selected( ), Multiple, IsKey, NextKey

CUATab — Property

Determines cursor behavior when you press Tab while on a Browse or Editor object.

Property of class
BROWSE, EDITOR

Data type Logical

Default The default for CUATab is true (.T.).

Description
When CUATab is .T. (the default value), pressing Tab moves to the next control in the Form's tab order. When CUATab is .F., pressing Tab moves to the next field in a Browse object or moves the cursor to the next tab stop position in an Editor object.

Example

```
local f
f = new SAMPLEFORM()
f.Open()
CLASS SAMPLEFORM OF FORM
   this.Top = 2
   this.Width = 76
   this.Text = "Sample Form"
   this.View = "animals.dbf"
   this.Height = 20
   this.Left = 11
   DEFINE BROWSE BROWSE1 OF THIS;
       PROPERTY;
         Top 3,;
         Width 56,;
         Alias "ANIMALS",;
         Height 16,;
         CUATab .F.,;
         Left 1
   DEFINE OKBUTTON OKBUTTON1 OF THIS;
       PROPERTY;
         Group .T.,;
         Top 3,;
         Width 14,;
         OnClick CLASS::OKBUTTON1_ONCLICK,;
         Height 1.5,;
         Left 60
   Procedure OKBUTTON1_OnClick
       Form.Close()
   Return
ENDCLASS
```

See Also
_curobj, Before, NextObj, SET CUAENTER

Cut( ) — Method

Cuts selected text and places it on the Windows clipboard.

Property of class
BROWSE, COMBOBOX, EDITOR, ENTRYFIELD, SPINBOX

Description
Use Cut( ) when the user has selected text and wants to remove it from the edit control and place it on the Windows clipboard. The action of Cut( ) is identical to the Cut menu item on the standard Windows Edit menu.

If you have assigned a menubar to the form, you can use the menubar's EditCutMenu property instead of using the Cut( ) property of individual objects on the form. For more information, see EditCutMenu.

Example
See Copy( ) for an example.

See Also
Copy( ), EditCutMenu, Paste( ), Undo( )

DataSource Property

Determines which data is displayed in a list box, a combo box, or an image object.

Property of class
COMBOBOX, IMAGE, LISTBOX, TABBOX

Data type
Character

Default
The default for DataSource is an empty string.

Description
You need to specify a valid value for the DataSource property to display an image in an image object, or prompts in a list box or a combo box.

The DataSource property is similar to the DataLink property. However, data displayed through the DataLink property can be changed, while data displayed through the DataSource property is always read-only.

You can specify one of five DataSource values for a list box or a combo box:

1. FILE <*filename skeleton expC*> creates prompts from file names in the current default directory.
2. FIELD <*field name*> creates prompts from all the values in a field in a table file. Each prompt represents a record, and you can move from record to record by selecting different prompts. To create prompts that don't move from record to record, copy the field into an array object with COPY TO ARRAY, then use the DataSource ARRAY <*array name*> option (described in this list).
3. STRUCTURE creates prompts from all the field names in a table.
4. ARRAY <*array name*> creates prompts from elements in an array object.
5. TABLES creates prompts from the names of all tables in the currently open database. See OPEN DATABASE for information on databases.

To evaluate which prompts were chosen from a list box, use LISTSELECTED( ) or Selected( ).

Note You can specify one of three DataSource values for an image object:

- RESOURCE <*resource id*><*DLL name*> designates a resource within a DLL file. <*resource id*> is a numeric literal that identifies a bitmap image in the DLL file. <*DLL name*> is the name of the DLL file and must include the file name extension if the DLL file isn't already in memory.
- FILENAME <*filename*> is the name of a file containing a bitmap image.
- BINARY <*binary field*> is the name of a binary field containing bitmap images.

Example

NEW operator syntax:

```
this.COMBOBOX1 = NEW COMBOBOX(this)
this.COMBOBOX1.DataLink = "Company"
this.COMBOBOX1.Value = "General Consolidated"
    DataSource = "FIELD COMPANY"
* or
*   DataSource = "STRUCTURE"
* or
*   DataSource = "FILE '*.PRG'"
* or
*   DataSource = "TABLES"
* or
*   DataSource = "Array 'ComboList'"
```

DEFINE object syntax:

```
DEFINE COMBOBOX COMBOBOX1 OF THIS;
  PROPERTY;
    DataLink "Company",;
    Value "General Consolidated",;
    DataSource "FIELD COMPANY"
* or
*   DataSource "STRUCTURE"
* or
*   DataSource "FILE '*.PRG'"
* or
*   DataSource "TABLES"
* or
*   DataSource "Array 'ComboList'"
```

See Also

DEFINE

DesignView — Property

Designates a view (.QBE or table) that is used when designing a form.

Property of class

FORM

Data type
Character

Default
The default for DesignView is an empty string.

Description

Use DesignView to facilitate creating and datalinking a form when you don't want to assign a View property to the form. The value in DesignView is ignored at runtime.

There are two main instances in which you may want to use DesignView instead of View.

- If you know which tables will be open when the form is opened at runtime, use DesignView to avoid opening the tables again when the form is opened. For example, if you are designing a Search dialog box that will be opened only when a specific table is already open, set the DesignView property of the dialog box instead of the View property.
- If you don't know which tables will be open when the form is opened at runtime, but need certain tables open to design the form, use DesignView to avoid specifying at design time which tables will be open at runtime.

If you specify a View property for a form, you should not also specify a DesignView property. If you want to design multiple forms having different DesignView properties, you should design the forms in different sessions.

See Also
Alias, DataLink, DataSource, View

DirExt( ) Method

Stores the name, size, date stamp, time stamp, and DOS and Windows95 attributes of files to an array object.

Property of class
ARRAY

Description
DirExt( ) is identical to Dir( ), but with extra columns for Windows95 file information. For more information, see Dir( ).

Example

```
DirList = NEW ARRAY(1)
? DirList.DirExt()
```

See Also
Dir( )

DropDownHeight — Property

Specifies the number of lines displayed in the list portion of the combo box.

Property of class
COMBOBOX

Data type
Numeric

Default
The default for DropDownHeight is 6.

Description
Use DropDownHeight to specify how much information will appear when a user drops down a list from a combo box.

Example
In the following example, the dropdown portion of the list contains either 10 lines or the total number of list items available, whichever is smaller.

```
cbarray=NEW ARRAY(5)             && Create array containing 5 elements
cbarray2=NEW ARRAY(15)           && Create array containing 15 elements

f = NEW FORM()
c1 = NEW COMBOBOX(f)
c1.DataSource = 'ARRAY cbarray'  && Acceptable values come from array
c1.Style = 2                     && Only array items can be selected
c1.DropDownHeight = IIF(cbarray.size<10,cbarray.size,10)  && 5 lines
c2 = NEW COMBOBOX(f)
c2.Left = c1.Left+20
c2.DataSource = 'ARRAY cbarray2'
c2.Style = 2
c2.DropDownHeight = IIF(cbarray2.size<10,cbarray2.size,10)  && 10 lines
f.Open()
```

See Also
Style

EditCopyMenu Property

Specifies a menu item that copies selected text from a control to the Windows clipboard.

Property of Class
MENUBAR

Data type
Object reference

Description
EditCopyMenu contains a reference to a menu object users select when they want to copy text.

You can use the EditCopyMenu property of a form's menubar to copy selected text to the Windows clipboard from any edit control in the form, instead of using the control's Copy( ) property. In effect, EditCopyMenu calls Copy( ) for the active control. This lets you provide a way to copy text with less programming than would otherwise be needed. The Copy menu item is automatically disabled (greyed out) when no text is selected, and enabled when text is selected.

For example, suppose you have a Browse object (b) and an Editor object (e) on a form (f). To implement text copying, you could specify actions that would call b.Copy( ) or e.Copy( ) whenever a user wanted to copy text. However, if you use a menubar, you can set the EditCopyMenu property to the menu item the user will select to copy text. Then, when the user selects that menu item, the text is automatically copied to the Windows clipboard from the currently active control. You don't need to use the control's Copy( ) property at all.

If you use the Menu Designer to create a menubar, EditCopyMenu is automatically set to an item named Copy on a pulldown menu named Edit when you add the Edit menu to the menubar:

```
this.EditCopyMenu = this.Edit.Copy
```

Example
See WindowMenu for an example.

See Also
CLASS MENUBAR, Copy( ), EditCutMenu, EditPasteMenu, EditUndoMenu, WindowMenu

EditCutMenu Property

Specifies a menu item that cuts selected text from a control and places it on the Windows clipboard.

Property of Class
MENUBAR

Data type
Object reference

Description
EditCutMenu contains a reference to a menu object users select when they want to cut text.

You can use the EditCutMenu property of a form's menubar to cut (delete) selected text and place it on the Windows clipboard from any edit control in the form, instead of using the control's Cut( ) property. In effect, EditCutMenu calls Cut( ) for the active control. This lets you provide a way to copy text with less programming than would otherwise be needed. The Cut menu item is automatically disabled (greyed out) when no text is selected, and enabled when text is selected.

For more information, see EditCopyMenu.

Example
For an example of EditCutMenu, see WindowMenu.

See Also
CLASS MENUBAR, Cut( ), EditCopyMenu, EditPasteMenu, EditUndoMenu, WindowMenu

EditPasteMenu Property

Specifies a menu item that copies text from the Windows clipboard to the currently active edit control.

Property of Class
MENUBAR

Data type
Object reference

Description
EditPasteMenu contains a reference to a menu object users select when they want to paste text to the cursor position in the currently active edit control.

You can use the EditPasteMenu property of a form's menubar to paste text from the Windows clipboard into any edit control in the form, instead of using the control's Paste( ) property. In effect, EditPasteMenu calls Paste( ) for the active control. This lets you provide a way to paste text with less programming than would otherwise be needed. The Paste menu item is automatically disabled (greyed out) when the clipboard is empty, and enabled when text is copied or cut to the clipboard.

For more information, see EditCopyMenu.

Example
For an example of EditPasteMenu, see WindowMenu.

See Also
CLASS MENUBAR, Paste( ), EditCopyMenu, EditCutMenu, EditUndoMenu, WindowMenu

EditUndoMenu — Property

Specifies a menu item that reverses the effects of the last Cut, Copy, or Paste action.

Property of Class
MENUBAR

Data type
Object reference

Description
EditUndoMenu contains a reference to a menu object users select when they want to undo their last Cut, Copy, or Paste action.

You can use the EditUndoMenu property of a form's menubar to undo a Cut or Paste action from any edit control in the form, instead of using the control's Undo( ) property. In effect, EditUndoMenu calls Undo( ) for the active control. This lets you provide a way to undo with less programming than would otherwise be needed.

For more information, see EditCopyMenu.

Example
For an example of EditUndoMenu, see WindowMenu.

See Also
CLASS MENUBAR, Undo( ), EditCopyMenu, EditCutMenu, EditPasteMenu, WindowMenu

FirstKey Property

Returns the subscript character string for an element of an associated array.

Property of class
ASSOCARRAY

Description
Use FirstKey when you want to step through the elements in an associated array object, starting from the first element in the array. Once you have issued FirstKey and are positioned on the first element, use NextKey( ) to step through the elements in order.

Note Elements in associated array objects are not necessarily stored in the order in which you add them to an array. This means you can't assume that the value returned by FirstKey will be consistent, or that it will return the first item you added to the array.

For more information, see CLASS ASSOCARRAY.

Example
The following example creates an associated array and displays its subscripts and contents.

```
aa = NEW ASSOCARRAY()
aa["San Francisco"] = "49ers"
aa["Los Angeles"] = "Rams"
x = aa.FirstKey
DO WHILE .NOT. EMPTY(x)
   ? x, aa[x]              && display element subscript and contents
   x = aa.NextKey(x)     && 'increments' index pointer
ENDDO
```

See Also
IsKey( ), NextKey( )

Header3D Property

Specifies whether the top and left portions of a browse object appear raised (three-dimensional).

Property of class
BROWSE

Data type
Logical

Default
The default for Header3D is true (.T.).

Description
Header3D affects the appearance of the top and left sides of a browse window when ShowHeading and/or ShowRecNo (respectively) are true. If Header3D is true, the Heading and Record number appear three-dimensional, making it easier to see that they don't represent values contained in the table. If ShowHeading and ShowRecNo are false, Header3D has no effect.

Example
The following example shows a form with browse windows having three different display formats.

```
f=NEW FORM()
f.View = "CLIENTS.DBF"
f.Width=80
b = NEW BROWSE(f)
   b.Alias = "clients"
   b.ColorNormal = "WindowText/Window"
   b.colorHighLight = "WindowText/Window"
   b.Header3D=.T.          && 3D top and left

b3d = NEW BROWSE(f)
   b3d.Alias = "clients"
   b3d.Left = b.left+20
   b3d.ColorNormal = "WindowText/Window"
   b3d.colorHighLight = "WindowText/Window"
   b3d.Header3D=.F.         && Normal top and left

b3da = NEW BROWSE(f)
   b3da.Alias = "clients"
   b3da.Left = b3d.left+20
   b3da.ColorNormal = "WindowText/Window"
   b3da.colorHighLight = "WindowText/Window"
   b3da.Header3D=.T.        && Ignored because
   b3da.ShowRecNo=.F.        && ShowRecNo and
   b3da.ShowHeading=.F.      && ShowHeading are false
f.open()
```

See Also
ShowHeading, ShowRecNo

Icon — Property

Specifies an icon format file (.ICO) or resource that displays when a form is minimized.

Property of class
FORM

Data type
Character

Default
The default for Icon is an empty string.

Description
Use Icon to specify an image to be used when a form is minimized. You can specify a resource (generally from a .DLL or .VBX file) or a file name.

Example
In the following example, the form is minimized when it opens, and an icon from a specified DLL file is displayed:

```
f = NEW Form()
f.Icon = "RESOURCE #20 C:\MYAPP\MYAPP.DLL"
f.WindowState=1
```

See Also
Minimize, WindowState

IsKey( ) Method

Returns a logical value that indicates if the specified character expression is a subscript of an element in an associated array.

Property of class
ASSOCARRAY

Description
Use IsKey(<expC>) to determine if an associated array contains an element with a subscript of <expC>. This might be useful if you want to add an item to the array only if it is not already there, or if you want to remove an element that has been added to the array.

Example
The following example creates an associated array based on an existing table. It then uses IsKey( ) to see if a specific element is in the array. If it isn't, it adds it to the array, and to the table that was used to populate the array.

```
aa = NEW ASSOCARRAY()
USE customer
SCAN
   cName = TRIM(name)
   aa[cName] = city                      && Create element for each record
ENDSCAN
NewName = "Just Testing"
NewCity = "Anywhere"
IF aa.IsKey(NewName) = .F.  && No matching element in array
   APPEND BLANK
   REPL name WITH NewName, ;
```

```
        city WITH NewCity            && Add record to table
    aa[NewName] = NewCity      && Add element to array
ENDIF
```

See Also
FirstKey, NextKey( ), RemoveKey( )

IsRecordChanged( ) Method

Returns a logical value that indicates whether the current record in the append buffer, created with BeginAppend( ), has been modified.

Property of class
Form

Data type
Logical

Description
Use IsRecordChanged( ) to determine whether a user has modified a new record created with BeginAppend( ). For example, if a user tries to close the new record without saving it, check for IsRecordChanged( ). If it's true, you can prompt the user to confirm they want to abandon their changes.

Example
See BeginAppend( ) for an example.

See Also
AbandonRecord( ), BeginAppend( ), SaveRecord( )

Keyboard( ) Method

Passes a character string to an edit control, simulating typed user input.

Property of class
BROWSE, COMBOBOX, EDITOR, ENTRYFIELD, SPINBOX

Description
Use Keyboard( ) when you want to pass text to an entry control as if the user had typed it in manually. For example, if you are writing a tutorial, you can ask the user for information, then use Keyboard( ) to demonstrate where the user would type the information in an editor object on a form.

Example

The following example illustrates using Keyboard( ) to simulate the typing of text in an entry field.

```
f = new KBDFORM()
f.Open()
CLASS KBDFORM OF FORM
   this.Text = "Form"
   this.Top = 0
   this.PageNo = 0
   this.Width = 80
   this.ColorNormal = "BtnText/BtnFace"
   this.OnOpen = CLASS::FORM_ONOPEN
   DEFINE ENTRYFIELD EF1 OF THIS;
       PROPERTY;
         Border .T.,;
         ColorHighLight "WINDOWTEXT/WINDOW",;
         Top 6,;
         PageNo 1,;
         Width 30,;
         ColorNormal "WINDOWTEXT/WINDOW",;
         Height 1,;
         Left 12
   Procedure Form_OnOpen
   this.ef1.setfocus()
   this.ef1.keyboard("Type your name here")

ENDCLASS
```

See Also

Paste( )

NextKey( ) — Method

Returns the subscript of the next element in an associated array.

Property of class

ASSOCARRAY

Description

Use NextKey( ) to step through the elements in an associated array object, starting from the first element in the array. Generally, you'll use FirstKey to position yourself on the first element in the array, and then use NextKey( ) to step through the elements in order. NextKey( ) requires one parameter: the index of the element of the array to start from when looking for the next element.

Example

See FirstKey for an example of NextKey( ).

See Also
FirstKey, IsKey( )

OnChar — Event

Executes a subroutine when a "printable" key or key combination is pressed while the control has focus.

Property of class
PAINTBOX

Data type
Function pointer or codeblock

Description
If you have created a paintbox object to develop a custom edit control, use OnChar to determine actions that should take place when the object has focus and the user presses a key or key combination that can be printed. (To specify actions triggered by other keys, see OnKeyDown.)

OnChar is similar to OnKeyDown. However, OnChar returns nothing for non-printable keys, such as Shift or CapsLock, while OnKeyDown returns a value for any key pressed.

Three numeric parameters are passed to the OnChar event:

- <*nChar*>: the scan code of the key or key combination
- <*nReptCnt*>*:* the number of times the keystroke is repeated based on how long the key is held down
- <*nFlags*>*:* a parameter used to specify if the Shift or Ctrl key was pressed

(For more information on nFlags, see any of the On Mouse events, such as OnLeftMouseDown.)

Example

```
local f
f = new SAMPLEFORM()
f.Open()
CLASS SAMPLEFORM OF FORM
   this.TopMost = .F.
   this.Height = 8.7051
   this.Text = "Sample"
   this.Left = 47.166
   this.Top = 3.2344
   this.PageNo = 1
   this.Width = 27
   this.ColorNormal = "N/BTNFACE"
   DEFINE RECTANGLE RECTANGLE1 OF THIS;
       PROPERTY;
```

```
            Height 5.5,;
            Text "",;
            Left 4.5,;
            Border .T.,;
            Top 1.5,;
            PageNo 1,;
            Width 19,;
            ColorNormal "BTNTEXT/BTNFACE"
      DEFINE PAINTBOX PAINTBOX1 OF THIS;
          PROPERTY;
            OnKeyDown CLASS::PAINTBOX1_ONKEYDOWN,;
            OnChar CLASS::PAINTBOX1_ONCHAR,;
            Height 5,;
            OnPaint CLASS::PAINTBOX1_ONPAINT,;
            Left 6,;
            Top 2,;
            PageNo 1,;
            OnFormSize CLASS::PAINTBOX1_ONFORMSIZE,;
            Width 17,;
            OnKeyUp CLASS::PAINTBOX1_ONKEYUP,;
            ColorNormal "BTNTEXT/BTNFACE"
   Procedure PAINTBOX1_OnChar(nChar, nRepCnt, nFlags)
         ? "OnChar values:"
         ? nChar
         ? nRepCnt
         ? nFlags
         ?
      RETURN

      Procedure PAINTBOX1_OnFormSize(sizeType, width, height)
          ? "OnFormSize values:"
          ? sizeType
          ? width
          ? height
          ?
      RETURN

      Procedure PAINTBOX1_OnKeyDown(nChar, nRepCnt, nFlags)
          ? "OnKeyDown values:"
          ? nChar
          ? nRepCnt
          ? nFlags
          ?
      RETURN

      Procedure PAINTBOX1_OnKeyUp(nChar, nRepCnt, nFlags)
          ? "OnKeyUp values:"
          ? nChar
          ? nRepCnt
          ? nFlags
          ?
      Return

      Procedure PAINTBOX1_OnPaint
          ? "PaintBox painted!"
```

```
        ?
    RETURN

ENDCLASS
```

See Also
Key, OnKeyDown, OnKeyUp, OnLeftMouseDown

OnFormSize — Event

Executes a subroutine whenever the parent form of a paintbox object is resized.

Property of class
PAINTBOX

Data type
Function pointer or codeblock

Description
OnFormSize is called whenever the parent form of a paintbox object is resized, restored, or maximized. This lets you reposition or resize the object based on the form's new size. For example, you could use OnFormSize to implement behavior similar to the Anchor property of the TABBOX class. keeping the bottom of the paintbox object positioned near the bottom of the form.

OnFormSize is similar to OnPaint. However, OnPaint is triggered when the parent form is opened and when items covering the paintbox object are moved away, while OnFormSize is not.

Example
See OnChar for an example.

OnInitMenu — Event

Specifies code that executes when a menubar or popup is opened.

Property of class
MENUBAR, POPUP

Data type
Function pointer or codeblock

Description
OnInitMenu is called whenever a menubar or popup is invoked, and is processed before the menubar's child menus or the popup is displayed.

You can use OnInitMenu to determine the status of menu items that will be displayed. For example, use OnInitMenu to determine if the Enabled or Checked property of a menu item should be true or false.

Example

```
Parameter FormObj
NEW SAMPLEMENU(FormObj,"Root")
CLASS SAMPLEMENU(FormObj,Name) OF MENUBAR(FormObj,Name)
   this.OnInitMenu = {; ? "Menu opened!"}
   DEFINE MENU FILE OF THIS;
       PROPERTY;
         Text "&File"
         DEFINE MENU EXIT OF THIS.FILE;
             PROPERTY;
               Text "E&xit",;
               OnClick {; Form.Close()}
ENDCLASS
```

See Also
Checked, Enabled

OnKeyDown — Event

Executes a subroutine when any key or key combination is pressed while the control has focus.

Property of class
PAINTBOX

Data type
Function pointer or codeblock

Description
If you have created a paintbox object to develop a custom edit control, use OnKeyDown to determine actions that should take place when the object has focus and the user presses any key or key combination. (To specify actions triggered by printable keys, see OnChar.)

OnChar is similar to OnKeyDown. However, OnChar returns nothing for non-printable keys, such as Shift or CapsLock, while OnKeyDown returns a value for any key pressed.

Three numeric parameters are passed to the OnKeyDown event:

- *nChar* - the scan code of the key or key combination
- *nReptCnt* - the number of times the keystroke is repeated based on how long the key is held down
- *nFlags* - a parameter used to specify if the Shift or Ctrl key was pressed

(For more information on nFlags, see any of the On Mouse events, such as OnLeftMouseDown.)

Example
See OnChar for an example.

See Also
Key, OnKeyDown, OnKeyUp, OnLeftMouseDown

OnKeyUp — Event

Executes a subroutine when any key or key combination is released while the control has focus.

Property of class
PAINTBOX

Data type
Function pointer or codeblock

Description
If you have created a paintbox object to develop a custom edit control, use OnKeyUp to determine actions that should take place when the object has focus and the user releases a key. (To specify actions triggered when the user presses a key, see OnKeyDown.)

Three numeric parameters are passed to the OnKeyup event:

- *nChar* - the scan code of the key or key combination
- *nReptCnt* - the number of times the keystroke is repeated based on how long the key is held down
- *nFlags* - a parameter used to specify if the Shift or Ctrl key was pressed

(For more information on nFlags, see any of the On Mouse events, such as OnLeftMouseUp.)

Example
See OnChar for an example

See Also
OnChar, OnKeyDown, OnLeftMouseUp

OnPaint — Event

Executes a subroutine whenever a paintbox object needs to be redrawn.

Property of class
PAINTBOX

Data type
Function pointer or codeblock

Description
OnPaint is called whenever a paintbox object needs to be redrawn. Events that trigger OnPaint include:

- the parent form is opened
- the parent form is resized
- a minimized parent form is restored or maximized
- a window or object which has been covering the paintbox object is moved away

Example
See OnChar for an example.

See Also
OnFormSize

PageCount( ) — Method

Returns the highest numbered page defined for a form.

Property of class
FORM

Description
Use PageCount( ) to determine how many pages a multi-page form contains. For example, if you have a "Next Page" button or menu choice, you can use PageCount( ) in conjunction with PageNo to determine if you are already on the last page of a form.

If you have a form that is set up as a result of user input or other program activities, you might use PageCount( ) in conjunction with CLASS TABBOX to define a series of tabs for the defined pages.

Example

```
f = NEW Form()
DEFINE PUSHBUTTON p OF f;
  PROPERTY;
    Height 2,;
    Left   2,;
    Top    2,;
    Text  "Push",;
    Width 10
f.Open()
? f.PageCount()
```

See Also
CLASS TABBOX

PageNo Property

PageNo Returns the active page of a form or the page on which a control appears.

Property of class
BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMGE, LINE, LISTBOX. OLE, PAINTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SHAPE, SPINBOX, TABBOX, TEXT

Data Type
Numeric

Default
There is no default for the PageNo property of a form. However, when you create a new form using the Form Designer, PageNo is set to 1.

The default for PageNo is 0 for a TabBox object and 1 for all other objects.

Description
The PageNo property of a form returns which page of the form is currently active. If you set a form's PageNo property to 0 (zero), all controls on all pages are displayed.

For all controls other than forms, the PageNo property specifies on which page of a multi-page form the control appears. A value of 0 (zero) means the control will appear on every page of the form.

You may want to implement multi-page forms whenever a single form contains a large number of objects. Dividing the objects logically among two or more pages helps organize the objects, and may make the form easier to use.

If you want to use tabs to let users switch pages, set the PageNo property of the TabBox to 0 (the default). This ensures that the tabs are visible while the user is on any page of the form. If you want any other control to appear on all pages (such as a Close button), set the PageNo property of the control to 0.

Example

The following example shows a form that contains two pages. The user switches between them by using buttons labeled Next Page and Previous Page. A Close button appears on every page.

```
LOCAL f
f = new MULTIPGFORM()
f.Open()
CLASS MULTIPGFORM OF FORM
   this.Top = 0
   this.Width = 60
   this.OnOpen = {;form.pageno=1}&& Make sure page 1 displays first
    DEFINE PUSHBUTTON PUSHBUTTON1 OF THIS;
       PROPERTY;
         Top 15,;
         PageNo 1,;
         Width 15,;
         Text "Next Page",;
         OnClick {;form.PageNo=2},;
         Left 42
   DEFINE PUSHBUTTON PUSHBUTTON2 OF THIS;
       PROPERTY;
         Top 15,;
         PageNo 2,;
         Width 15,;
         Text "Previous Page",;
         OnClick {;form.PageNo=1},;
         Left 42
   DEFINE PUSHBUTTON PUSHBUTTON3 OF THIS;
       PROPERTY;
         Top 17,;
         PageNo 0,;
         Width 10,;
         Text "Close",;
         OnClick {;form.Close()},;
         Left 42
   DEFINE TEXT TEXT1 OF THIS;
       PROPERTY;
         Top 4,;
         PageNo 1,;
         Width 34,;
         Text "This appears on page 1",;
         Height 2, ;
         Left 13
   DEFINE TEXT TEXT2 OF THIS;
       PROPERTY;
         Top 4,;
         PageNo 2,;
         Width 34,;
         Text "This appears on page 2",;
         Height 2, ;
         Left 13
ENDCLASS
```

See Also
CLASS TABBOX, PageCount( ), SpeedBar

Paste( ) Method

Copies text from the Windows clipboard to the currently active edit control.

Property of class
BROWSE, COMBOBOX, EDITOR, ENTRYFIELD, SPINBOX

Description
Use Paste( ) when the user wants to copy text from the Windows clipboard to the cursor position in the currently active edit control. The action of Paste( ) is identical to the Paste menu item on the standard Windows Edit menu.

If you have assigned a menubar to the form, you can use the menubar's EditPasteMenu property instead of using the Paste( ) property of individual objects on the form. For more information, see EditPasteMenu.

Example
See Copy( ) for an example.

See Also
Copy( ), Cut( ), EditPasteMenuUndo( )

PenStyle Property

Specifies the type of line to be used as the border of a shape object.

Property of class
SHAPE

Data type
Numeric

Default
The default for PenStyle is 0 (Solid).

Description
Use PenStyle to control the appearance of the border of a shape object.

You can specify any of five settings for PenStyle:

Number	Description	Example
0	Solid	———
1	Dash	– – – –
2	Dot	.....
3	Dash Dot	. – . – . –
4	DashDotDot	.. – .. – .. –

Example
NEW operator syntax:

```
Sh2=NEW SHAPE(this)
Sh2.Left=3
Sh2.Top=8
Sh2.PenStyle = 3                      && Dash Dot
DEFINE object syntax:
DEFINE SHAPE Sh2 OF THIS;
  PROPERTY Left 3, Top 8, PenStyle 3  &&Dash Dot
```

See Also
PenWidth, ShapeStyle

PenWidth — Property

Specifies the width in pixels of the line used as the border of a shape object.

Property of class
SHAPE

Data type
Numeric

Default
The default for PenWidth is 1.

Description
Use PenWidth to specify the thickness of the line used to border a shape object. If you set PenWidth to a value greater than 1, then PenSyle can only be set to 0.

Example
NEW operator syntax:

```
Sh2=NEW SHAPE(this)
Sh2.Left=3
Sh2.Top=8
Sh2.PenWidth = 3                        && 3 pixels
```

```
DEFINE object syntax:
DEFINE SHAPE Sh2 OF THIS;
  PROPERTY Left 3, Top 8, PenWidth 3     && 3 pixels
```

See Also
PenStyle, ShapeStyle

PopupMenu Property

Specifies a popup menu for a form.

Property of class
FORM

Data Type
Object reference

Description
Assign the PopupMenu property to an existing popup menu to have the popup appear when the user right clicks on the form.

Example
The following example creates a form and attaches an existing popup menu (Pop0328.pop) to the form's OnOpen property. It then opens the form and the Inspector, so you can inspect the form's PopupMenu property. If you right-click while the form is open, the popup menu is displayed.

```
f = new POPFORM()
f.Open()
INSPECT(f)
CLASS POPFORM OF FORM
   this.TopMost = .F.
   this.Height = 20
   this.Left = 53
   this.Top = 0
   this.PageNo = 1
   this.Width = 60
   this.OnOpen = CLASS::FORM_ONOPEN
   Procedure Form_OnOpen
      IF TYPE("This.PopupMenu") # "O"
         DO Pop0328.pop with this,"MyPopTest"
         form.PopupMenu = form.MyPopTest
      ENDIF
   Return
ENDCLASS
```

The code in Pop0328.pop was generated by the Menu Designer:

```
* Pop0328.pop
Parameter FormObj,PopupName
NEW POP0328MENU(FormObj,PopupName)
CLASS POP0328MENU(FormObj,PopupName) OF POPUP(FormObj,PopupName)
   this.Top = 0
   this.Left = 0
   this.TrackRight = .T.
   DEFINE MENU CLOSE OF THIS;
       PROPERTY;
         Text "Close",;
         OnClick {;form.close()}
ENDCLASS
```

See Also
OnOpen, TrackRight

Refresh( ) Method

Updates data displayed in control objects within a form.

Property of class
FORM

Description
Use Refresh( ) to update data displayed in a form to reflect the current state of the data as it exists on disk. For example, you can use Refresh( ) in a multi-user environment to ensure that the displayed data reflects all recent changes made by other users.

Example
The following example lets you use a scrollbar or an entry field to change the data in the StartBal field of the Clients table. Because Refresh( ) is assigned to the scrollbar's OnChange property, the value in the entry field and the table always reflect the value as chosen with the scrollbar.

```
LOCAL f
f = NEW SBAR2FORM()
f.Open()
CLASS SBAR2FORM OF FORM
   this.Top = 0
   this.PageNo = 1
   this.Width = 49
   this.View = "CLIENTS.DBF"
   this.Height = 20
   this.Left = 50
   DEFINE BROWSE BROWSE1 OF THIS;
       PROPERTY;
         Top 2,;
         PageNo 1,;
         Width 25.835,;
         CUATab .T.,;
```

```
        Alias "CLIENTS",;
        ScrollBar 2,;
        Fields "CLIENT_ID,STARTBAL",;
        Height 11,;
        Left 6,;
        ShowRecNo .F.
    DEFINE SCROLLBAR SCROLLBAR1 OF THIS;
        PROPERTY;
        Top 16,;
        PageNo 1,;
        Width 20,;
        ColorNormal "ScrollBar",;
        OnChange {;form.refresh()},;
        DataLink "CLIENTS->STARTBAL",;
        Height 1,;
        Vertical .F.,;
        Left 9, ;
        Rangemax 32766

    DEFINE ENTRYFIELD ENTRYFIELD1 OF THIS;
        PROPERTY;
        Top 15,;
        PageNo 1,;
        Width 11,;
        ColorNormal "WINDOWTEXT/WINDOW",;
        Border .T.,;
        DataLink "CLIENTS->STARTBAL",;
        Height 1,;
        Left 14.5
    Procedure SCROLLBAR1_OnChange

    form.refresh()
ENDCLASS
```

See Also
REFRESH, SHOW OBJECT

RemoveAll( ) Method

Deletes all elements from an associated array object.

Property of class
ASSOCARRAY

Description
Use the RemoveAll( ) method to remove all elements from an associated array. You might want to do this if you want to repopulate the array with new values.

Example
The following example removes all elements from an associated array.

```
aa = NEW ASSOCARRAY()
aa["USA"] = "Washington, DC"
aa["Spain"] = "Madrid"       && Array contains two elements
aa.RemoveAll()               && Array now contains no elements
```

See Also
IsKey( ), RemoveKey( )

RemoveKey( ) Method

Deletes an element from an associated array object.

Property of class
ASSOCARRAY

Description
Use the RemoveKey( ) method to remove elements from an associated array object. RemoveKey( ) accepts a character string as its parameter. This string represents the subscript of the associated array element you want to remove.

Example
The following example removes an element from an associated array.

```
aa = NEW ASSOCARRAY()
aa["USA"] = "Washington, DC"
aa["Spain"] = "Madrid"     && Array contains two elements
aa.RemoveKey("USA")        && Array now contains one element
```

See Also
Delete( ), IsKey( ), RemoveAll( )

See Also
DEFINE, REPLACE BINARY

SaveRecord( ) Method

Saves a temporary record by appending it to the currently active table.

Property of class
FORM

Description
Use SaveRecord( ) to add to the currently active table a new record stored in a temporary memory buffer you created with BeginAppend( ).

For more information, see BeginAppend( ).

Example
See BeginAppend( ) for an example.

See Also
AbandonRecord( ), BeginAppend( ), IsRecordChanged( )

ShapeStyle Property

Determines the shape of a shape object.

Property of class
SHAPE

Data type
Numeric

Default
The default for ShapeStyle is 3 (Circle).

Description
Use ShapeStyle to specify a shape for a shape object.

The shapes you can specify are as follows:

Value	Shape
0	Rectangle with rounded corners
1	Rectangle
2	Ellipse
3	Circle
4	Square with rounded corners
5	Square

For example, if you want to provide a square, colored background for an area on a form, you can create a shape object with a ShapeStyle value of 5, which specifies a square shape. Use the ColorNormal property to specify the color of the shape object.

Example
The following example creates a form and places an elliptical blue object with a bright white border inside the form.

```
MyForms = NEW FORM("Shape Display")
MyShape = NEW SHAPE(MyForm, "OURSHAPE"    &&Name property = "OURSHAPE"
MyShape.ShapeStyle = 2                    && Elliptical shape
MyShape.ColorNormal = "W+/B"              && Bright white border, blue interior
MyForm.Open()
```

The Name property of the new Shape object contains "OURSHAPE".

See Also
ColorNormal, PenStyle, PenWidth

ShowSpeedTip — Property

Determines if tips about a control on a form appear in a balloon near the control when the mouse rests on those controls. The controls must have tips text defined via the SpeedTip property for tips to appear.

Property of class
FORM

Data type
Logical

Default
The default for ShowSpeedTip is true (.T.).

Description
Use ShowSpeedTip to determine whether tip messages appear for a control on a form. If ShowSpeedTip is .T., and controls have tips defined via the SpeedTip property, the tip will appear when the mouse comes to rest on the control. If ShowSpeedTip is .F., the tips will not appear. ShowSpeedTip has no effect on controls where no tip has been defined.

Example
In the following example, the tips won't be displayed because ShowSpeedTip has been set to false.

```
f = NEW Form()
f.ShowSpeedTip = .F.
DEFINE Pushbutton PSpeedTip OF f;
Property;
    SpeedTip    "Push to close",;
    OnClick {; Form.Close()},;
    Text    "&Close"  f.Open()
```

See Also
SpeedTip StatusMessage

SpeedTip

Property

Specifies the text that appears when the mouse remains on a control for more than one second.

Property of class
CHECKBOX, ENTRYFIELD, PUSHBUTTON, RADIOBUTTON, SPINBOX

Data type
Character

Default
The default for SpeedTip is an empty string.

Description
Use SpeedTip to create a brief text message which appears in a balloon when the mouse rests on a control. Usually this message gives the user a clue as to the function of the control. To suppress the display of Speed Tips, set the ShowSpeedTip property of the form to false (.F.).

Example

```
f = NEW Form()
DEFINE PUSHBUTTON PSpeedTip OF f;
  Property;
    SpeedTip "Push to close",;
    OnClick {; Form.Close()},;
    Text "&Close"  f.Open()
```

See Also
ShowSpeedTip, StatusMessage

TopMost

Property

Specifies whether forms display on top of all other forms

Property of class
FORM

Description
Use the TopMost property to determine if a form stays in the foreground while focus transfers to other windows.

For example, when an application displays an image object in its own form, it might be desirable to keep the image visible while the user gives focus to other forms. Assigning a

value of true (.T.) to the TopMost property keeps the form in the foreground regardless of which form has focus.

TopMost has an effect only when the MDI property is false (.F.).

Example

The following example displays a form containing an image on top of another form. The form with the image is visible even when the other form has focus.

```
f = new TOPMOSTFORM()
f2= new OTHERFORM()
f.open()
f2.open()
CLASS TOPMOSTFORM OF FORM
   this.Top = 18
   this.PageNo = 1
   this.Width = 50
   this.Text = "Modal TopMost"
   this.TopMost = .T.
   this.MDI = .F.
   this.Height = 15
   this.Left = 90
   DEFINE IMAGE IMAGE1 OF THIS;
       PROPERTY;
         Top 2,;
         PageNo 1,;
         DataSource "FILENAME C:\WINDOWS\LEAVES.BMP",;
         Width 30,;
         Alignment 3,;
         Height 10,;
         Left 11
ENDCLASS
CLASS OTHERFORM OF FORM
   this.Top = 4
   this.PageNo = 1
   this.Width = 87
   this.Text = "Other form"
   this.TopMost = .F.
   this.MDI = .T.
   this.Height = 26
   this.Left = 63
ENDCLASS
```

See Also

MDI, WindowState

TrackRight — Property

Determines if the user can select a popup menu item with a right mouse click.

Property of class
POPUP

Date type
Logical

Default
The default for TrackRight is true(.T.).

Description
When TrackRight is true (the default), users can select popup menu items with either the right mouse button or the left mouse button.

Set TrackRight to false if you don't want users to be able to select items from a popup menu with a right mouse click.

Example

```
f = NEW Form()
DEFINE POPUP p OF f;
  PROPERTY;
    TrackRight .F.
```

See Also
OnLeftMouseDown, OnRightMouseDown

WindowMenu — Property

Specifies a menu object that displays a list of all open MDI windows.

Property of Class
MENUBAR

Data type
Object reference

Description
WindowMenu contains a reference to a menu object that has a menubar as its parent. When users open this menu object, dBASE displays a pulldown list of all open MDI windows.

WindowMenu automatically places a separator line on the pulldown list between any menu prompts and the list of open windows. The currently active window shows a check next to the window name.

If you use the Menu Designer to create a menubar, WindowMenu is automatically set to an item named Window on the menubar:

```
this.WindowMenu = this.Window
```

Example

```
NEW SAMPLEMENU(FormObj,"Root")
CLASS SAMPLEMENU(FormObj,Name) OF MENUBAR(FormObj,Name)
   DEFINE MENU FILE OF THIS;
       PROPERTY;
         Text "&File"
         DEFINE MENU EXIT OF THIS.FILE;
             PROPERTY;
               Text "E&xit"
   DEFINE MENU EDIT OF THIS;
       PROPERTY;
         Text "&Edit"
         DEFINE MENU UNDO OF THIS.EDIT;
             PROPERTY;
               Text "&Undo"
         DEFINE MENU CUT OF THIS.EDIT;
             PROPERTY;
               Text "Cu&t"
         DEFINE MENU COPY OF THIS.EDIT;
             PROPERTY;
               Text "&Copy"
         DEFINE MENU PASTE OF THIS.EDIT;
             PROPERTY;
               Text "&Paste"
   DEFINE MENU WINDOW OF THIS;
       PROPERTY;
         Text "&Window"
         DEFINE MENU ARRANGE OF THIS.WINDOW;
             PROPERTY;
               Text "&Arrange"
   DEFINE MENU HELP OF THIS;
       PROPERTY;
         Text "&Help"
         DEFINE MENU ABOUT OF THIS.HELP;
             PROPERTY;
               Text "&About"
   This.EditUndoMenu  = This.Edit.Undo
   This.EditCutMenu   = This.Edit.Cut
   This.EditCopyMenu  = This.Edit.Copy
   This.EditPasteMenu = This.Edit.Paste
   This.WindowMenu    = This.Window
ENDCLASS
```

See Also

CLASS MENUBAR, EditCopyMenu, MDI

Chapter

8

Local SQL

Visual dBASE provides the ability to mix dBASE and SQL commands for operations against both local and remote data. This chapter describes the syntax of SQL commands that can be used within dBASE when working with non-database server data (i.e. dBASE and Paradox tables).

Note that the syntax described here is for use against dBASE/Paradox tables only. Database servers (such as Interbase or Oracle) have their own implementations of SQL syntax. When working with server data, the server's syntax must be used. For detailed information on SQL support in *Visual* dBASE, see the *Programmer's Guide*. For information on the SQL dialect and extensions used at your server, see your SQL server documentation.

Memory variable substitution in SQL queries

dBASE supports the substitution of memvar values in SQL queries. dBASE memory variables are indicated with a colon, as in the following example

```
x = "Robert"
SELECT * FROM customers WHERE firstname = :x
```

The memory variable x is resolved and "Robert" is substituted in its place.

Naming conventions

Table names

Table names may be comprised of alphanumeric characters, underscores (_), and the period (.). They may include full file and path specifications or BDE alias specifications (in the format :ALIASNAME:TABLENAME). They may even duplicate SQL keywords.

However, table names which include anything other than alphanumeric characters and underscores, or include file or alias specifications, must always be enclosed in single or double quotes. For example:

```
SELECT * FROM 'C:\SAMPLE.DAT\TABLE'                    includesfull path specification
SELECT * FROM "TABLE.DBF"                              includes a period
SELECT PASSID FROM "PASSWORD"                          duplicates an SQL keyword
SELECT BID_DATE FROM ":FSFDBASE:BIDS"                  includes BDE alias specification
```

Column names

Column names may be comprised of alphanumeric characters, underscores (_), and spaces ( ).They may also duplicate SQL keywords.

However, column names that include spaces or duplicate SQL keywords must always be:

- enclosed in single or double quotes
- prefaced with an SQL table name or table correlation name, in the format `TABLENAME.COLUMNNAME`.

For example:

```
SELECT E."EMP ID" FROM EMPLOYEE                        includes a space
SELECTDATELOG."DATE" FROM TABLE                        duplicates an SQL keyword
```

ALTER TABLE

Adds or drops (deletes) one or more columns (fields) from a table.

Syntax

ALTER TABLE <*table name*> ADD <*column name*><*data type*> | DROP <*column name*>
[, ADD <*column name*><*data type*> ...] [,DROP <*column name*>...]

Description

Use ALTER TABLE to modify the structure of an existing table. ALTER TABLE with the ADD clause adds the column <*column name*> of the type <*data type*> to <*table name*>. Use the DROP clause to remove the existing column <*column name*> from <*table*>.

Warning Data stored in a dropped column is lost without warning, regardless of the SET SAFETY setting.

Multiple columns may be added and/or dropped in a single ALTER TABLE command.

Use ALTER TABLE as a means of modifying the structure of a table without using the dBASE Table Structure dialog.

Example

The following statement adds two columns (REFER and LASTCALL) and deletes one column (FIRST_CONT) from a the table CUSTOMER.DBF:

```
ALTER TABLE CUSTOMER ADD REFER CHAR(20), ADD LASTCALL DATE, DROP FIRST_CONT
```

See also

CREATE TABLE, DROP TABLE, INSERT, MODIFY STRUCTURE (dBASE)

CREATE INDEX

Creates a new index on a table.

Syntax

CREATE INDEX <*index name*> ON <*table name*> <*column name*> [, <*column name*>...]

Description

Use CREATE INDEX to create a new index <*index name*>, in ascending order, based on the values in one or more columns <*column name*> of <*table name*>. Unlike the dBASE language, expressions cannot be used to create an index, only columns.

When working with dBASE .DBF tables, the index can only be created for a single column. The new index is created as a new index tag in the production index. A production index is created if it does not exist.

CREATE INDEX is equivalent to the INDEX ON <*field list*> TAG <*tag name*> syntax in the dBASE language.

Example

The following statement adds an index called ZIP on the ZIP_POSTAL column of the CUSTOMER.DBF table:

```
CREATE INDEX ZIP ON CUSTOMER ZIP_POSTAL
```

See also

DROP INDEX, INDEX (dBASE)

CREATE TABLE

Creates a new table.

Syntax

CREATE TABLE tablename (<*column name*> <*data type*> [,<*column name*> <*data type*>...])

Usage

Use CREATE TABLE to create a new table. The type of table produced (dBASE or Paradox) depends on the current setting of SET DBTYPE.

At least one <*column name*> <*data type*> must be defined. The column definition list must be enclosed in parentheses.

The following table lists SQL syntax for data types used with CREATE TABLE, and describes how they are mapped to dBASE and Paradox types.

Table 8.1 Data type mappings for CREATE[1]

SQL Syntax	dBASE	Paradox
SMALLINT	Numeric	Short
INT	Numeric	Long
DECIMAL(x,y)	N/A	BCD
NUMERIC(x,y)	Numeric(x,y)	Number
FLOAT(x,y)	Numeric(x,y)	Number
CHARACTER(n)	Character	Alpha
DATE	Date	Date
BOOLEAN	Logical	Logical
BLOB(n,s)[2]	Memo/Binary	Memo/Binary
TIME	N/A	Time
TIMESTAMP	N/A	TimeStamp
MONEY	Numeric(20,4)	Money
AUTOINC	N/A	Autoincrement
BYTES(n)	N/A	Bytes

1. Parameters
   x = length; if omitted, to 20 for dBASE
   y = decimal places; if omitted, defaults to 4 for dBASE
   n = length; if omitted, length defaults to 1
   s = subtype; if a blob subtype is omitted, the subtype defaults to Memo

2. Blob subtypes
   BINARY
   FMTMEMO (Paradox only)
   GRAPHIC (Paradox only)
   MEMO
   OLE

CREATE TABLE is a alternate way of creating a table without using the dBASE Table Structure dialog or the dBASE CREATE STRUCTURE EXTENDED, CREATE FROM commands.

Examples

The following example creates a dBASE table called SALES with the following structure:

Table 8.2 SALES.DBF structure

Field name	Field type	Field length	Decimal places
SALESID	Character	6	
CUSTOMERID	Character	10	
ORDERDATE	Date	8	
ORDERNMBR	Numeric	7	0

Table 8.2 SALES.DBF structure (continued)

Field name	Field type	Field length	Decimal places
ORDERAMT	Numeric	9	2
DELIVERED	Logical	1	

```
CREATE TABLE SALES (
   SALESID        CHAR(6),
   CUSTOMERID CHAR(10),
   ORDERDATE      DATE,
   ORDERNMBR NUMERIC(7,0),
   ORDERAMT       NUMERIC(9,2),
   DELIVERED      BOOLEAN)
```

See also
ALTER TABLE, CREATE (dBASE), CREATE FROM (dBASE), CREATE STRUCTURE EXTENDED (dBASE), DROP TABLE

DELETE FROM

Deletes rows (records) from a table.

Syntax
DELETE FROM <*table name*> [WHERE <*search condition*>]

Usage
Use DELETE FROM to delete rows, or records, from <*table name*>. Without the WHERE clause, all the rows in the table are deleted. Use the WHERE clause to specify a <*search condition*>. Only records matching the <*search condition*> are deleted.

When DELETE FROM is run against dBASE .DBF tables the following rules apply:

1. If a WHERE clause is used, DELETE FROM only **marks** rows for deletion, even if all the rows match the <*search condition*>. In this way, DELETE FROM behaves like the dBASE DELETE command. The rows are recallable unless the table is packed.
2. Without the WHERE clause, all the rows in the table are actually deleted. In this case, DELETE FROM behaves like the dBASE ZAP command. The rows are not recallable, and the table will have zero rows.

When DELETE FROM is run against a Paradox table, all the rows matching the <*search condition*> are actually deleted. If no WHERE clause is used, all the rows in the table are deleted. The data in the deleted rows in **not** recallable.

Example
The following example deletes all the rows in a dBASE table called CUSTOMER and results in a table with zero rows.

```
DELETE FROM CUSTOMER
```

The following example marks all the rows in a dBASE table called CUSTOMER for deletion, but does not actually delete the rows from the table.

```
DELETE FROM CUSTOMER WHERE CUSTOMER_N > 0
```

The following example marks all the rows where the CITY field is equal to "Freeport" for deletion in a dBASE table called CUSTOMER.

```
DELETE FROM CUSTOMER WHERE CITY = "Freeport"
```

The following example deletes all the rows where the CITY field is equal to "Freeport" in a Paradox table called CUSTOMER.

```
DELETE FROM "CUSTOMER.DB" WHERE CITY = "Freeport"
```

See also

DELETE (dBASE), PACK (dBASE), SELECT, ZAP (dBASE)

DROP INDEX

Drops (deletes) an existing index from a table.

Syntax

DROP INDEX <*table name*>.<*index name*>

Usage

Use DROP INDEX to drop, or delete, the index <*index name*> from <*table name*>. For dBASE .DBF tables <*index name*> must be the name of a tag in the production index.

Example

The following statement drops the index tag NAME from the production index of a dBASE table called EMPLOYEE:

```
DROP INDEX EMPLOYEE.NAME
```

The following statement drops a primary index on the Paradox table, EMPLOYEE.DB:

```
DROP INDEX "EMPLOYEE.DB".PRIMARY
```

See also

CREATE INDEX, DELETE TAG (dBASE), DROP TABLE

DROP TABLE

Drops (deletes) a table.

Syntax
DROP TABLE <*table name*>

Usage
Use DROP TABLE to delete the table <*table name*> from disk. The associated production index file and memo file, if any, are also deleted.

Example
The following statement drops a dBASE table call EMPLOYEE:

```
DROP TABLE EMPLOYEE
```

See also
CREATE TABLE, DELETE FROM, DELETE TABLE (dBASE)

INSERT INTO

Adds new rows (records) to a table.

Syntax
INSERT INTO <*table name*> [(<*column list*>)] VALUES (<*value list*>) | SELECT <*command*>

Usage
Use INSERT INTO to add rows, or records, to a table. There are two forms of this command. In the first form, you use <*value list*> to specify individual column values that are to be inserted for the new row. The values to be inserted must match in number, order, and type with the columns specified in <*column list*>, if <*column list*> is specified. Columns in the new row for which no value is given are left blank. If no <*column list*> is given, the order of the columns as they appear in the table is assumed. Without a <*column list*> a value must be provided for each column in the <*value list*>.

In the second form, the SELECT clause is executed just like a SELECT command. The row or rows returned by the SELECT are inserted into <*table name*>. The columns of the rows returned by the SELECT are matched up with the columns listed in <*column list*>. Therefore, the columns returned by SELECT must match in number, order, and type with the columns specified in <*column list*>, if <*column list*> is specified. If no <*column list*> is given, the number, order, and type of the columns returned by the SELECT must match the number, order, and type of the columns in <*table name*>.

Example
The following example makes a copy of the structure of the dBASE table CUSTOMER.DBF called CACUST.DBF, then adds customers to the new file.

```
USE CUSTOMER                        && open the customer table
COPY STRUCTURE TO CACUST            && copy the structure
USE                                 && close customer
* The next line adds a new row (record) to CACUST and inserts John Smith, Riverside, CA
* in the NAME, CITY, and STATE_PROV columns (fields) respectively
INSERT INTO CACUST (NAME, CITY, STATE_PROV) VALUES ("John Smith", "Riverside", "CA")
* The next line retrieves all the records where the state is CA from CUSTOMER and adds
them to CACUST
INSERT INTO CACUST SELECT * FROM CUSTOMER WHERE STATE_PROV = "CA"
```

See also

APPEND (dBASE), APPEND BLANK (dBASE), APPEND FROM (dBASE), COPY (dBASE), COPY TABLE (dBASE), CREATE TABLE, REPLACE (dBASE), SELECT

SELECT

Retrieves data from one or more tables.

Syntax

SELECT <*column list*> FROM <*table list*> [WHERE <*search condition*>] [GROUP BY <*column list*>] [ORDER BY <*column list*>] [HAVING <*search condition*>] [SAVE TO <*filename*>] [ALIAS <alias name>]

Usage

Use SELECT to retrieve data from a table or set of tables based on some criteria.

The <*column list*> is a comma-delimited list of columns in the table(s) that you wish to retrieve. The columns are retrieved in the order given in the list. If two or more tables used by SELECT use the same field names, distinguish the tables by using the table name and a dot (“.”). For example, if you’re SELECTing from the CUSTOMER table and the PRODUCT table, and they both have a field called NAME, enter the fields as CUSTOMER.NAME and PRODUCT.NAME in <*column list*>. To retrieve all the columns from <*table list*>, use an asterisk (*) for <*column list*>. To eliminate rows containing duplicate values within the same column, precede the <*column list*> with the keyword DISTINCT.

FROM <*table list*> The FROM clause specifies the table or tables from which to retrieve data. <*table list*> can be a single table or a comma-delimited list of tables.

WHERE <*search condition*> The optional WHERE clause reduces the number of rows returned by a SELECT to those that match the criteria specified in <*search condition*>.

GROUP BY <*column list*> The optional GROUP BY clause specifies how retrieved rows are grouped for aggregate functions. Any column names that appear in the GROUP BY <*column list*> must also appear in the SELECT <*column list*>.

ORDER BY <*column list*> The optional ORDER BY clause specifies the column to order the retrieved rows by.

HAVING <*search condition*> The optional HAVING clause specifies a <*search condition*> that evaluates to being true of false for each row in the group.

SAVE TO <*filename*> The optional SAVE TO clause specifies that the results of the SELECT are to be saved to a new table called <*filename*>.

ALIAS <alias name> The optional ALIAS clause specifies the name of the alias given to the work area the results of the SELECT appear in. If not specified, ALIAS defaults to SQL_<*integer*>.

The default output of SELECT statements produces an open work area (similar to USE). All dBASE commands and functions can be used on the result of a SELECT. If the query produced a temporary table, the temporary table is deleted when the answer set is closed. If you wish to save the results, use the SAVE TO option of SELECT or use the COPY TO command after the SELECT.

SELECT opens the answer set in the first unused workarea, starting from area 1. If the query was successful, the currently selected area is changed to the workarea where the answer set was produced.

Examples

The following examples show simple SELECTs:

```
SELECT NAME, PHONE FROM CUSTOMER WHERE STATE_PROV = "CA"
SELECT CUSTOMER_NO FROM CUSTOMER WHERE LAST_NAME = "Johnson"
SELECT PART_NO, SUM(QUANTITY) AS PQTY FROM PARTS GROUP BY PART_NO
```

The following example shows a join in which fields from each table are involved in some type of equality check require a WHERE clause:

```
SELECT DISTINCT PARTS.PART_NO, PARTS.QUANTITY, GOODS.CITY FROM PARTS, GOODS
   WHERE PARTS.PART_NO = GOODS.PART_NO AND PARTS.QUANTITY > 20
   ORDER BY PARTS.QUANTITY, GOODS.CITY, PARTS.PART_NO
```

The following example shows the use of the DESCENDING keyword in the ORDER BY clause. Note that in this case you must also specify DISTINCT.

```
SELECT DISTINCT CUSTOMER_NO FROM CUSTOMER ORDER BY CUSTOMER_NO DESCENDING
```

See also

CREATE QUERY(dBASE), DELETE FROM, INSERT FROM, SET FILTER (dBASE), SET KEY (dBASE), SET RELATION (dBASE), UPDATE

UPDATE

Adds or changes values in existing columns in existing rows of a table.

Syntax

UPDATE <*table name*> SET <*column name*> = <*expression*> [, <*column name*> = <*expression*>...]
WHERE <*search condition*>

Usage

Use UPDATE to update (change) values within existing columns in existing rows of a table. The column specified by <*column name*> is updated with the value of <*expression*>

in all rows that match the <*search criteria*> of the WHERE clause. If the WHERE clause is omitted, the column is updated in all rows in the table. Multiple columns may be updated in a single UPDATE command. A given column of a table may only appear once to the left of a "=" in the SET clause.

Example

The following command updates that YTD sales to zero for each customer that was contacted in the previous calendar year:

```
UDATE CUSTOMER SET YTD_SALES = 0 WHERE FIRST_CONT < {01/01/95}
```

See also

INSERT FROM, REPLACE (dBASE), SELECT

Chapter

9

Learning and extending *Visual* dBASE

Visual dBASE is a the most comprehensive and open Xbase environment available today. There are numerous resources to help you learn more about the many facets of the product. You can also extend *Visual* dBASE with companion products and tools such as the Local InterBase Server and VBX controls. Read on to discover sources of dBASE information beyond the documentation and how to harness the power of the open architecture.

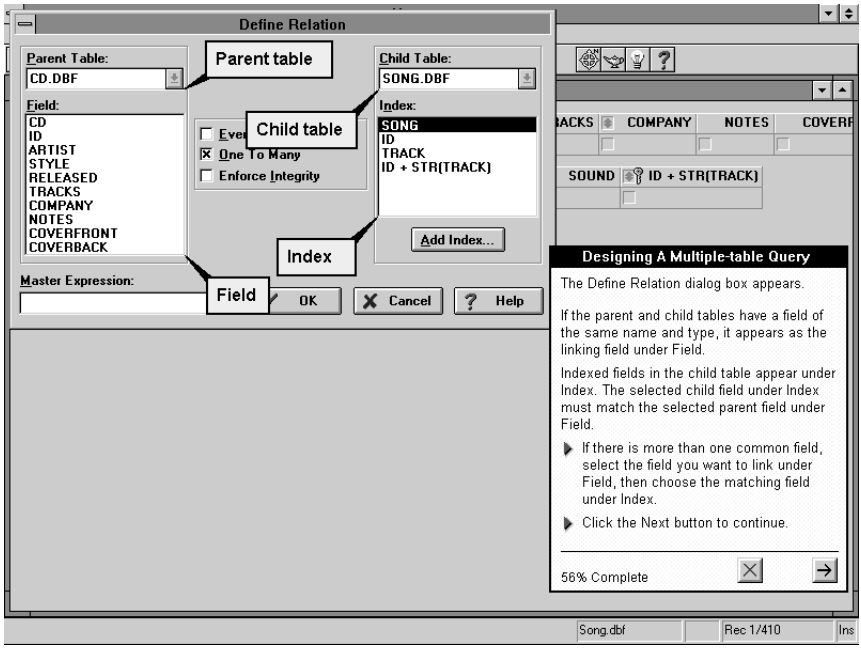
Learning more about *Visual* dBASE

Information on *Visual* dBASE is available in many forms to help you learn more about the product. *Visual* dBASE comes with Interactive Tutors, an extensive online help system and many sample programs. There are also videos, books and magazines dedicated to helping you learn all about the *Visual* dBASE environment as well as the dBASE language.

Interactive Tutors

Visual dBASE provides easy, step-by-step interactive tutors for learning how to create, use and manage tables, queries, forms and reports. This feature is especially useful for users who are new to dBASE as well as experienced dBASE power users that are migrating from DOS to Windows.

Figure 9.1 Learning with the Interactive Tutor



The Interactive Tutors work as both a traditional step by step guide and as a coach for specific tasks. You can work through the tutorial a lesson at a time with the sample tables or select a specific topic and have it guide you through a task using your own data. Figure 9.1 shows a portion of the lesson on joining tables in the Query Designer.

Making online Help work for you

Context-sensitive help is always a keystroke or a click away. Simply press *F1* or click on the Help SpeedButton to see help on the current window with focus. You will also find Help buttons in every visual property builder. If you are in the program editor, you can select any command and press *F1* to see help on that command.

The online help system includes the entire dBASE language and class reference. You can look up commands, functions, and classes alphabetically or by category. The online help system contains hundreds of examples that you can cut and paste into your own programs. Another quick way to get help is by typing HELP <topic> in the Command window.

Refer to online help for the latest information on syntax, classes, and properties. Like the README.TXT file, online help often contains more recent information than the printed documentation.

Learning by example

Over 100 sample files show how to accomplish common programming tasks using *Visual* dBASE. Many smaller tables, forms, queries, reports, and labels reside in the samples directory <visualdb >\SAMPLES. One larger application installs to <visualdb>\SAMPLES\MUSIC. This sample works with inventory data for a music store. Information and samples for creating call-back capable DLL files resides in <visualdb>\SAMPLES\EXTERN. See the EXTERN.TXT file for more information.

Online documentation

The CD-ROM version of *Visual* dBASE includes the complete documentation set in an online format. You can view and search any book using the online reader. Hypertext links let you jump directly to related topics. While the *Language Reference* is part of the standard online help system, the other books are provided with Adobe Acrobat.

Visual dBASE provides extensive documentation covering all features of the product. The *User's Guide* and *Using Crystal Reports for Visual dBASE* give power users detailed explanations of each visual tool. Developers can refer to the *Programmer's Guide* for information on developing sophisticated applications.

- *User's Guide* (Acrobat)
  Installation
  Introduction
  Borland Database Engine configuration
  Designing and using tables, queries, and forms
- Using Crystal Reports for *Visual* dBASE (Acrobat)
  Working with reports and labels
- *Programmer's Guide* (Acrobat)
  Programming
  Objects and Classes
  Event handlers
  Custom controls
  Data Manipulation Language
  Network programming
  Migrating DOS applications.
- *Language Reference* (Online Help)
  Commands
  Functions
  Classes
  Preprocessor
  SQL
  Reference charts

Books, magazines, and videos

In addition to the documentation and online help that comes with the product, there is a wealth of information covering all aspects of the dBASE language and environment. You can get in-depth information from books, the latest techniques from magazines and self-paced lessons on videos.

With strong support from the publishing community, there are currently over thirty books covering the dBASE language and environment. Here is a sampling of what is available.

Table 9.1 Books on dBASE

Books	ISBN	Author	Publisher
dBASE for Windows Handbook	679-79131-0	Cary Prague	Random House
dBASE for Windows Unleashed	0-672-305038	Ernest Escobar and Paul Mahar	Sams Publishing
dBASE for Windows Developer's Guide	0-672-30198	Tom Hovis	Sams Publishing
Learn dBASE for Windows Programming	0-201-60836-7	Martin L. Rinehart	Addison-Wesley
dBASE for Windows for Dummies	1-56884-179-5	Scott Palmer	IDG Books
Visual Guide to dBASE for Windows	1-56604-178-3	Carl Townsand	Ventana Press

The following table lists magazines dedicated to dBASE for Windows. In addition to these, Databased Advisor and DBMS often provide articles about dBASE tips and techniques.

Table 9.2 Magazines on dBASE

Magazines	Publisher	Phone
dBASE Advisor	Advisor Publications	(619) 483-6400
Inside dBASE for Windows	The Cobb Group	(502) 493-3300

Borland's Video Series offers two dBASE videos to introduce you to the dBASE language and visual interface.

Table 9.3 Videos on dBASE

Videos	Lessons
Learning dBASE for Windows	Navigator, Table Designer, Table Records Window, Query Designer, Form Designer, and Report Designer.
dBASE for Windows Programming	Covers stock classes and object oriented programming

Getting support

For more help, the Borland Assist program offers technical support plans to suit different users needs, from individual consultants to large corporations.

Standard support is available at no charge and includes an Installation Hotline, toll-free Automated Support, toll-free TechFax, a download BBS and online support forums on

CompuServe, the Internet, BIX, and Genie. The toll-free and online services are available 24 hours a day, seven days a week. The Install Hotline is available from 6:00 A.M. to 5:00 P.M. Pacific Time.

Borland also offers extensive support plans for individuals, corporations, and help desk professionals. Quick response time and consultative help are available from senior engineers on the Advisor Line. You can call the Advisor Line through a 900 number and an 800-number that can be billed to most major credit-cards.

Table 9.4 *Visual* dBASE Technical Support

Service	Number
Up and Running	(408) 461-9110
TechFax	1-800-822-4269
Enhanced Technical Support Information	1-800-523-7070
900 Advisor Line	1-900-555-1009
Credit Card Advisor Line	1-800-285-1118
Borland download BBS	(408) 431-5096
Automated Support BBS	(408) 431-5250

Table 9.5 Online Services with *Visual* dBASE Technical Support

Service	Address
Internet FTP site	borland.com
Borland World Wide Web	http://www.borland.com/
CompuServe	GO DBASEWIN
BIX	JOIN BORLAND
Genie	BORLAND

Expanding the power of Visual dBASE

Visual dBASE is an open environment. You can plug in new components as needed. With additional products, add-ons, and extensions, Borland gives you the tools you need to build robust, state-of-the-art, applications for Windows, Windows95, and Windows NT.

The *Visual* dBASE Compiler

To help developers distribute applications, Borland offers the *Visual* dBASE Compiler. The *Visual* dBASE Compiler includes the compiler technology and utilities you need to distribute your dBASE applications as easily installable royalty free EXE files. The *Visual* dBASE Compiler automatically integrates into the dBASE desktop and provides a help compiler and the Install Builder.

The *Visual* dBASE Compiler gives you an easy way to compile your dBASE applications into EXE files. The compiler can automatically descend through an application to find

and link in all relevant files without the need for a make or project file. The dBASE compiler also lets you specify an icon and a splash image.

The Help Compiler generates HLP files that can connect your dBASE applications to the Windows Help system. You can create context-sensitive help systems for your dBASE applications by setting the HelpID property of any control to a topic in your HLP file. You can use any editor that saves to RTF (Rich Text Format) to create your help system. RTF capable editors include Word, WordPerfect, and the Windows95 WordPad accessory.

Borland's application deployment technology completes the last step of the development process. For the first time, you can use the same installation and setup tools that Borland uses to create setup diskettes. The resulting diskettes include the famous 'freeway' installer customized for your own dBASE executables. The installation tools provide data compression, group creation, and options for installing the Borland Database Engine, Crystal Reports, and database drivers.

The Local InterBase Server

dBASE works directly with the Local InterBase Server giving you the ideal "offline" environment for creating client/server systems. You run the Local InterBase Server and dBASE on a single workstation to minimize development costs while using a high performance ANSI SQL-92 compliant server.

InterBase is an advanced server with support for stored procedures, triggers and constraints. InterBase is optimized for complex data including Binary Large Objects (BLObs) and multidimensional arrays.

You can leverage the power of InterBase to create fully scalable dBASE applications with server support on Windows NT, NetWare, and Unix. Local InterBase Server deployment kits are also available.

Client/Server connections

You can add SQL-Link drivers to the Borland Database Engine to connect to Oracle, Sybase, InterBase, Informix and Microsoft SQL Server. SQL-Link drivers work directly with the Borland Database Engine to give you performance unequaled by ODBC drivers.

Using VBX controls

To take advantage of many of the prepackaged VBX controls available today, Borland combined many of the industry's most popular and versatile controls together in one box. The Borland Visual Solutions Pack is a collection of more than 30 drop-in components that you can integrate into your *Visual* dBASE applications.

The Visual Solutions Pack includes controls for spreadsheets, WYSIWYG word processors, 3-D and 2-D charts, image editors, serial communication interfaces, animated buttons, gauges, sliders and a host of fun gadgets.

You can use Borland C++ to write and compile your own DLL and VBX files for extending *Visual* dBASE. Using the Borland C++ compiler insures that your DLL files will have complete call-back compatibility with dBASE. Your DLL files can self-register their functions with the dBASE environment.

Developing Windows Help

To simplify your development of Windows Help system, Borland offers ForeHelp. ForeHelp is a Windows development tool that helps you create Windows Help without getting into the intricacies of RTF files. ForeHelp works with the Help Compiler that comes with the *Visual* dBASE Compiler. For more information on ForeHelp call 1-800-628-9299.

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

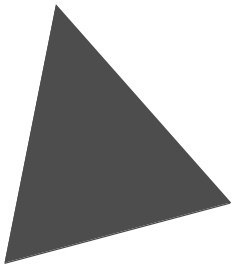
V

W

X

Z

Upgrade Guide



Visual dBASE®

Borland International, Inc., 100 Borland Way
P.O. Box 660001, Scotts Valley, CA 95067-0001

1E0R695
9596979899-987654321
D2

Contents

Figures